Baltie

podręcznik programowania
nie tylko dla dzieci

Rudolf Pecinovský
Jiří Vácha

Rudolf Pecinovský, Jiří Vácha

Baltie

podręcznik programowania nie tylko dla dzieci

Naszym żonom i dzieciom,
bez których nigdy byśmy tego nie byli w stanie zrobić

Spis treści

Wstęp

Witaj w świecie Baltiego, małego czarodzieja. W świecie, w którym możesz nauczyć się nie tylko programować, lecz także rozwiązywać bardziej efektywnie niektóre zagadki z codziennego życia.

Baltie jest programem, który w łatwy sposób wprowadza dzieci i dorosłych w świat programowania. Jest programem, który elegancko łączy klasyczne programowanie z współczesnym środowiskiem graficznym. Pomimo jego rosnącej popularności ciągle brakuje podręcznika, który pomógłby początkującym użytkownikom Baltiego w próbach programistycznych. Książka, którą trzymasz w rękach powinna wypełnić tę próżnię. Powinna być pierwszą z serii podręczników, w których chciałbym powoli zapoznać zarówno dzieci, jak i ich rodziców i nauczycieli z podstawami programowania. Cieszy mnie, że podczas pisania tej książki przeczytało ją z zainteresowaniem kilku dorosłych, początkujących programistów. Przekonali mnie, że dała im znacznie lepszy wstęp do świata programowania niż klasyczne podręczniki.

W przyszłości mam zamiar oprócz podręczników dla nowicjuszy przygotować podręcznik metodyczny dla nauczycieli (i rodziców) oraz bardziej szczegółowy podręcznik dla zaawansowanych pragnących nabyć specjalne umiejętności użyteczne przy ulepszeniu swoich programów.

W tej książce skoncentruję się przede wszystkim na podstawach programowania. W następnych książkach chciałbym zapoznać was z zaawansowanymi technikami programowania a przede wszystkim ze wspaniałymi możliwościami języka Baltie i jego środowiska. Pokazać, jak prosto i elegancko można zaprogramować rzeczy, które w innych językach trwałyby znacznie dłużej i wymagałyby znacznie więcej wiedzy i doświadczeń.

Kiedy tworzyłem pomoc do programu Baltie, otrzymałem teksty od programistów, którzy go stworzyli. Doprowadziłem pomoc do postaci, która wydała mi się maksymalnie dokładna i zrozumiała. Czas jednak pokazał, że przede wszystkim początkujący użytkownicy Baltiego mieli problemy z niektórymi częściami pomocy. Dlatego zaangażowałem do pisania podręcznika dwóch prywatnych pomocników – swojego ośmioletniego syna Michała i trzynastoletnią córkę Paulę. Oni przeczytali wszystko i próbowali sami programować korzystając z przeczytanego tekstu. Ich uwagi stały się impulsem do zmiany krytykowanych części książki tak, że zrozumieli je zarówno oni, jak ich rówieśnicy.

Podczas pisania skoncentrowałem się początkowo na podstawach programowania. Sądziłem, że czytelnicy zapoznali się już podstawami obsługi komputera. Potem jednak okazało się, że mnóstwo przyszłych programistów zapoznaje się z komputerem właśnie poprzez Baltiego. Dlatego umówiłem się z Jiřím Váchą, że napisze kilka rozdziałów o rzeczach, które powinniście znać zanim zaczniecie programować. Rozdziały o trybach działania *Budowanie* i *Czarowanie* jak i rozdział o podstawach pracy z edytorem *Paint* to przeważnie jego dzieło. Również podrozdział o stosowaniu pomocy i o programie *Tutor* to jego robota.

Kiedyś w młodzieńczym entuzjazmie napisałem wierszowany podręcznik programowania. Zanim zdążyłem go wydać, skradziono mi komputer z tym tekstem. Jednak niektóre rymowanki zapamiętałem (częściowo dlatego, że niektórych użyłem w książce *Podstawy programowania*). Ponieważ wierszowane reguły łatwiej zapamiętać od zwykłego tekstu spróbuję czasami w tekście podręcznika umieścić rymowanki. Mam nadzieję, że nie będziecie się na mnie za to gniewać.

Książka posiada mnóstwo najróżniejszych programów towarzyszących. Wiem, że przepisywanie programów z podręcznika jest stosunkowo nieprzyjemną czynnością, która nikogo nie bawi. By ułatwić wam wypróbowanie przede wszystkim bardziej skomplikowanych programów umieściliśmy wszystkie programy, które ja, Michał lub Paula stworzyliśmy i o których będzie mowa, na stronie internetowej ***www.baltik.cz***, skąd można je pobrać.

Jeśli Baltie stanie się waszym towarzyszem, nie zapominajcie czasami odwiedzić tę stronę. Oprócz ostatnich wersji Baltiego znajdziecie tu też klub programistów Baltiego, porady, zbiór różnych ciekawych programów Baltiego, konkursy z nagrodami i mnóstwo innych ciekawostek, które warto na tej stronie zobaczyć.

Jednym z ciekawych obiektów na tej stronie będzie elektroniczna wersja tej książki, która będzie miała formę, jaką znacie z pliku pomocy. Wersja elektroniczna powinna być w najbliższej przyszłości wzbogacona o możliwość bezpośredniego wgrania programu do środowiska Baltiego i inne małe ulepszenia. Na koniec tego wstępu mam do Was jedną prośbę: jeśli znajdziecie w książce błąd, coś będzie się wydawać niezrozumiałe lub będzie wam czegoś brakować, wyślijcie mi wiadomość na mój adres e-mail. Wszystkie uwagi spróbuję uwzględnić w wersji elektronicznej oraz, jeśli będzie kolejna wersja papierowa, także w niej.

Rudolf Pecinovský

rudolf@pecinovsky.cz

Co czytelnik powinien wiedzieć i umieć

Jak już napisałem jest to podręcznik wstępnym przeznaczony dla początkujących programistów. Jednak by zacząć przygodę z programowaniem trzeba przynajmniej trochę umieć posługiwać się komputerem. W następnym dlatego zakładam, że Czytelnik używał już kiedyś komputera i nie muszę wyjaśniać, co to takiego klawiatura lub mysz i jak się nimi posługiwać.

Jeśli zapragniecie zapoznać się bardziej z komputerami i nie chcecie kupować podręczniku, spróbujcie odwiedzić stronę internetową http://www.pecinovsky.cz/e-knihy. Po kolei będę na niej umieszczać elektroniczne wersje swoich książek, które będziecie mogli bezpłartnie pobrać.

Dalej zakładam, że oprócz podstaw użytkowania komputera dajecie sobie radę z pierwszymi dwoma stopniami, z trybami działania Baltiego *Budowanie* i *Czarowanie*. Powinno wystarczyć tyle wiadomości, ile pokazuje kurs wstępny *Tutor*. Dla powtórzenia i usystematyzowania tej wiedzy dołączyłem przed nauką programowania parę rozdziałów wprowadzających.

Książka ta jest naprawdę zwięzła. Dlatego nie zamierzam wyjaśniać tu rzeczy szczegółowo opisanych w pliku pomocy. Zamiast tego skoncentruję się na nauczeniu was programowania. Zamiast opisywania funkcji z pliku pomocy czasami po prostu odeślę was do niego.

Struktura książki

Jak właśnie napisałem, ta księga ma nauczyć was programować. Dlatego po kolei wyjaśnimy sobie wszystko, co powinniście jako początkujący programiści wiedzieć i umieć. Cała lektura podzielona jest na trzy części, które dotyczą pracy w różnych trybach działania.

Część 1: Preludium – Zanim zaczniemy programować

Zanim przejdziemy do samego programowania, przygotowałem dla was wspólnie z Jiřím Váchą małą rozgrzewkę. Najpierw ustalimy terminologię tak, by w dalszym tekście nazywać odpowiednie rzeczy odpowiednimi słowami i by nie korzystać z nazw, których nie znacie. Powiemy sobie, co to takiego okno aplikacji i jak będziemy nazywać jego poszczególne części. Potem wyjaśnimy sobie, co to takiego pliki i foldery i krótko powtórzymy podstawowe informacje na ich temat.

Następne trzy rozdziały zostaną poświęcone krótkiemu przedstawieniu najważniejszych informacji o trybach *Budowanie* i *Czarowanie* oraz o możliwościach, które oferuje nam edytor graficzny *Paint.*

Część 2: Zaczynamy programować – Działanie na poziomie Nowicjusz

Po rozgrzewce zaczniemy z pracę z samym programowaniem i po kolei pokażemy, co można zaprogramować na poziomie *Nowicjusz.* Najpierw nauczymy się tworzyć i zmieniać krótkie programy. Powiemy sobie o kilku zasadach, których przestrzeganie znacznie ułatwi tworzenie większych programów. Potem nauczymy się rozłożyć (fachowo mówi się *dekomponować*) złożone problemy na kilka problemów prostszych i po rozwiązaniu problemów prostych złożyć z powrotem rozwiązanie problemu pierwotnego.

Powiemy sobie też coś o *debugowaniu* - to jest proces, podczas którego programista szuka w swoim programie błędów i je usuwa. Pokażę wam, jak to robić szybciej. Pokażę także parę sztuczek, z których skorzystacie przede wszystkim podczas debugowania dużych programów lub programów przygotowanych w kilku wersjach.

Wyjaśnimy sobie, jak możecie wydrukować swój program, by pokazać go innym i nauczyć ich swoich sztuczek czy poprosić o radę. Na sam koniec nauczymy Baltiego, jak ma przedstawić na ekranie jedną prostą przygodę, w której ożywimy figurki i przedmioty. Ostatni rozdział poświęcony jest możliwościom ustawiania środowiska Baltiego.

Część 3: Programujemy naprawdę – Programowanie na poziomie Zaawansowany

Trzecia, największa część zostanie poświęcona programowaniu na poziomie *Zaawansowany.* Najpierw zapoznacie się z nowymi możliwościami dostępnymi na tym poziomie. Potem wrócimy do niektórych rzeczy omówionych w poprzednich częściach, które wyjaśnimy teraz dokładniej.

Po wstępnym wprowadzeniu zapoznamy się z literałami i wyjaśnimy sobie, co to takiego typy danych. Pokażemy jak

drukować na ekranie liczby i teksty. Działaniami na ekranie będziemy zajmować się w następnym rozdziale, w którym nauczymy się, co to takiego współrzędne, jak umieścić przedmiot lub tekst w konkretnym miejscu ekranu. Powiemy też o niektórych ciekawych współrzędnych.

Potem będziemy zajmować się jednym z największych wynalazków programowania: pomocnikami. Pokażemy sobie, jak z nich korzystać i jak z ich pomocą zbudować krótszy i czytelniejszy program. Kolejnym zagadnieniem będzie niemniej ważny obszarem programowania dotyczący zmiennych. Nauczymy programy zapamiętywać najróżniejsze dane, które otrzymają podczas swojej pracy i potem korzystać z nich. Będzie też mowa o zmiennych lokalnych występujących w pomocnikach i o ich podstawowym znaczeniu przy tworzeniu efektywnych i jasnych programów.

Przed następnym zestawem wiadomości pokażemy sobie, jak tworzyć większe programy i porozmawiamy o kilku przydatnych konwencjach, które pomogą nam skuteczniej szukać i usuwać ewentualne błędy w programach.

Po tej dygresji wrócimy do wyjaśniania następnych konstrukcji i zaczniemy uczyć nasze programy myśleć. Nauczymy się jak sprawdzać spełnienie pewnych warunków oraz jak zmusić program, by po spełnieniu lub niespełnieniu warunku zadecydował, co zrobić dalej.

Tak uzbrojeni odejdziemy nieco od czystego programowania i będziemy przez chwilę rozmawiać o tym, jak zmienić wygląd liczb i tekstów wysyłanych na ekran. Na koniec nauczymy się czytać i przetwarzać dane wprowadzane przez użytkownika z klawiatury.

Słownik wyrazów obcych

Chciałem tę książkę napisać tak, żeby mogły przeczytać ją nawet dzieci z trzeciej klasy. Ponieważ podczas nauki programowania nie obejdzie się bez kilku obcych słów dołączyłem na koniec książki jeszcze mały słowniczek wyrazów obcych użytych w książce. Słowa, o których sądziłem, że mogą być trudniejsze do zrozumienia, są wypisane w słowniku z krótkim wyjaśnieniem, co to słowo znaczy.

Użyte oznaczenia czyli konwencje typograficzne

Orientację w tekście będą wam ułatwiać różne elementy typograficzne:

Uwaga!	Ważne terminy i części tekstu, których trzeba podkreślić, są wydrukowane **czcionką tłustą.**
Nazwa	Nazwy firm, produktów oprogramowania, aplikacji i poszczególnych obiektów programu odznaczam *kursywą.*
Element	Nazwy elementów Baltiego i polecenia menu wydrukowane są **czcionką bezszeryfową**
Plik	Nazwy plików wydrukowane są ***czcionką bezszeryfową pochyłą***
Włóż → Ramę	Poszczególne polecenia w ciągu poleceń, wprowadzanych z menu, oddzielam strzałkami, np. **Narzędzia → Opcje → Widok → zawijać do okna**.
KLAWISZ	Nazwy klawiszy odznaczam KAPITALIKAMI – np. ENTER lub ALT+S.

W tekście często spotkasz się z żółtymi akapitami z ikoną, która charakteryzuje rodzaj informacji w tym akapicie:

Na początku każdego rozdziału umieszczony będzie akapit oznaczony pustym kwadracikiem. Tu dowiesz się, co cię czeka w tym rozdziale i czego w nim się nauczysz.

Na końcu każdego rozdziału znajdziesz akapit oznaczony odhaczonym kwadracikiem. W nim podsumujemy, czego nauczyłeś się w tym rozdziale i o co jesteś mądrzejszy..

Pisząca ręka oznacza notatkę, która nie jest konieczna dla rozwiązania problemu, lecz dotyczy tematu i wyjaśnia szerzej omawiane zagadnienia.

„Uśmiechnięta twarz ” ma zwrócić twoją uwagę na różne porady i sztuczki ułatwiające niektóre działania i pozwalające na osiągnięcie lepszych wyników.

Palec ostrzegający oznacza tekst, który mówi o czymś, na co powinieneś uważać, co może cię nieprzyjemnie zaskoczyć lub co mogłoby tworzyć problemy.

Bomba to znak katastrofy lub przynajmniej wielkich nieprzyjemności. Ta ikona oznacza tekst omawiający możliwości utraty danych, blokady systemu i innych nieprzyjemnych zdarzeń.

Część 1:

Preludium

Zanim zaczniemy programować

1. Zanim zaczniemy

Czego nauczymy się w tym rozdziale

W tym rozdziale wyjaśnimy sobie podstawowe terminy w sposób możliwie najbardziej zrozumiały. Każdy z nas uczył się nieco inaczej zasad obsługi komputera i dlatego może nazywać trochę inaczej te same rzeczy. Najpierw zatem ustalimy nazwy używanych przez nas pojęć. Przy okazji mniej doświadczeni użytkownicy dowiedzą się, co to jest plik i jak pliki są zorganizowane w foldery.

Po wyjaśnieniu podstawowych pojąć nauczymy się korzystać z pomocy Baltiego. Wyjaśnimy jak można uruchomić program *Tutor*, który jest bardzo cierpliwym nauczycielem podstawowych zasad pracy z programem Baltie.

1.1 Żeby wszystko zrozumieć – używane nazwy

Każdy z was inaczej uczył się posługiwania komputerem. Powinniśmy najpierw ustalić pewne nazwy (terminy), których będziemy używać w książce

Terminy używane przy pracy z myszą

Gdy w tekście powiemy, że mamy na coś **wskazać** myszą, trzeba będzie przesunąć wskaźnik myszy (powinien mieć wygląd strzałki) tak, żeby wskazywał na żądane miejsce na ekranie.

Określenie **kliknąć myszą** oznacza, że mamy na obiekt wskazać myszą po czym nacisnąć i zwolnić lewy przycisk myszy.

Jeśli powiemy, żeby coś **chwycić,** trzeba na ten obiekt wskazać myszą, nacisnąć lewy przycisk myszy i trzymać go.

Jeżeli mamy coś **przesunąć,** chwytamy obiekt myszą, przesuwamy wskaźnik myszy z chwyconym przedmiotem na nowe miejsce i tam upuszczamy obiekt puszczając lewy przycisk myszy.

Gdy powiemy, żeby gdzieś **kliknąć dwukrotnie**, oznacza to, że należy na to miejsce wskazać i dwa razy szybko kliknąć. Między tymi kliknięciami nie wolno przesunąć myszy, bo w tym przypadku komputer potraktuje kliknięcia jako dwa pojedyncze. Tak samo nie wolno między kliknięciami robić za długich przerw. Maksymalny czas między kliknięciami można ustawić w *Windows* w *Panelu Sterowania*.

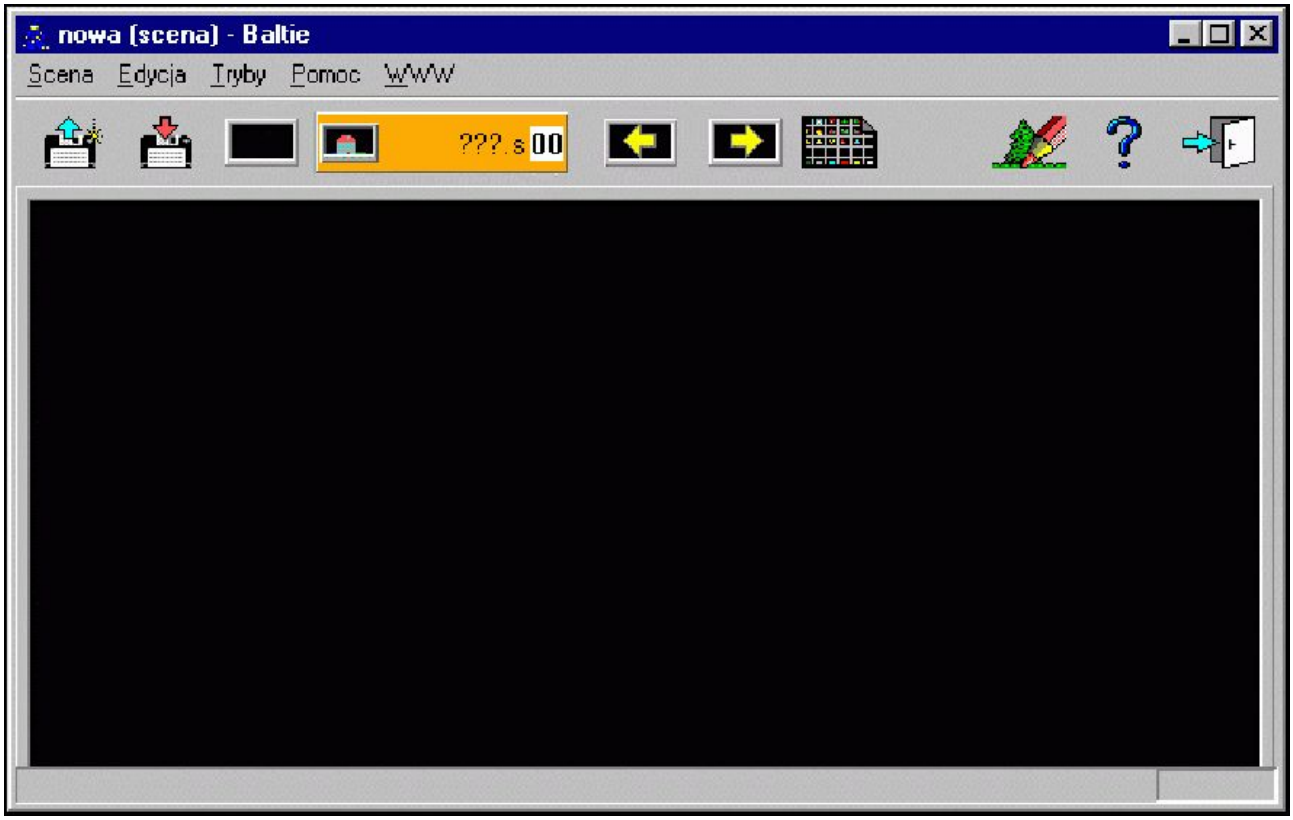
Gdy powiemy, że trzeba **nacisnąć** jakiś **przycisk** na ekranie należy kliknąć myszą na tym przycisku. Niektóre przyciski nie reagują na zbyt szybkie kliknięcie. Jeśli tak stanie się wystarczy przytrzymać przycisk myszy trochę dłużej.

Jeśli będziemy musieli coś **kliknąć** lub coś chwycić **prawym przyciskiem,** zrobimy to samo, co przed chwilą, tylko zamiast lewego przycisku myszy użyjemy prawego.

Nazwy obiektów na ekranie

Po uruchomieniu Baltiego otworzy się **okno aplikacji**. Jest to część ekranu, którą Baltie rezerwuje dla siebie i rejestruje nasze działania na tym obszarze.

Okno aplikacji ma kilka części. Omówimy teraz ich nazwy (patrz rysunek 1.1).



Rysunek 1.1
Okno aplikacji Baltiego w trybie Budowanie

- Pasek przy górnej krawędzi okna (na rysunku jest w nim napisane **nowa scena - Baltie**) to **nagłówek okna.**
- Pasek pod nim z napisami **Program | Edycja | Tryby | Możliwości | Pomoc** (dalej będziemy używać tych nazw bez podkreślenia liter) to **menu główne**. Każdy napis oznacza polecenie rozwijające jeden wykaz dostępnych poleceń, czyli właśnie **menu**. Kliknięciem na ten napis rozwiniemy menu i możemy z niego wybrać kolejnym kliknięciem polecenie. Na rysunku 2.1 zostało wybrane polecenie **Budowane sceny** z menu **Tryby**. Kolejność kliknięć będziemy zapisywać strzałkami pomiędzy poszczególnymi poleceniami. Kolejność z rysunku 2.1 zapiszemy jako **Tryby** → **Budowanie**.
- Pasek pod menu głównym to **panel narzędzi. N**aciśnięciem wybranego przycisku uruchomimy wybrane narzędzie.
- Duży czarny prostokąt pod panelem narzędzi to **obszar roboczy.** Na nim będziemy tworzyć nasze prace.
- Szary pasek na dolnej krawędzi okna aplikacji to **pasek stanu.** Tu Baltie wyświetla dodatkowe informacje a przede wszystkim prostą podpowiedź do tego, co w danej chwili robimy.

1.2 Pliki i praca z nimi – dla początkujących

Komputer powinien mnóstwo rzeczy zapamiętać żeby stać się naszym dobrym pomocnikiem. Powinien móc zapamiętywać informacje nawet po wyłączeniu. Komputerowi dajemy do zapamiętania najróżniejsze rzeczy: teksty, rysunki, programy, filmy i inne. Komputer może przechować to, nad czym pracujemy, zapisując wszystko do tak zwanych **plików**.

Plik

Plik można sobie wyobrazić jako okładkę na wszystko, co chcemy przechować w komputerze. Komputerowi wszystko jedno, czy przechowujemy swój życiorys, zdjęcia z wakacji czy program do obliczenia wygrywającej kombinacji w totolotku. Wszystko co ma przechować, włoży do okładki –**pliku** – i ten plik zapisze na dysku.

Aby komputer mógł orientować się w okładkach (plikach), każda okładka (każdy plik) musi mieć swoją nazwę, którą sami musimy wpisać. Istnieją pewne ograniczenia dotyczące nazw plików. Wystarczy zapamiętać, że nazwa, która składa się tylko z liter, cyfr, odstępów i myślników, zawsze będzie poprawna. Jeśli użyjemy jakiegoś niewłaściwego znaku, nic złego się nie stanie. Komputer nie pozwoli wpisać takiej nazwy.

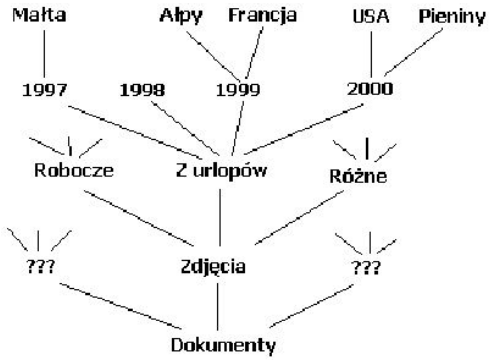
Nazwy plików należy dobierać tak, żeby móc szybko przypomnieć sobie, co w tym pliku jest zapisane. Jeśli będziemy nazywać pliki różnymi skrótami i szyframi będziemy później mieć problemy ze znalezieniem potrzebnych informacji.

W rzeczywistości nazwa pliku w komputerze składa się z dwóch części: z nazwy i z **rozszerzenia,** tzn. z części nazwy za ostatnią kropką. Rozszerzenie jest przeważnie trzyliterowe. Określa ono typ pliku. Porównując rozszerzenia z własnym wykazem typów komputer zgaduje, co plik zawiera i co w takim razie powinien z nim zrobić. O ile nazwa pliku zależy tylko od nas to rozszerzenia nie wolno wybierać dowolnie. Najczęściej program, którego używamy zapisując plik, wybierze rozszerzenie automatycznie. Czasami zapisując plik można wybrać spośród kilku rozszerzeń. Kiedykolwiek w tej książce będzie potrzeba wiedzieć coś więcej o rozszerzeniach, zawsze to w porę wyjaśnimy.

Chociaż *Microsoft* twierdzi, że rozszerzenie to część nazwy pliku za ostatnią kropką, spotkałem się z sytuacjami, gdy niektórzy programiści zapomnieli o tym i ich program uważał za rozszerzenie wszystko, co było za pierwszą kropką. Bywa też tak, że użytkownik zakończy nazwę pliku kropką. Program może to zrozumieć jako brak rozszerzenia w nazwie pliku. Jeśli nie chcemy być zaskakiwani dziwnym zachowaniem *Windows*, **nie używajmy kropki w nazwach plików ale używajmy jej tylko do oddzielenia nazwy pliku od rozszerzenia**.

Folder

Ponieważ jest w komputerze bardzo dużo plików (bez problemu można znaleźć w komputerze nawet kilkadziesiąt tysięcy plików), dobrze jest je uporządkować. Robi się to tak jak w normalnym życiu. Wyobraźmy sobie, że porządkujemy zdjęcia – każde wkładamy do osobnej okładki (pliku). Okładki wkładamy do pudełka. Jeśli zdjęć będzie więcej, podzielimy okładki na kilka pudełek według rodzaju zdjęć. Pudełka włożymy do szafy. Zbieranie zdjęć zacznie być zabawne i za chwilę mamy pełną szafę fotografii. Wkrótce zapełniamy pudełkami kilka szaf, więc wydzielamy osobne pomieszczenie itd. Tak samo robimy w komputerze, tylko że pliki wkładamy nie do pudełek, lecz do **folderów**. Folder to w rzeczywistości plik do przechowywania plików. Folderów nie wkładamy do szaf, lecz do innych folderów możemy nazywać je np. rodzicielskimi.



Rysunek 1.2
Drzewo folderów ze zdjęciami

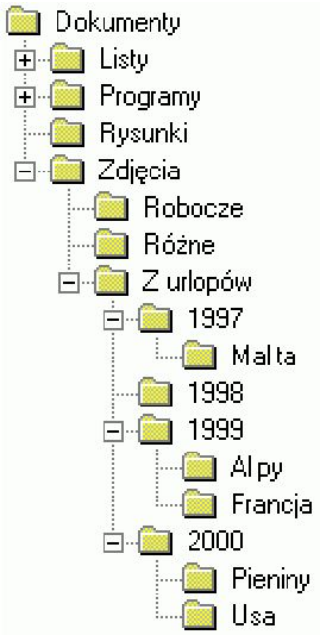
Spróbujmy pokazać to na przykładzie. Powiedzmy, że jesteśmy kolekcjonerami zdjęć, które chcemy przechowywać w komputerze. Swoje zdjęcia możemy podzielić np. tematycznie na zdjęcia robocze, wakacyjne, wycieczkowe i inne. Zdjęcia z wakacji można podzielić według lat, w których je zrobiliśmy. Każdy rok podzielimy według miejsca, gdzie zrobiliśmy zdjęcia. Taki podział możemy wyświetlić graficzne tak jak na rysunku 1.2.

Dostaniemy ciekawą strukturę, którą matematycy nazywają **drzewo**. Miejsce, z którego wychodzą wszystkie gałęzie, to **korzeń**. Korzeniem naszego drzewa jest folder nazwany ***Moje Dokumenty***. To folder, do którego powinniśmy zapisywać wszystko, co tworzymy. Folder ***Moje Dokumenty*** ma kilka pod-folderów, między którymi jest i folder ***Zdjęcia,*** w którym przechowujemy wszystkie zdjęcia. Folder ***Zdjęcia*** zawiera trzy pod-foldery. Na rysunku skoncentrowaliśmy się tylko na folderze ***Z urlopów*** (pod-foldery innych folderów nas teraz nie obchodzą). Opis drzewa na pewno każdy dokończy bez trudu.

Komputer wyświetla strukturę folderów trochę inaczej – nasze drzewo widać na rysunku 1.3. Foldery są porządkowane według alfabetu. Folder, który zawiera inne pod-foldery, ma na lewo od ikony folderu mały kwadracik ze znakiem + albo –.

- Znak + oznacza, że pod-foldery nie są wyświetlone (na naszym rysunku np. foldery ***Listy*** i ***Programy***). O takim folderze można powiedzieć, że jest **zwinięty**.
- Znak – odznacza, że w tej chwili wyświetlone są wszystkie podfoldery tego folderu. Mówimy, że folder jest **rozwinięty**.

Jak już powiedzieliśmy wszystkie wytworzone dokumenty powinniśmy zapisywać w komputerze do folderu ***Moje Dokumenty*** lub do jego pod-folderów. Żeby nie mieszać plików Baltiego z innymi plikami czy nawet z plikami innych użytkowników tego komputera, przydałby się do pracy z Baltiem osobny folder. Może się nazywać np. ***Baltie.*** Jeśli używamy komputera wspólnie z innymi użytkownikami, warto założyć w folderze ***Moje Dokumenty*** najpierw swój własny pod-folder (nazwany najlepiej swoim imieniem) a w nim dopiero pod-folder ***Baltie.***



Rysunek 1.3
Drzewo folderów – widok w Windows

Te informacje są prawdziwe dla większości komputerów. W niektórych systemach użytkownicy mogą mieć inaczej uporządkowane drzewo folderów. Pracując przy komputerze, którego organizacją wewnętrzną ktoś się opiekuje, trzeba zwrócić się do administratora z prośbą, żeby przygotował nam prywatny folder ***Moje Dokumenty*** i pokazał, jak z niego korzystać.

Do tego tematu wrócimy jeszcze w rozdziale ***Sceny i pliki***, kiedy będziemy opowiadać o zapisywaniu programów i scen do plików.

1.3 Pomoc

Jeżeli zdarzy się, że będziemy mieć wątpliwości dotyczące obiektu, z którym pracujemy, warto zerknąć na **pasek stanu**. Prawie zawsze znajdziemy tam krótką poradę dotyczącą obiektu, na który właśnie wskazujemy myszą. Jeśli będziemy chcieli dowiedzieć się o więcej na jakiś temat, przyda się polecenie **Pomoc → Spis treści.** Okno pomocy otworzy się na zakładce *Spis treści - skrócony*. Klikamy na tekście *Jak korzystać z tej pomocy*. Otworzy się strona z wyjaśnieniami, jak posługiwać się pomocą.

Pomoc kontekstową, czyli wyjaśnienia do obecnie wykonywanej operacji, można otworzyć naciśnięciem przycisku **Pomoc** w panelu narzędzi (przycisk z wielkim znakiem zapytania) lub klawisza **F1.** Inną możliwością otrzymania pomocy kontekstowej jest upuszczenie trzymanego elementu nad przyciskiem **Pomoc.** W ten sposób możemy dowiedzieć się więcej o elemencie, który upuścimy.

Polecamy uważne przeczytanie tematu *Jak korzystać z tej pomocy* oraz wypróbowanie opisanych powyżej działania. W ten sposób można znaleźć odpowiedzi na większość pytań pojawiających się podczas pracy z Baltie. **Używanie pomocy powinno zatem stać się normalną praktyką**. Środowisko Baltiego ma tyle możliwości, że trudno wszystko zapamiętać.

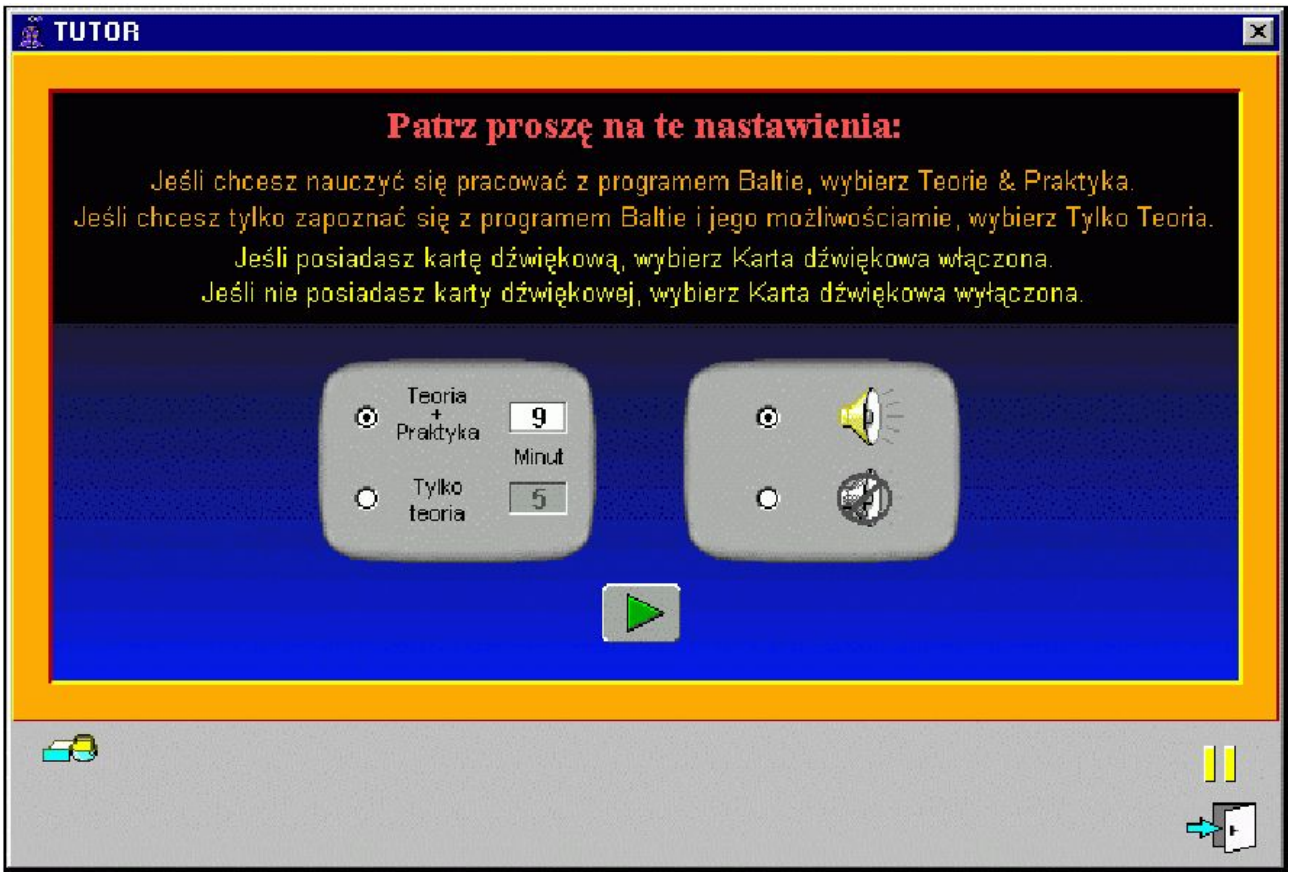
Przewodnik Tutor

Jeśli mamy zainstalowaną pełną wersję Baltiego, możemy skorzystać z przewodnika po całym jego środowisku. Wystarczy uruchomić program *Tutor,* który wyjaśni podstawy pracy we wszystkich trybach Baltiego. Mając komputer wyposażony w głośniki możemy nie tylko zobaczyć ale i posłuchać przewodnika.

Program *Tutor* pozwoli też wypróbować niektóre z omawianych działań. Przy okazji możemy sobie choć trochę

wyobrazić, co oznacza programowanie w Baltie, bo cały program *Tutor* jest napisany w Baltie. Program *Tutor* uruchamiamy w następujący sposób:

1. Po uruchomieniu Baltie wprowadzamy polecenie **Pomoc → Audiowizualny program uczący.** Program *Tutor* zostanie uruchomiony automatycznie.
2. Program przedstawi się oknem wstępnym, po którym pojawi się ekran opcji, który widać na rysunku 1.4. Na tym ekranie program zażąda wprowadzenia niektórych informacji dodatkowych.
   - Jeśli nie mamy karty dźwiękowej, wybieramy opcję **Karta dźwiękowa wyłączona** (przekreślony głośnik) wskazując myszą na białe okrągłe pole przed symbolem lub opisem wybranej opcji i klikając lewym przycisk myszy.
   - Jeśli nie chcemy ćwiczyć a tylko spokojnie zobaczyć możliwości Baltie, wybieramy opcję **Tylko teoria.**
3. Przesuwamy wskaźnik myszy na przycisk **OK** i naciskamy lewy przycisk myszy. Przewodnik rozpocznie działanie a my możemy obserwować, co dzieje się na ekranie. Po zakończeniu pracy z programem *Tutor* powiemy więcej o systemie pomocy.



Rysunek 1.4
Ustawianie opcji programu Tutor

1.4 Czego nauczyliśmy się

W tym rozdziale wyjaśniliśmy sobie podstawowe terminy, aby przy dalszej nauce lepiej się rozumieć. Przy okazji mniej doświadczeni użytkownicy dowiedzieli się co to jest plik i folder oraz jak jest w komputerze przedstawiane jest drzewo folderów. Po wyjaśnieniu terminów nauczyliśmy się posługiwać się systemem pomocy Baltie i pokazaliśmy, jak można uruchomić program *Tutor*.

2. Tryb Budowanie

Czego nauczymy się w tym rozdziale

W tym rozdziale wyjaśnimy, co to jest Scena, Przedmiot i Bank Przedmiotów. Nauczymy się budować scenę, zapisać ją do pliku i wczytywać z pliku zapisaną scenę. Porozmawiamy też o tym, jak uniknąć problemów rozpoczynając pracę z nową sceną.

Tryb *Budowanie* jest najprostszym trybem działania Baltie. W tym trybie możemy przygotować rysunek, który w Baltie nazywa się **scena**. Tworząc sceny można przećwiczyć obsługę komputerowej myszy.

2.1 Co to jest Scena, Przedmiot, Bank Przedmiotów

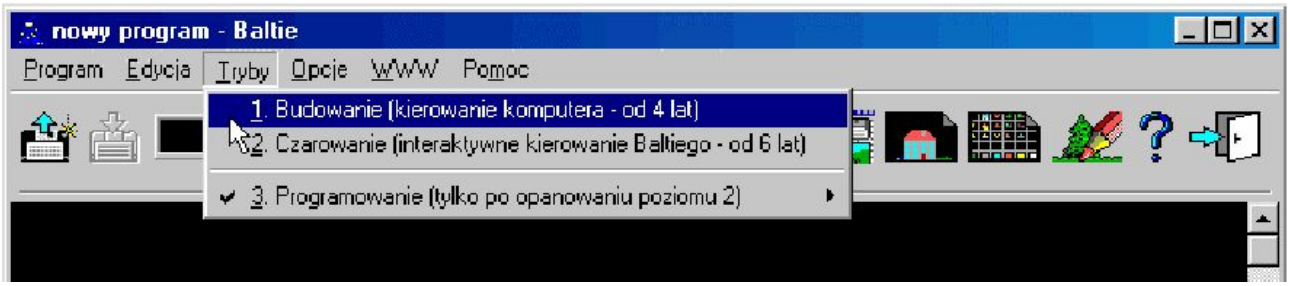
Scena to obszar, na którym tworzymy rysunki. Te rysunki układa się z niewielkich części, tak jak układa się rysunki z klocków z fragmentami rysunków. Pojedyncze części nazywamy **przedmiotami.** Na scenie mieści się razem 150 przedmiotów ustawionych w dziesięć wierszy po piętnaście przedmiotów.

Przedmioty to małe prostokątne rysunki, które Baltie przechowuje w specjalnych przechowalniach, które nazywają się **bankami**. Każdy bank (poprawnie należałoby mówić *bank przedmiotów*) przechowuje 150 przedmiotów, które są (tak jak w scenie) ułożone do 10 wierszy po 15 przedmiotów (patrz rysunek 2.3).

Sceny i banki są w komputerze przechowywane (oczywiście) w plikach. Sceny zapisywane są w plikach, których nazwy różnią się tylko rozszerzeniem. Pliki ze scenami tworzą grupy, które mogą mieć do 100 członków. Podobne grupy tworzą banki. W ich grupach może być do 200 członków. O tym, jak tworzyć takie grupy, powiemy podczas wyjaśniania sposobu zapisywania scen i banków do plików.

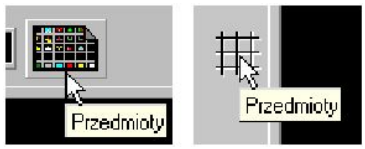
2.2 Budujemy scenę

Po pierwszym uruchomieniu Baltie powinien przejść automatycznie do trybu pracy *Budowanie,* w którym jego okno aplikacji powinno wyglądać tak, jak okno na rysunku 1.1. Jeśli Baltie był już wcześniej uruchomiony, może otworzyć się w innym trybie. Należy więc sprawdzić wygląd okna aplikacji i jeżeli wygląd panelu narzędzi różni się od tego z rysunku 1.1 wprowadzamy polecenie **Tryby** → **Budowanie** (patrz rysunek 2.1). W ten sposób przełączymy się do trybu *Budowanie.*



Rysunek 2.1
Przełączanie trybów

Teraz już możemy zacząć budować scenę. Jak już powiedzieliśmy, scenę budujemy z przedmiotów, które są przechowywane w bankach. Przedmiot, który chcemy umieścić na scenie musimy wybrać z konkretnego banku. Banki włączymy klikając na przycisk **Przedmioty** (rysunek 2.2 na lewo) lub gdziekolwiek na szarą przestrzeń wokół sceny, gdzie wskaźnik myszy zmienia się ze strzałki na siatkę (rysunek 2.2 na prawo). W ten sposób otworzymy okno **Przedmioty** (rysunek 2.3), w którym dostępne są wszystkie banki z jednej grupy. Zawsze widzimy tylko jeden bank – nazwijmy go **bieżący bank**. Jego zawartość zajmuje prawie całe okno. Na dole pod obrazem zawartości bieżącego banku przedmiotów są zakładki z numerami dostępnych banków (tzn.



Rysunek 2.2
Otworzenie okna Przedmioty

z numerami poszczególnych banków z grupy). Bieżący bank ma zakładkę w kolorze szarym (na rysunku 2.3 bieżącym bankiem jest bank nr 0).

Bieżący bank można zmienić na jeden z dwóch sposobów:

– kliknięciem na zakładkę ustawimy jako bieżący bank z konkretnym numerem,

– kliknięciem na jeden z przycisków z żółtą strzałką, które można znaleźć w środku szarej przestrzeni, nastawimy jako bieżący poprzedni lub następny bank.

Po wybieraniu przedmiotu, który chcemy umieścić na naszej scenie, klikamy na niego myszą. W ten sposób chwycimy go, okno **Przedmioty** zamknie się i wrócimy do sceny trzymając przedmiot. Na scenie umieścimy go przesuwając wskaźnik myszy na miejsce, gdzie chcemy umieścić przedmiot i ponownie klikając upuścimy przedmiot na wybrane miejsce.

Powiedzieliśmy, że upuścimy przedmiot na wybrane miejsce. Łatwo zauważyć, że przed upuszczeniem przedmiot nie musi być dokładnie na swojej przyszłej pozycji, lecz można go upuścić w pobliżu wybranego miejsca. Po upuszczeniu przedmiot zostanie dopasowany do właściwej pozycji wynikającej z układu elementów sceny – 10 wierszy po 15 przedmiotów.

Rysunek 2.3
Okno Przedmioty z bieżącym bankiem nr 0

Jeśli nie trafimy przedmiotem na właściwe miejsce, nic się złego nie stanie. Jeśli potrzebujemy przesunąć przedmiot na inne miejsce, wskazujemy na niego (rys. 2.4 na lewo) i chwytamy naciśnięciem lewego przycisku myszy. Wskaźnik myszy zmieni się ze strzałki na rękę (rys. 2.4 w środku) i dopóki trzymamy naciśnięty przycisk myszy, dopóty możemy przedmiot dowolnie przesuwać. Zwalniając przycisk myszy umieścimy przedmiot w nowym miejscu.

Rysunek 2.4
Przesuwanie przedmiotu

Często chcemy mieć na scenie w kilku miejscach ten sam przedmiot – np. przedmiot „ściana z oknem” w wielkim domu, który ma kilka okien. Nie musimy za każdym razem sięgać do banku, możemy „rozmnożyć” przedmiot wprost na scenie. Zrobimy to tak jak przy przesuwaniu przedmiotu, tylko zamiast lewego przycisku myszy użyjemy prawego. Spowoduje to przesuwanie kopii przedmiotu. Na wskaźniku w kształcie ręki będzie przy tym widoczny znak plus (rys. 2.4 na prawo).

Rysunek 2.6
Wyczyszczenie sceny

Jeśli chcemy usunąć ze sceny jakiś przedmiot, możemy chwycić go lewym przyciskiem myszy i przesunąć na szarą przestrzeń wokół sceny. Gdy na przedmiocie pojawi się szare przekreślenie (patrz rysunek 2.5) możemy go upuścić.

Gdy chcemy wyczyścić całą scenę, bo nam się nie podoba rysunek, nie musimy usuwać pojedynczych przedmiotów. Wystarczy w panelu narzędzi nacisnąć przycisk **Wyczyść scenę** (patrz rysunek 2.6).

Rysunek 2.5
Usuń przedmiot

2.3 Sceny i pliki

W tym rozdziale zajmiemy się przede wszystkim zapisywaniem sceny do pliku. Informacje te są przeznaczone głównie dla osób pracujących z pełną, zarejestrowaną wersją programu. Jeśli używasz wersji demonstracyjnej, możesz ten rozdział pominąć. Wersja demonstracyjna różni się od wersji pełnej tylko tym, że nie umożliwia zapisywania plików.

Jeśli utworzymy jakąś piękną scenę, na pewno zechcemy ją zapisać aby można ją było później użyć w programie lub komuś pokazać. Popatrzmy, jak to zrobić.

Przygotowanie folderu dla naszych prac

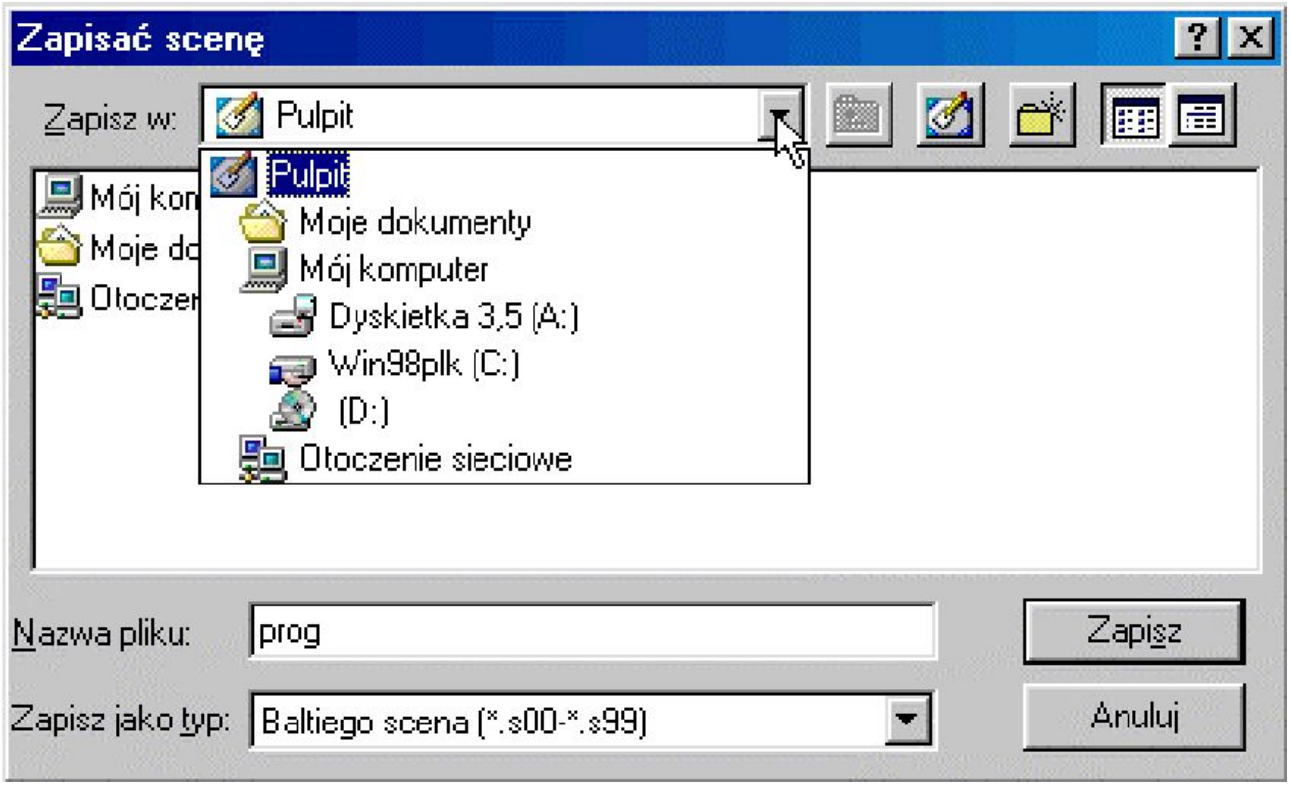
Zanim przejdziemy dalej przypomnijmy sobie, o czym mówiliśmy na końcu części o folderach:

Wszystko co tworzymy, zapisujemy do folderu *Moje Dokumenty* lub do jego pod-folderów.

Ustaliliśmy też, w jaki sposób można zorganizować swoje foldery. Dalej opiszemy sposób przygotowania własnego drzewa folderów:

Ten opis dotyczy typowego komputera wyposażonego w system operacyjny *Windows 98* lub *Windows 2000/XP*. Jeżeli komputer zachowuje się inaczej, niż opisano to w kolejnych akapitach, może być konieczny kontakt ze specjalistą.

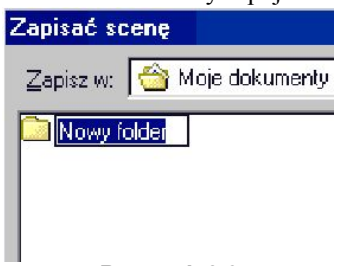
1. Wprowadźmy polecenie **Scena→ Zapisz jako.** Otworzy się okno dialogowe **Zapisać scenę.**
2. Pod nagłówkiem okna znajduje się pole z nazwą folderu, do którego Baltie chce zapisać pliki. Kliknijmy na przycisku ze strzałką na dół, które znajduje się po prawej stronie tego pola. Rozwinie się obraz struktury drzewa folderów w komputerze (patrz rysunek 2.7 – wskaźnik myszy wskazuje na nim przycisk, który należało nacisnąć).



Rysunek 2.7
*Okno dialogu **Zapisać scenę** z rozwiniętym fragmentem drzewa folderów*

1. Na rysunku przedstawiającym strukturę drzewa klikamy na pozycję z folderem ***Moje Dokumenty***. W środku okna zostanie wyświetlona lista plików i folderów, które znajdują się w folderze ***Moje Dokumenty***.
2. W panelu narzędzi klikamy przycisk **Utwórz folder** (patrz rysunek 2.8). W obszarze roboczym pojawi się nowy folder nazwany **Nowy folder.** Nazwa folderu pozostanie zaznaczona i system oczekuje na jej zmianę.

Rysunek 2.8
Twórz folder



Rysunek 2.9
Nowy folder oczekuje na zmianę nazwy

3. Wprowadzamy nazwę, którą ma nowy folder (na przykład ***Ruda***) i naciskamy ENTER. Nazwa została wprowadzona.
4. Dwukrotnie klikamy na ikonie folderu – otwieramy go.

Jeśli system zamiast otworzenia folderu wyświetla komunikat, że nie może znaleźć jakiegoś elementu (patrz rys. 2.10), najczęściej wystarcza ponownie spróbować otworzyć folder lub nacisnąć klawisz F5, co w *Windows* oznacza odświeżenie zawartości okna.

Rysunek 2.10
Fałszywa wiadomość o nie istniejącym folderze

5. Wracamy do punktu 2 i zakładamy w swoim folderze pod-folder ***Baltie.***

Teraz już mamy wszystko przygotowane do zapisywania efektów swojej pracy.

Zapisujemy utworzoną scenę

Mamy już scenę, którą chcielibyśmy zapisać. Oto jak to zrobić krok po kroku:

1. Wprowadzamy polecenie **Scena** → **Zapisz jako** otwierając okno dialogu **Zapisz scenę.**
2. Jeśli okno nie otworzy się z folderem, do którego chcemy zapisać scenę, rozwijamy listę folderów według rysunku 2.7, klikamy na folderze ***Moje Dokumenty*** i kolejno dwukrotnie klikając na ikonie folderu dochodzimy do właściwego folderu. Można wykonać to szybciej, ale na obecnym etapie pozostaniemy przy takiej metodzie.
3. Do pola oznaczonego **Nazwa pliku** wprowadzamy nazwę pliku, w którym chcemy zapisać scenę. Wpisaną nazwę zatwierdzamy kliknięciem przycisku **Zapisz.**
4. Plik ze sceną zostanie zapisany i na wielkim pomarańczowym przycisku w środku panelu narzędzi (przycisk **Przenieś scenę**) pojawi się wpisana nazwa pliku lub przynajmniej jej początek.

Zanim przejdziemy dalej, dwie dodatkowe uwagi:

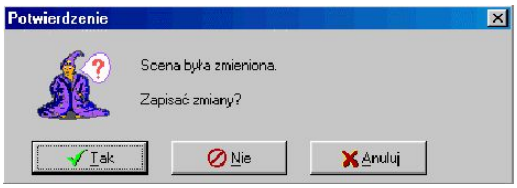
Na początku tego rozdziału powiedzieliśmy, że sceny mogą tworzyć grupy liczące do 100 członków, które są zapisane plikach z tą samą nazwą i innym rozszerzeniem. Pliki zawierające sceny mogą mieć rozszerzenia ***.s00***, ***.s01*** i tak dalej aż do ***.s99***. Gdy nie wpiszemy rozszerzenia w polu **Nazwa pliku**, scena zostanie zapisana do pliku z rozszerzeniem ***.s00***. Po wpisaniu jednego z wyżej wspominanych rozszerzeń (np. ***.s03***), Baltie zapisze scenę do pliku z takim rozszerzeniem. Jeśli wprowadzimy inne rozszerzenie, Baltie pomyśli, że to jest część nazwy pliku z kropką i doda do tej nazwy rozszerzenie z numerem, który jest aktualnie ustawiony w panelu narzędzi.

Nie należy używać kropki w nazwach plików ze scenami. Starsze wersje Baltie w niektórych sytuacjach uznawały za nazwę tylko część do pierwszej kropki, dlatego mogłoby się zdarzyć, iż Baltie wczyta inny plik, niż ten, który chcieliśmy otworzyć.

Otworzenie zapisanej sceny i przygotowanie nowej

Jeśli zadecydujemy się utworzyć scenę, którą będziemy chcieli zapisać, należy zapisać ją jak najwcześniej. Najlepiej jest przygotować plik zanim zaczniemy tworzyć scenę. Kolejność działań jest bardzo podobna jak przy zapisywaniu sceny:

1. Naciskamy przycisk **Otwórz / Nowa scena.** Otworzy się okno dialogu **Otworzyć lub stworzyć scenę.**
2. Tak, jak już wyjaśniliśmy przy zapisywaniu sceny ustalamy, by okno było otwarte się w folderze, do którego chcemy scenę zapisać.
3. Wybieramy nazwę dla pliku, który ma przechowywać scenę.
4. W polu **Nazwa pliku** wpisujemy wybraną nazwę pliku, do którego chcemy zapisać nową scenę. Należy przy tym pamiętać o właściwym wpisaniu (lub nie wpisywaniu) rozszerzenia.
5. Naciśnięciem klawisza ENTER lub przycisku **Otwórz** zatwierdzamy wprowadzone dane.
6. Jeśli akurat pracujemy nad jakąś sceną, która zmieniła się od ostatniego zapisania, Baltie najpierw ostrożnie spyta, czy chcemy scenę zapisać (patrz rysunek 2.11). Decyzję podejmujemy naciskając odpowiedni przycisku w oknie dialogu.
7. Po zapisaniu sceny pod nową, nieistniejącą nazwą Baltie wyczyści obszar roboczy i przygotuje się do tworzenia nowej sceny. Jej nazwę wyświetli w nagłówku okna aplikacji.



Rysunek 2.11
Pytanie o zapisaniu otwartej sceny

Dokładnie w ten sam sposób można otworzyć scenę, którą wcześniej zapisano do pliku. Jedyną różnicą jest to, że zamiast nazwy nie istniejącej sceny należy wprowadzić nazwę sceny, którą chcemy otworzyć. To nic trudnego – nawet nie potrzeba umieć pisać. Wystarczy znaleźć w centralnym polu okna dialogu właściwy plik i kliknąć na jego nazwie. Jego nazwa automatycznie pojawi się w polu **Nazwa pliku** i wystarczy tylko zatwierdzić nasz wybór.

Jeszcze szybszy sposób jest dwukrotnie kliknąć na ikonie pliku. Plik natychmiast zostanie otwarty.

Zapisanie nowszej wersji programów

Jak już powiedzieliśmy, od chwili, gdy zapiszemy scenę na przycisku w środku panelu narzędzi pojawi się nazwa pliku ze sceną (lub przynajmniej początek nazwy). Od tej chwili możemy łatwiej zapisywać scenę. Kiedy już poprawimy swoją scenę i będziemy chcieli zapisać nową wersję sceny, wystarczy nacisnąć tylko przycisk **Zapisz scenę**. Znajduje się na lewej krawędzi panelu narzędzi (patrz rysunek 2.12). Scena zostanie zapisana do pliku, którego nazwa jest na pomarańczowym przycisku.

Rysunek 2.12
Szybkie zapisanie nowej wersji sceny

W pliku pomocy możemy przeczytać, że do zapisywania sceny służy przycisk **Zapisz scenę.** Przy używaniu tego przycisku należy jednak zachować ostrożność. **Przycisku Zapisz scenę** używamy **tylko do zapisywania wcześniej zapisanej lub aktualnie otwartej sceny**. To, że możemy go użyć poznajemy po tym, że na przycisku **Przenieś scenę** (wielki pomarańczowy przycisk w środku panelu narzędzi) jest nazwa pliku ze sceną. Jeśli naciśniemy przycisk **Zapisz scenę** w chwili, gdy na przycisku **Przenieś scenę** są trzy znaki zapytania, Baltie sam przydzieli plikowi nazwę i folder. Jeśli nie wiemy, jaką nazwę Baltie chce przydzielić plikowi i do którego folderu chce zapisać plik, trudno będzie się później odszukać ten plik.

W tym momencie może się wydawać, że praca z plikami jest bardzo skomplikowana. Jest to wrażenie wynikające z braku doświadczenia. Gdy zaczniemy programować i będziemy używać trybu *Budowanie* tylko do tworzenia scen, na których Baltie będzie realizować przygotowane przez nas zadania, okaże się, że praca z plikami Baltie jest zorganizowana tak, by jak najwygodniej przygotowywać programy.

2.4 Czego nauczyliśmy się w tym rozdziale

Teraz już wiemy, jak zbudować scenę z przedmiotów przechowywanych w bankach Baltiego. Stworzone sceny umiemy zapisać do plików i odwrotnie, zapisane sceny z pliku otworzyć. Wiemy, co i jak zrobić, by Baltie zapisał plik tak, jak chcemy. Potrafimy nawet zakładać w swoim komputerze nowe foldery tak, by pliki Baltie nie mieszały się z innymi plikami.

3. Tryb Czarowanie

Czego nauczymy się w tym rozdziale

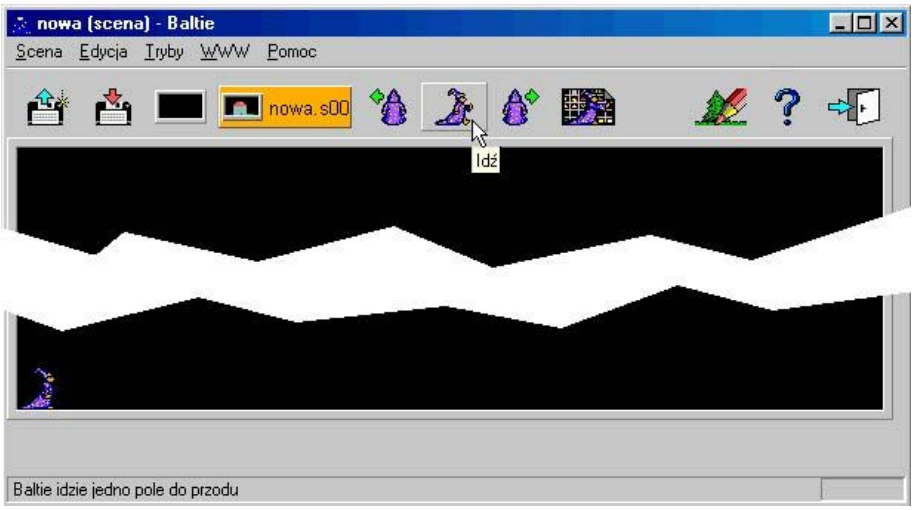
W tym rozdziale zapoznamy się z figurką Baltiego. Nauczymy się wydawać polecenia Baltiemu. Zobaczymy, jak Baltie porusza się na scenie i jak potrafi czarować.

Czarowanie to drugi poziom pracy z Baltiem. Na tym poziomie też będziemy tworzyć scenę, lecz już nie wolno przesuwać przedmiotów ręcznie, bo są one zaczarowane. Na szczęście na pomoc przychodzi nam mały czarodziej Baltie, nasz przyjaciel i pomocnik. Będzie nas prowadzić przez wyższe poziomy programu. Od tego poziomu wszystko za nas będzie robić Baltie. Nam pozostanie wydawanie mu odpowiednich poleceń.

3.1 Baltie na scenie

Na tryb *Czarowanie* przełączymy się poleceniem **Tryby** → **Czarowanie.** Po przejściu do tego trybu wygląd okna aplikacji trochę się zmieni (patrz rysunek 3.1). Trochę zmniejszy się przycisk **Przenieś scenę** i w części centralnej panelu narzędzi dwa przyciski ze strzałkami zostaną zastąpione trzema przyciskami z figurką czarodzieja Baltiego. Ponadto na scenie w lewym dolnym rogu po raz pierwszy pojawia się Baltie – mała figurka, która czeka na nasze polecenia.

Nowe przyciski służą do wprowadzania poleceń dla Baltiego. Jeśli zapomnimy, co znaczą, wystarczy wskazać na przycisk polecenia myszą (bez klikania) i przeczytać tekst, który pojawi się w dymku przy wskaźniku myszy. Bardziej szczegółowy opis widoczny będzie w pasku stanu (patrz rysunek 3.1).



Rysunek 3.1
Okno Baltiego w trybie Czarowanie (aby rysunek nie zajmował za dużo miejsca wycięto środek)

Spróbujmy kliknąć na poszczególne przyciski poleceń i sprawdzić, co Baltie zrobi. Okaże się, że potrafi iść do przodu, obrócić się na lewo i na prawo. Baltie idzie zawsze w stronę, w którą jest obrócony. Zauważmy, że jeżeli Baltie dotarł do krawędzi sceny a my chcemy, by szedł do przodu, polecenie nie zostanie wykonane.

3.2 Baltie czaruje

Baltie też potrafi wyczarować na najbliższym polu **przed sobą** przedmiot, który wybierzemy w banku przedmiotów. Przed nim oczywiście powinno być pole, nie krawędź sceny – w tym przypadku nic nie wyczaruje. Przedmiot do wyczarowania wybieramy tak samo, jak robiliśmy to w trybie *Budowanie.* W przypadku, gdy chcemy kilka razy wyczarować ten sam przedmiot, Baltie oferuje małe ułatwienie: po wyczarowaniu pierwszego przedmiotu pojawi się wpanelu

Rysunek 3.2
Zbudowany domek

narzędzi po prawej stronie przycisku **Wyczaruj z wyborem** przycisk **Wyczaruj** z ostatnio wyczarowanym przedmiotem (patrz rysunek 3.3 – ostatnio wyczarowanym przedmiotem był dzwonek). Jeśli chcemy ponownie wyczarować ten sam przedmiot (nawet w innym miejscu sceny), nie musimy znowu otwierać banków przedmiotów. Wystarczy nacisnąć ten przycisk i Baltie wyczaruje ponownie przed sobą przedmiot.

Rysunek 3.3
Przycisk Wyczaruj

Wypróbujmy działanie w trybie *Czarowanie* i zabawmy się np. w mistrza murarskiego, który powinien zbudować mały domek. Nie będziemy tego robić samodzielnie – jesteśmy przecież „mistrzem murarskim" i do pracy mamy swoich ludzi, a w rzeczywistości jednego małego czarodzieja. Wystarczy tylko kolejno podawać mu polecenia.

3.3 Zapisanie i otworzenie sceny

Tryb *Czarowanie* nie służy do tworzenia skomplikowanych scen. Jest po to, by można było sprawdzić, jakie polecenia wydawać Baltiemu, by zrobił to, co chcemy. Jednak także w tym trybie możemy zapisać scenę do pliku i z powrotem otworzyć ją z pliku. Postępujemy przy tym dokładnie tak samo, jak w trybie *Budowanie.*

☺ Między pierwszym i drugim trybem możemy się dowolnie przełączać oraz zapisywać i odtwarzać utworzone sceny w każdym trybie.

3.4 Czego nauczyliśmy się w tym rozdziale

☑ Poznaliśmy z Baltiego – czarodzieja. Nauczyliśmy się wydawać mu polecenia. Dowiedzieliśmy się, że podczas poruszania się na scenie Baltie zawsze idzie w tym kierunku, w który jest obrócony. Podczas czarowania Baltie umieści przedmiot zawsze przed sobą. Jeżeli przed Baltiem jest już krawędź sceny, nie może wykonać poleceń **Idź** ani **Czaruj**.

4. Edytor graficzny Paint

Czego nauczymy się w tym rozdziale

W tym rozdziale wyjaśnimy, jak można narysować własne przedmioty i zapisać je do banków przedmiotów. Dowiemy się paru nowości o bankach Baltie oraz o tym, jak Baltie posługuje się bankami.

Paint jest edytorem graficznym, czyli programem dla tworzenia i edycji obrazów. W porównaniu do zwyczajnych edytorów graficznych jest rozszerzony o narzędzia ułatwiające rysowanie i edycję przedmiotów Baltie, zapisanych w bankach przedmiotów. Do rysowania i edycji obrazów *Paint* oferuje mnóstwo narzędzi. Ich opis szczegółowy znajdziemy w pliku pomocy w temacie *Narzędzia edytora Paint.*

4.1 Banki przedmiotów

Zanim rozpoczniemy samodzielne tworzenie obrazów, powinniśmy najpierw powiedzieć jeszcze kilka słów o bankach przedmiotów i o tym, jak są zorganizowane.

Baltie posiada dwa rodzaje bank przedmiotów:

- **Systemowe banki przedmiotów** (możemy je też nazywać bankami Baltiego) są zapisane do plików z nazwą ***Baltie.*** Należą do nich wszystkie banki, które otrzymaliśmy z systemem (tzn. z programem) i które są umieszczone w tym samym folderze, co wszystkie pliki programu Baltie, czyli w folderze, w którym Baltie jest zainstalowany. Warto oznaczyć te pliki w *Windows* jako *tylko do odczytu*, by ich niechcąco nie skasować.
- **Banki przedmiotów użytkownika** są zapisane w plikach z nazwą, którą wprowadził użytkownik. Jeśli w trybie *Budowanie* otworzymy okno **Przedmioty,** pojawią się w nim banki systemowe i te banki użytkownika, które znajdują się **w tym samym folderze** i mają **tą samą nazwę,** jak tworzona scena. Można powiedzieć, iż należą one do wspólnej grupy.

Dalej podajemy zależność między numerami banków przedmiotów i rozszerzeniami nazw plików, w których te są banki zapisane:

– banki z numerami 0 – 99 zapisane są w plikach z rozszerzeniami ***.b00*** do ***.b99***

– banki z numerami 100 – 199 zapisane są w plikach z rozszerzeniami ***.c00*** do ***.c99***

Patrząc na te reguły można łatwo stwierdzić, że bank nr 5 zapisany jest w pliku z rozszerzeniem ***.b05*** , zaś bank z rozszerzeniem ***.c12*** ma w Baltie numer 112.

Zasady wyświetlenia banku w oknie Przedmioty

Gdy Baltie otwiera okno **Przedmioty,** ustala, które banki w nim wyświetlić. Zasady są jasne: do banków wyświetlanych włączone są wszystkie „zaprzyjaźnione" banki użytkownika, czyli banki zapisane w plikach z tą samą nazwą. Natomiast z banków systemowych Baltie wybierze tylko te, które nie mają wśród banków użytkownika odpowiednika z tym samym numerem. Krótko: najpierw banki użytkownika, a jeśli zostały niewykorzystane numery – banki systemowe.

Jeśli chcemy utworzyć np. scenę ***Pokemon*** i utworzę dla niej banki użytkownika ***Pokemon.b00*** i ***Pokemon.b01,*** w oknie **Przedmioty nie zostaną** wyświetlone banki systemowe, zapisane w plikach ***Baltie.b00*** i ***Baltie.b01,*** ponieważ najpierw zostaną uwzględnione banki użytkownika z tymi numerami (tzn. banki zapisane w plikach z tym samym rozszerzeniem).

Polecenie nadania nazwy banku

Z tego co powiedzieliśmy możemy wyprowadzić prostą metodę tworzenia nazw plików używanych do zapisywania banków:

Banki, których chcemy w przyszłości używać w rozmaitych rodzajach scen:

- zapisujemy do folderu, w którym jest zainstalowany Baltie, do plików z nazwą ***Baltie***.
- przydzielamy im numer, który nie będzie kolidował z numerami istniejących banków otrzymanych z systemem. Aby zostawić autorom Baltiego miejsce na przyszłą rozbudowę, zalecane jest używanie dla nowych banków numerów od 50 do 99.

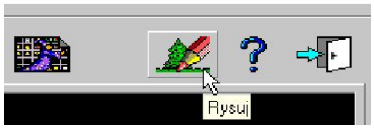
Banki, których chcemy użyć tylko w jednej grupie scen:

- zapisujemy do plików z „nazwą grupy”
- przydzielamy im numery, który nie będzie kolidowały z numerami banków systemowych. Zalecamy używanie numerów od 100 do 199.

Jeśli będziemy trzymać się tych reguł, pliki z bankami systemowymi zawsze będą miały rozszerzenie zaczynające od litery **b** (banki Baltiego). Pliki z bankami użytkownika będą miały rozszerzenie zaczynające się od litery **c** . (Pomysł na literę **c** pochodzi od angielskiego słowa *customize* – dostosować).

4.2 Uruchomienie edytora Paint

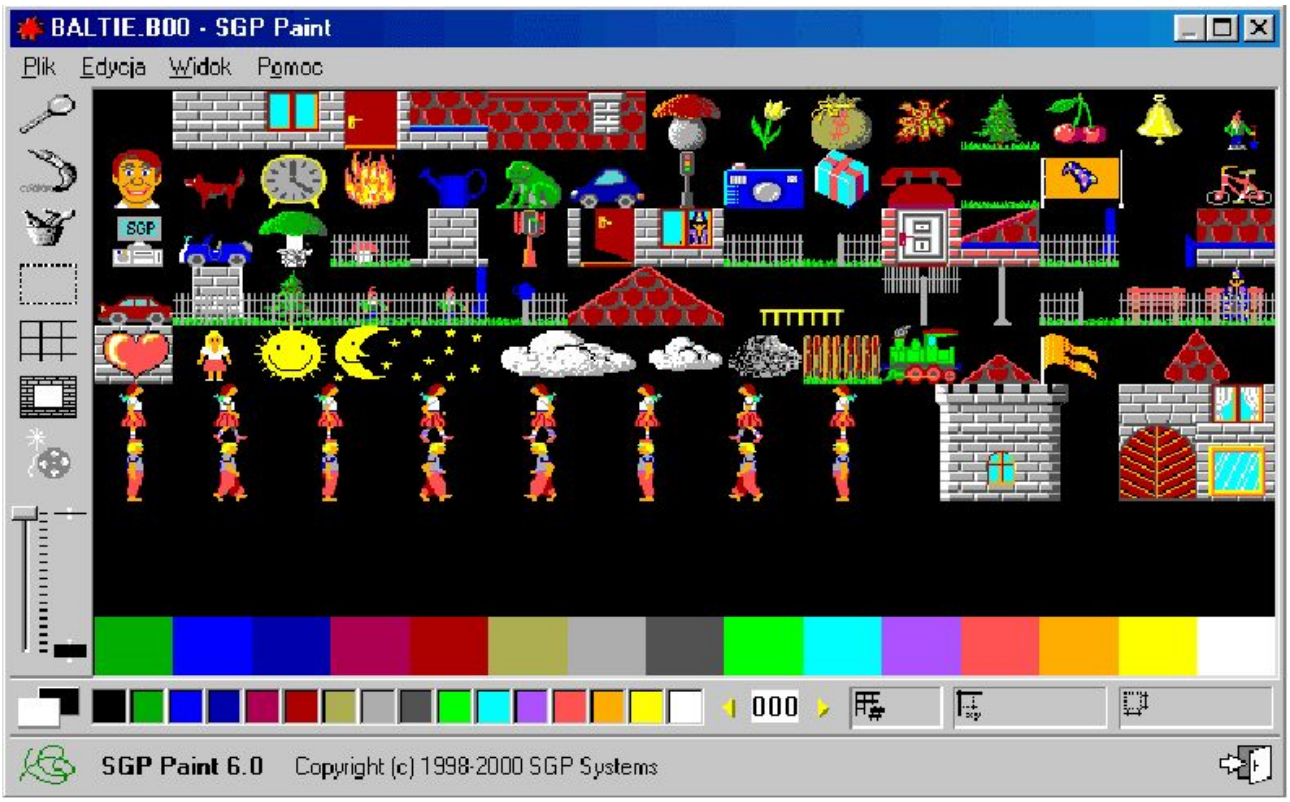
Paint można uruchomić w każdym trybie pracy naciśnięciem przycisku **Rysuj,** który znajduje się w prawej części panelu narzędzi (patrz rysunek 4.1). Można go także uruchomić z okna **Przedmioty.** W tym oknie znajduje się przycisk **Rysuj** (znów w prawej części panelu narzędzi, lecz panel znajduje się na dole okna **Przedmioty**). Trzecim sposobem uruchomienia edytora *Paint* jest przeciągnięcie przedmiotu, który chcemy edytować, na przycisk **Rysuj** i upuszczenie go.



Rysunek 4.1
Przycisk Rysuj

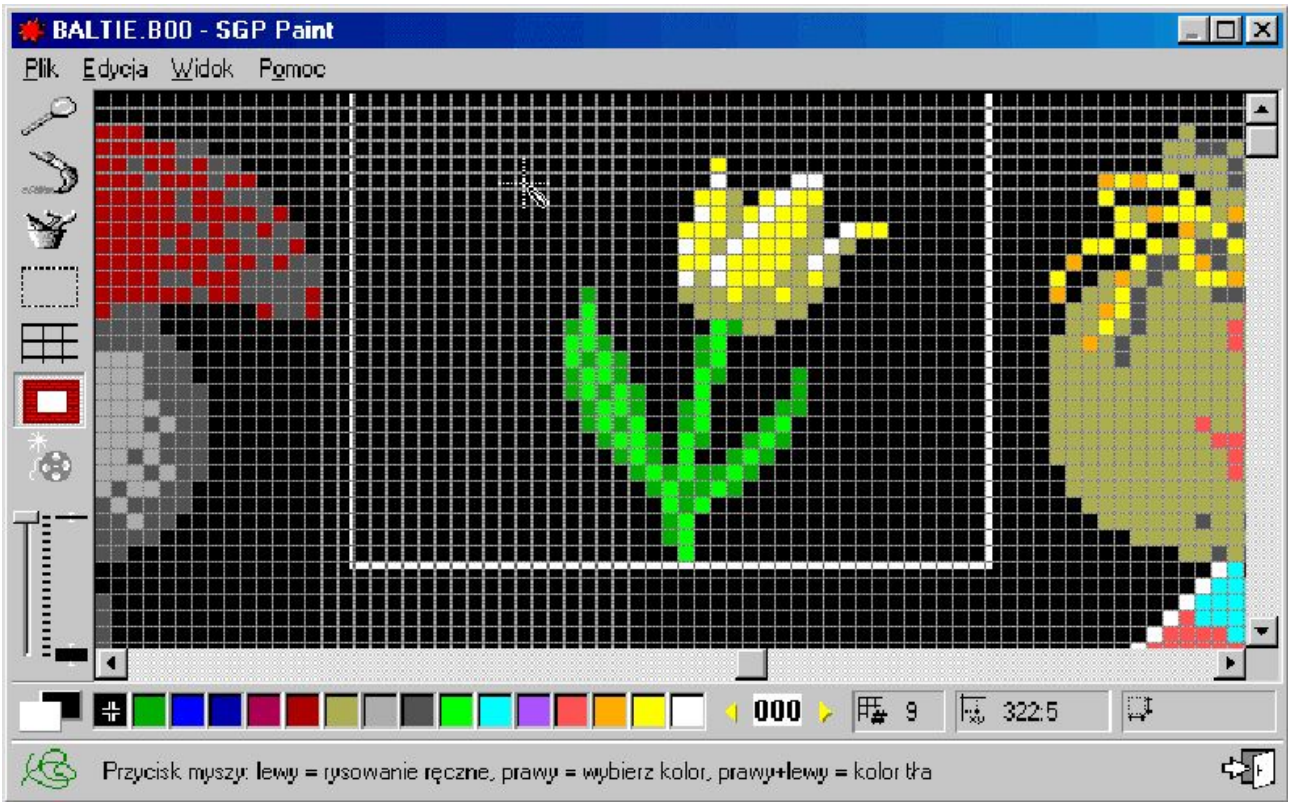
Po uruchomieniu edytor *Paint* otworzy swoje okno aplikacji. To, co zostanie wyświetlone i w którym trybie działania się uruchomi, zależy od sposobu uruchomienia edytora. I tak:

- Po uruchomieniu z okna **Przedmioty** zostanie wyświetlony bieżący bank i włączony tryb pracy z przedmiotami (patrz rysunek 4.2).



Rysunek 4.2
Okno aplikacji edytora Paint z wyświetlonym bankiem 0 i włączoną pracą z przedmiotami

- Po uruchomieniu z okna aplikacji Baltiego zarówno tryb działania, jak i wyświetlony bank będą ustawione tak, jak przy ostatnim opuszczeniu edytora. Przy pierwszym uruchomieniu zostanie wyświetlony bank nr zero.
- Po uruchomieniu edytora ciągnięciem i upuszczeniem przedmiotu na przycisk **Rysuj** wyświetlony zostanie ośmiokrotnie powiększony przedmiot, który chcemy edytować i włączony zostanie tryb rysowania (patrz rysunek 4.3).



Rysunek 4.3
Okno aplikacji edytora Paint ustawione do edycji przedmiotu nr 9

4.3 Posługiwanie się edytorem Paint

Edytor *Paint* działa w jednym z następnych dwóch trybów:

– w trybie *Rysowanie* możemy tworzyć obrazy zajmujące dowolną część sceny i możemy w nim tworzyć nowe przedmioty lub zmieniać wygląd istniejących przedmiotów,

– tryb *Praca z przedmiotami* nie jest w zasadzie przeznaczony do rysowania, ale możemy pracować z całymi przedmiotami – kopiować je w nowe miejsce w banku, przesuwać, ustawiać przezroczystość i kilka innych operacji.

Między trybami przełączamy się wybierając odpowiednie polecenie z menu graficznego **Praca z przedmiotami.** To menu można rozwinąć naciskając przycisk w panelu poleceń po lewej stronie (patrz rysunek 4.4). Z poszczególnymi poleceniami w rozwiniętym menu najłatwiej zapoznać się śledząc opisy pojawiające się w pasku stanu.

Rysunek 4.4
Menu rozwijamy naciśnięciem przycisku

Wygląd siatki, która dzieli obraz banku na poszczególne przedmioty ułatwia rozpoznanie, czy włączony jest tryb *Praca z przedmiotami,* czy tryb *Rysowanie*. Jeśli siatka wygląda tak jak na rysunku 4.2, jest włączony tryb *Praca z przedmiotami*, jeśli siatki nie widać, jest włączony tryb *Rysowanie.* Z trybu *Praca z przedmiotami* do trybu *Rysowanie* można przełączyć się jeszcze innym, prostszym sposobem: jeśli w trybie *Praca z przedmiotami* zechcemy edytować wybrany przedmiot, wystarczy kliknąć na ten przedmiot. *Paint* automatycznie przełączy się do trybu *Rysowanie*

i nastawi największe, ośmiokrotne powiększenie, aby przedmiot było łatwo edytować (patrz rysunek 4.3).

Rysowanie i inne możliwości edytora *Paint* zostały stosunkowo szczegółowo opisane w pliku pomocy. Warto zatem po uruchomieniu edytora *Paint* wykonać polecenie **Pomoc → Spis treści.**

Przesuwanie i kopiowanie przedmiotów

Jeśli z dowolnych powodów nie spodoba nam się układ przedmiotów w banku, możemy go zmienić. Jeśli chcemy przesunąć przedmiot w inne miejsce, wystarczy chwycić go, trzymać i przesunąć na nową pozycję. *Paint* przesunie na jego poprzednią pozycję przedmiot, na miejsce którego wkładamy przedmiot przesuwany (zamieni przedmioty miejscami). Przesunięcie przedmiotu jest więc w rzeczywistości zamianą miejscami dwóch przedmiotów.

Należy tu powiedzieć, że przedmiotu nie można chwycić tylko kliknięciem, lecz trzeba trzymać wciśnięty przycisk myszy podczas całego przesuwania. Jak powiedzieliśmy parę akapitów wyżej, kliknięciem na przedmiocie przełączymy się do trybu *Rysowanie.*

Czasami potrzeba narysować więcej podobnych przedmiotów. Wygodniej będzie wtedy skorzystać z możliwości kopiowania przedmiotów. Przedmioty kopiuje się za pomocą prawego przycisku myszy. Zwróćmy uwagę na to, by nie tylko kliknąć na kopiowanym przedmiocie, ale trzymać naciśnięty przycisk myszy podczas całego kopiowania. Określenie „podczas całego kopiowania" jest tu ważniejsze, niż podczas przesuwania, bo przy kopiowaniu przedmiot, który był poprzednio na miejscu, gdzie kopiujemy przedmiot, zostanie **usunięty**.

Jeśli zdarzy się pomyłkowo umieścić przedmiot w złym miejscu, wystarczy wprowadzić polecenie **Edycja → Cofnij,** które przywrócisz stan przed kopiowaniem. To polecenie trzeba wydać zaraz po błędnym działaniu, bo *Paint* potrafi cofnąć się tylko o jedną akcję.

Jeśli nie chwycimy przedmiotu prawym przyciskiem, lecz tylko na nim klikniemy, pokaże się menu lokalne (patrz rysunek 4.5), które przyda się, gdy chcemy przesuwać lub kopiować przedmioty do innego banku. Menu lokalne umożliwi nam między innymi na użycie *Schowka,* Jest on częścią *Windows* i udostępnia nam miejsce, w którym można coś na chwilę przechować.

Rysunek 4.5
Menu lokalne

Jeśli więc chcemy przedmiot przesunąć lub skopiować do innego banku, wystarczy otworzyć menu lokalne i określić, czy chcemy przedmiot z banku bieżącego do *Schowka* **Wyciąć** (zostanie po nim puste miejsce) czy **Kopiować.** Potem przesuwamy się do banku docelowego, klikamy w nim na miejsce, na które chcemy wkleić schowany przedmiot i w menu lokalnym wybieramy polecenie **Wklej**. Zalecamy raczej **Kopiować** przedmioty, a potem usuwać zbędne. Pozwoli to uniknąć „znikania" przedmiotów w razie błędu przy wykonywaniu **Wytnij**.

Jak łatwo zauważyć, menu lokalne zawiera też polecenie **Wklej przezroczyście.** Polecenie to omówimy później. Gdy poznamy zasadę „czarowania przezroczystego", łatwiej będzie zrozumieć, jak to polecenie działa i kiedy go używać.

Przedmiot pozostaje w *Schowku* nawet po wklejeniu do nowego banku. Możemy go więc wklejać wielokrotnie w różne miejsca. Zawartość schowka zmienia się dopiero wtedy, gdy włożymy do niego coś innego.

4.4 Czego nauczyliśmy się w tym rozdziale

W tym rozdziale mówiliśmy o bankach przedmiotów i o pracy w edytorze graficznym *Paint.* Poznaliśmy różnicę między bankami systemowymi i bankami użytkownika. Zapoznaliśmy się z zasadami, dzięki którym unikniemy w przyszłości przykrych niespodzianek. Nauczyliśmy się też uruchamiać edytor graficzny *Paint* i przełączać między jego trybami działania. Wyjaśniliśmy, jak wybierać, przesuwać i kopiować całe przedmioty, nawet do innych banków.

Część 2:

Zaczynamy programować

Działanie na poziomie Nowicjusz

5. Zapoznajemy się ze środowiskiem Baltiego

Czego nauczymy się w tym rozdziale

W tym rozdziale najpierw wyjaśnimy, co to takiego program. Nauczymy się tworzyć prosty program, edytować go i zapisać do przyszłego wykorzystania. Wyjaśnimy, co to jest projekt Baltiego, pokażemy, jak zapisać projekt pod nową nazwą i jakie są szczególne cechy tego zapisu. Potem nauczymy się zapisać projekt tak, by zajmował jak najmniej miejsca i by można go łatwiej przenosić na dyskietkach lub wysyłać przez Internet. Przy okazji pokażemy, jak w Baltiem można wysłać cały projekt pocztą elektroniczną. Na koniec nauczymy się otwierać nowe programy.

Uwaga!
Przed wydaniem tej książki przeczytało ją kilka osób, także dzieci. Większość z nich podczas lektury popełniała ten sam błąd: chciały jak najszybciej tworzyć złożone programy, opisane w drugiej części książki. Pobieżnie czytali wstępne części pomijając wiele informacji. W efekcie dalej napotykali tematy, których nie mogli zrozumieć, bo podczas czytania odpowiedniego rozdziału pominęli pewne fragmenty.

Nie powtarzajmy zatem cudzych błędów! Lepiej bez pośpiechu poznawać kolejne elementy i przechodzić do następnej części dopiero po nabraniu pewności, że przeczytany dotychczas tekst jest zrozumiały. Najlepiej też na bieżąco wypróbowywać nowe wiadomości. Zgódźmy się, że **im szybciej przeczytamy książkę, tym dłużej będzie trwało opanowanie poznanego materiału.**

Z pośpiechem związane jest też tworzenie programów. Samo przeczytanie tekstu nie wystarczy. Zawsze, kiedy podane będą odpowiednie przykłady (i najlepiej jeszcze częściej) warto spróbować stworzyć program, który rozwiąże postawione zadanie. Samym czytaniem gotowych programów nie można nauczyć się tyle, co samodzielnie próbując. Najlepszym sposobem nauki programowania jest tworzenie własnych programów i porównywanie ich z programami napisanymi przez innych.

5.1 Co to jest program

Zanim zaczniemy się uczyć tworzyć programy powinniśmy najpierw wyjaśnić, co to takiego*program*. Bez zbytniego teoretyzowania umówmy się, że:

Program komputerowy jest przepisem, który mówi komputerowi, jak wykonać zadanie.

W następnym tekście często będę zastępować słowo *komputer* słowem *Baltie.* Programy, które będziemy wspólnie tworzyć, będą przepisem dla czarodzieja Baltiego, jak wykonać postawione zadanie.

Program składa się z sekwencji (ciągu) poleceń, które Baltie po kolei wykonuje. Każde polecenie składa się z jednego lub kilku elementów poukładanych w ustalonej kolejności. Baltie wykonuje polecenia jedno po drugim w tym samym porządku, w jakim czytamy słowa w książce, tzn. z lewej strony do prawej i z góry na dół.

Tryb programowania w Baltie ma dwa poziomy, między którymi przełączamy się wprowadzając odpowiednie polecenia w menu **Tryby** → **Programowanie**:

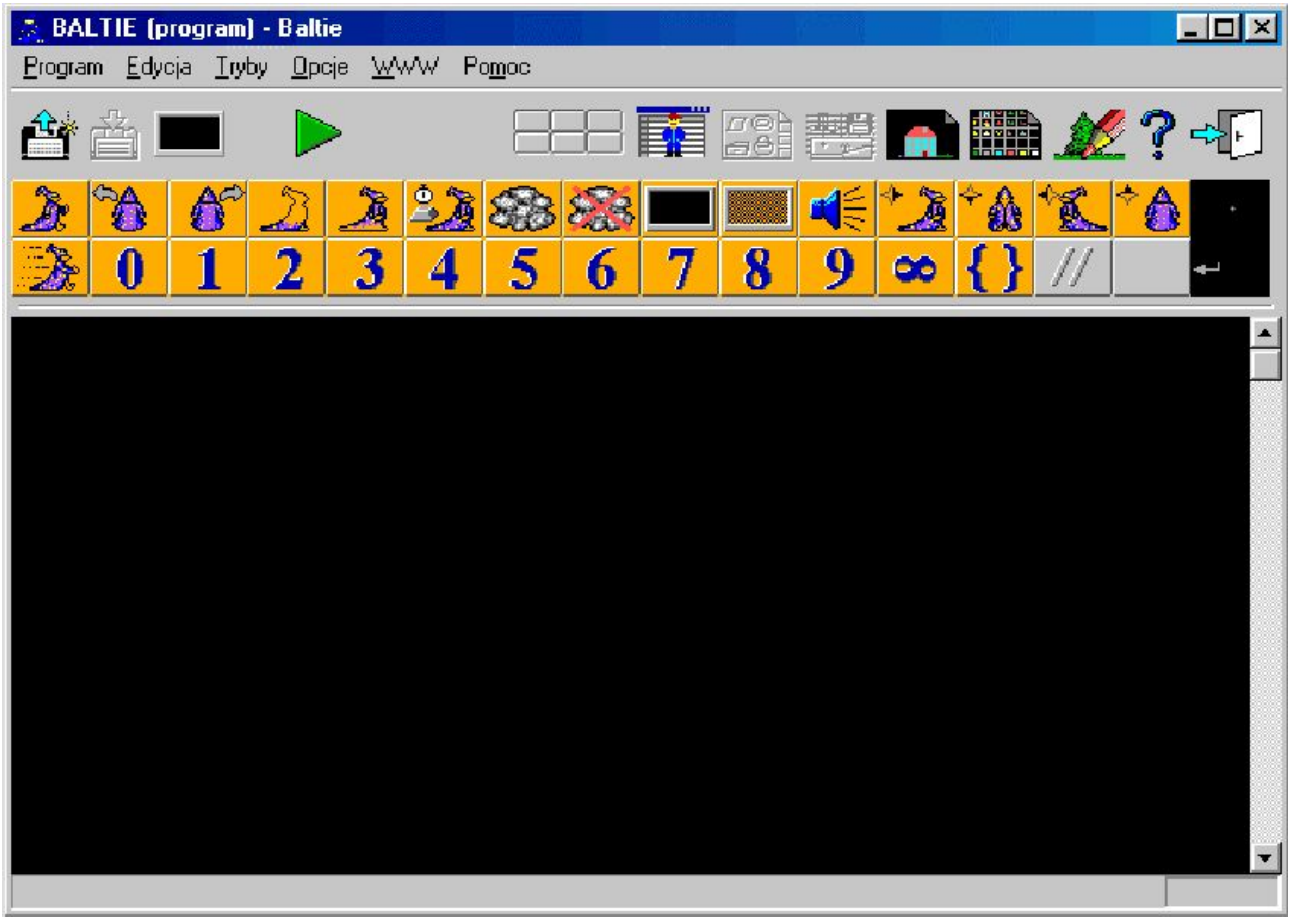
- Poziom **Nowicjusz** przeznaczony jest dla początkujących programistów nie mających żadnej wiedzy czy doświadczenia – tych nabierzemy podczas tworzenia swoich pierwszych programów. Wszystko ułożone jest tak, aby programista rozpoczynając naukę mógł pamiętać jak najmniej rzeczy.
- Poziom **Zaawansowany** oferuje wszystkie funkcje rzeczywistego języka programowania. Tutaj jednak zakładamy, że programista już zna podstawowe zasady pracy i poradzi sobie się w bardziej skomplikowanym środowisku. Nie polecamy przełączać się na ten poziom, zanim nie zostanie wystarczająco opanowane programowanie na poziomie *Nowicjusz*.

W tym rozdziale będziemy programować na poziomie *Nowicjusz*. Nauczymy się podstaw programowania, z których najważniejszą jest na pewno umiejętność podzielenia skomplikowanych problemów na kilka prostszych, mniejszych, łatwiej rozwiązywalnych. Wykorzystamy w tym celu prostsze środowisko koncentrując się na rozwiązywanym zadaniu, co pozwoli nam szybciej opanować potrzebne umiejętności.

Poziomowi *Zaawansowany* zostaną poświęcone następne rozdziały. Obydwa poziomy bazują na tych samych podstawach, co pozwoli w przyszłości wykorzystać wszystko, czego nauczymy się pracując z tą książką. Pojawią się tylko nowe funkcje i możliwości.

5.2 Tworzymy pierwszy program

Włączmy zatem Baltiego i wprowadźmy polecenie **Tryby → Programowanie → Nowicjusz.** Komputer wyświetli nowy obszar roboczy, inny od znanego nam trybu *Czarowanie*. W panelu narzędzi pod menu głównym są przyciski, z których większość już znamy z działania w poprzednich trybach. Poniżej pojawił się nowy panel narzędzi z trzydziestoma przyciskami. Będziemy go nazywać **panelem elementów**.



Rysunek 5.1
Panel elementów dla poziomu Nowicjusz

Swoje programy będziemy budować na czarnej płaszczyźnie poniżej panelu elementów. W podobnym miejscu montowaliśmy swoje sceny w trybie *Budowanie*. Tę przestrzeń będziemy nazywać **obszarem roboczym** (nie należy jej mylić z oknem roboczym *Windows*). Podczas tworzenia i edycji programów wykorzystamy umiejętności, które nabyliśmy podczas budowania sceny.

Zwróćmy jeszcze uwagę na **pasek stanu,** to jest na szary pasek w dolnej krawędzi okna aplikacji (o tym już mówiliśmy). Na prawej krawędzi paska stanu znajdziemy informację, w którym miejscu programu się znajdujemy. Jest tu podana pozycja elementu, na który wskazujemy. Ma postać zestawu dwóch liczb zapisanych jako: *wiersz:kolumna*. Najważniejsza jest jednak lewa część paska stanu. Tutaj Baltie nieustająco podaje informacje o poleceniach, na które wskazujemy myszą.

Najwyższy już czas zbudować i uruchomić swój pierwszy program. Odłożymy więc na chwilę rozważania na temat zasad budowania programu i spróbujmy rozpocząć tworzenie naszego pierwszego programu. Wrócimy przy tym do program, który pojawił się na końcu *Przewodnika* (*Tutora*).

1. Chwyćmy element **Idź (**  **)**, który jest w lewym górnym rogu panelu elementów, przenieśmy go obszar roboczy i tam go upuśćmy. Zauważmy, że zawsze sam wskoczy do lewego górnego kąta obszaru niezależnie od miejsca, w którym go upuścimy.
2. **Załóżmy**, że ten program jest zbudowany tylko z jednego polecenia. Jest więc gotowy i uruchomimy go naciśnięciem przycisku **Start (**  **)**. Otworzy się **okno wykonywania,** w którego lewym dolnym kącie będzie stać Baltie. Potem przejdzie kawałek drogi i okno wykonywania zamknie się.

Niewiele widać z działania tego programu – w bardziej skomplikowanych programach nawet nie zauważylibyśmy, czy Baltie naprawdę wykonał to, co powinien. Wydamy więc Baltiemu polecenie, by po skończeniu pracy poczekał, dopóki nie przyjrzymy się efektom jego działań.

1. Chwytamy element **Czekaj** () i wprowadzamy go do programu **za** elementem **Idź.** Znów możemy zauważyć, że niezależnie od tego, na którym miejscu pierwszego wiersza upuścimy element, automatycznie przesunie się na lewą stronę tak, by znaleźć się na końcu wiersza.
2. Uruchamiamy program. Początek będzie taki jak poprzednio, ale, gdy Baltie przejdzie kawałek drogi, okno wykonywania nie zniknie, lecz zostanie otwarte i Baltie w będzie czekać, dopóki nie naciśniemy klawisza lub nie klikniemy gdzieś w oknie myszą.

Rysunek 5.2
Idź i czekaj

Jeśli chcemy spokojnie przejrzeć wynik działania programu, musimy zawsze na końcu programu umieścić element **Czekaj** lub jakiś jego odpowiednik - polecenie, które zapobiegnie automatycznemu zamknięciu okna wykonywania programu!

Cofnięcie edycji
Każdy z nas czasami popełnia błędy. Jeśli podczas tworzenia lub edycji programu popełnimy błąd, np. wstawiając element w złe miejsce lub usuwając coś przez pomyłkę, możemy powrócić do poprzedniego stanu poleceniem **Edycja → Cofnij.** Co ważne, polecenie to trzeba wykonać **natychmiast** po popełnionej pomyłce, bo Baltie potrafi cofnąć operacje tylko o jeden krok.

Spróbujemy zbudować dłuższy program. Polecimy Baltiemu, by zrobił pięć kroków. Znamy polecenie **Idź**, wystarczy zatem spowodować, by wykonał to polecenie pięciokrotnie. Do jednego polecenia **Idź** dodamy cztery kolejne **Idź**. Pamiętajmy, że nowe elementy **Idź** potrzeba wprowadzić **przed** element **Czekaj.** Małe różnice w kierunku pionowym nie grają roli – rysunek 5.3.

Rysunek 5.3
Następny element

Rysunek 5.4
Pięć kroków do przodu

Po wprowadzeniu wszystkich czterech nowych elementów **Idź** otrzymamy program z rysunku 5.4.

Z tym programem będziemy chwilę eksperymentować, aby nauczyć się manipulowania pojedynczymi elementami i grupami elementów.

Rysunek 5.5
Potrzebna pomoc

Pomoc

W miarę jak nasze programy będą coraz bardziej rozbudowane, będziemy używać nowych elementów. Jeżeli zapomnimy, co oznacza dany element lub gdy będziemy chcieli dowiedzieć się więcej o elemencie programu, wystarczy chwycić go i upuścić na znak zapytania przy prawej krawędzie panelu narzędzi – patrz rysunek 5.5. Otworzy się wtedy okno pomocy na stronie opisującej nasz element. Czasem zamiast szczegółowych informacji wystarczy tylko mała podpowiedź widoczna w **pasku stanu**. Tutaj zawsze znajduje się krótki opis wskazywanego elementu lub przycisku.

5.3 Podstawowa edycja programu

Wiemy już, że **nowy element** wprowadzamy do programu przeciągając go z panelu elementów w odpowiednie miejsce w programie. Drugą możliwością jest pojedyncze kliknięcie na ten element. W ten sposób możemy chwycić go i już bez trzymania przycisku myszy przenieść w odpowiednie miejsce w programu. Tam klikamy ponownie – i już.

Wiemy też, że Baltie automatycznie dosuwa elementy do lewej strony za ostatni element wiersza. Jeśli więc chcemy umieścić element jako ostatni w wierszu, wystarczy upuścić go gdziekolwiek poza końcem wiersza. Jeżeli natomiast wkładamy nowy element między dwa inne elementy, trzeba to zrobić dokładnie.

Wprowadzając do programu nowy element nie musimy zawsze sięgać do panelu elementów. Jeśli ten element już znajduje się w programie, możemy go **skopiować**. Wystarczy kliknąć na nim prawym przyciskiem myszy. Chwytamy kopię, przenosimy na odpowiednie miejsce i tam następnym kliknięciem prawym przyciskiem wstawiamy.

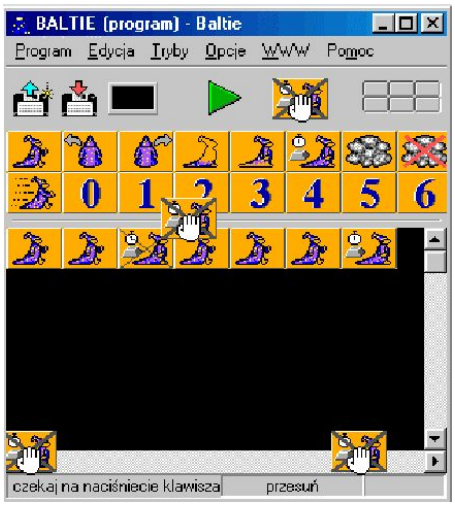
Rysunek 5.6
Kopiowanie Czekaj

Spróbujmy w ten sposób skopiować element **Czekaj** pomiędzy elementy **Idź** (patrz rysunek 5.6).

Aby sprawdzić rezultat naszej pracy, spróbujmy uruchomić ten program. Baltie przejdzie pierwsze parę kroków i będzie czekać na naciśnięcie klawisza na klawiaturze lub przycisku myszy. Potem przejdzie resztę kroków i znów poczeka na klawisz.

Jeśli zamiast klikać prawym przyciskiem myszy użyjemy lewego, element nie będzie kopiowany, lecz **przesuwany.** Baltie zaznaczy tę „drobną" różnicę przekreślając przesuwany element na jego poprzedniej pozycji i nie umieszczając znaku + na ręce, która przesuwa element.

W podobny sposób będziemy **usuwać elementy** – wystarczy przenieść element poza obszar roboczy, czyli na ramkę okna aplikacji, albo w górę na panel elementów. Gdy element znajdzie się poza obszarem roboczym, zostanie w programie przekreślony podobnie jak przy przesuwaniu, lecz grubszą kreską. W ten sposób Baltie sygnalizuje, że po upuszczeniu elementu nie zostanie on przesunięty w inne miejsce programu lecz usunięty. Kilka przykładów można zobaczyć na rysunku 5.7.



Rysunek 5.7
Różne możliwości usuwania elementu

5.4 Zapisujemy program

Zanim zabierzemy się za konstruowanie bardziej skomplikowanych programów, zobaczmy, jak zapisuje się programy. Bez tej umiejętności konieczne byłoby ponowne budowanie programu przy każdym włączeniu komputera.

Jeśli nie dysponujemy pełną wersja programu i korzystamy z wersji demonstracyjnej, można pominąć ten rozdział. Wersja demonstracyjna różni się od wersji pełnej możliwością zapisywania plików ze scenami i programami.

Jak można dowiedzieć się z pliku pomocy, do zapisywania programów służy przycisk **Zapisz program** w panelu narzędzi. Jednak nie spieszmy się z jego użyciem, bo zachowuje się nieco inaczej niż w innych aplikacjach. Twórcy Baltie tak bardzo starają się pomagać nowicjuszom, że trochę komplikują życie bardziej zaawansowanym użytkownikom komputera.

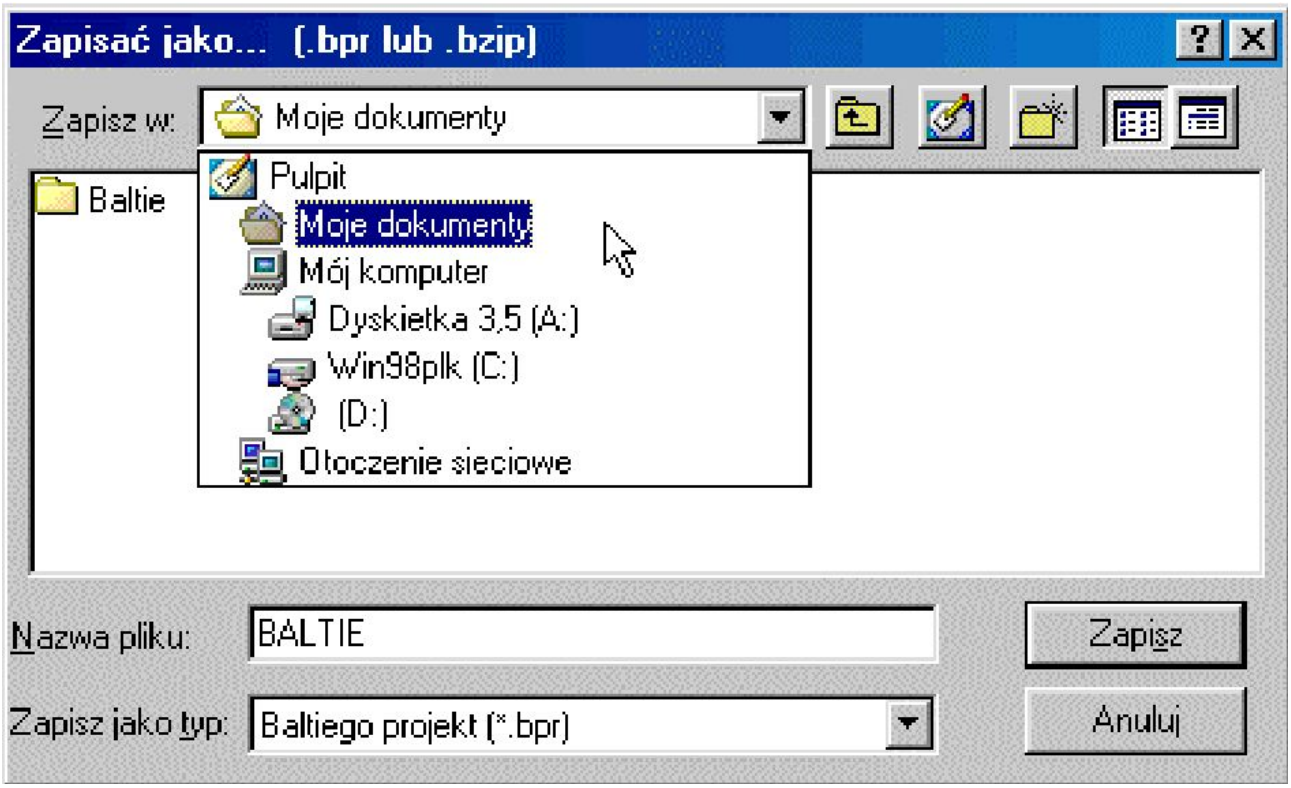
Jeśli chcemy, by komputer zapamiętał nasz program, powinniśmy zapisać go do pliku. Ten plik powinien być umieszczony w odpowiednim folderze. Baltie nie interesuje się strukturą drzewa folderów w naszym komputerze i jeśli nie wskażemy innego folderu, to zapisze nasz program do folderu, w którym został zainstalowany. Jest to jedno z najgorszych miejsc, w których można zapisywać pliki. Gorszym jest już tylko folder z plikami systemowymi *Windows*.

Pliki z własnymi dokumentami, do których należą także nasze programy, **zapisujemy zawsze do innego folderu niż folder aplikacji** (np. Baltie). Jeżeli nie będziemy tego przestrzegać możliwe jest, że kiedyś usuwając zbędne dokumenty usunie się i aplikację – lub odwrotnie, usuwając aplikację usunie się przy okazji potrzebne dokumenty.

Powiedzieliśmy, czego nie robić, teraz powiedzmy, jak należy to wykonać poprawnie. Do zapisywania własnych plików wydzielony jest w systemie *Windows* folder nazwany ***Moje Dokumenty.*** W nim właśnie najlepiej utworzyć podfolder, który nazwijmy np. ***Baltie.*** Do tego folderu będziemy zapisywać wszystkie swoje programy i pliki używane przez te programy (np. sceny, dźwięki itp.).

Rysunek 5.8
Utwórz folder

Teraz zapiszemy program. Wprowadzamy polecenie **Program → Zapisz jako.** Otworzy się okno dialogu **Zapisz jako... (.bpr lub .zip)**. Rozwijamy listę **Zapisać do** i klikamy na folder ***Moje Dokumenty*** (patrz rysunek 5.9).



Rysunek 5.9
Okno dialogu Zapisz jako...

Jeśli jeszcze nie ma utworzonego folderu ***Baltie,*** klikamy w oknie dialogu na przycisku **Utwórz nowy folder,** zakładamy folder i wchodzimy do niego. Teraz wprowadzamy do pola **Nazwa pliku** nazwę, pod którą chcemy zapisać program. Na potrzeby tej książki będziemy nazywać pliki z programami według schematu: najpierw dwucyfrowy numer rozdziału, kreska, dwucyfrowy numer programu w rozdziale, odstęp i słowne odznaczenie programu. Ponieważ to jest pierwszy program w pierwszym rozdziale, nazywać się będzie ***01-01 Pierwszy program.*** Oczywiście można też użyć innej nazwy.

Rysunek 5.10
Nazwa programu jest w nagłówku okna

Po zapisaniu programu w nagłówku okna aplikacji Baltiego pojawi się nazwa pliku, do którego program został zapisany. Od tej chwili możemy używać przycisku **Zapisz program** na panelu narzędzi, co będzie powodować wpisanie do pliku aktualnej wersji programu.

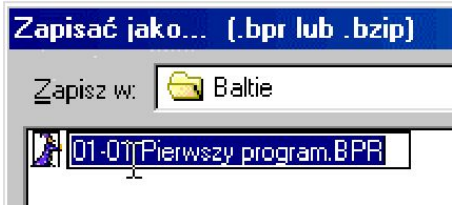
Zauważmy, że po zapisaniu programu ten przycisk nie będzie aktywny (będzie szary). Uaktywni się dopiero wtedy, kiedy coś w programie zmienimy i Baltie będzie miał coś nowego do zapisania.

WAŻNE
Baltie nie tworzy kopii zapasowych. Jeśli więc chcemy być pewni, że nie stracimy swojego program przy przypadkowym wyłączeniu prądu, warto robić kopie zapasowe samodzielnie. Można na przykład utworzyć w folderze ***Baltie*** pod-folder ***Kopie*** i do niego kopiować nowe pliki używając *Eksploratora Windows*. Cenne pliki warto przechowywać także poza komputerem – na dyskietce, płycie CD lub w Internecie.

Polecenie **Zapisać jako** przyda się też wtedy, gdy chcemy wypróbować różne zmiany w programie, ale jednocześnie zachować to, co już wymyśliliśmy. W ten sposób zawsze można będzie wrócić do poprzedniej wersji programu. Problem w tym, że przeważnie chcemy zapisać tę nową wersję do pliku z inną nazwą a obecnie używaną nazwę pliku pozostawić. Przećwiczymy działanie, które pozwoli nam zachować wyjściową programu i utworzyć nowe pliki z wersjami zmienionymi – będziemy mogli eksperymentować.

Nazwijmy program, który zapisujemy, by do niego później powrócić, **starym**, zaś ten, w którym chcemy wypróbować nasze nowe pomysły, **nowym**.

1. Naciśnięciem przycisku **Zapisz program** zapisujemy bieżącą wersję programu – od teraz będzie starym programem.
2. Wprowadzamy **Program** → **Zapisz jako** i otwieramy okno dialogowe **Zapisz jako**.
3. W polu z listą plików klikamy wolno dwa razy na nazwę pliku, w którym zapisana jest poprzednia wersja programu (stary program). UWAGA! Nie dwukrotnie kliknąć, lecz kliknąć wolno dwa razy, tzn. najpierw kliknąć, poczekać chwilę, i kliknąć ponownie. Pierwszym kliknięciem przeniesiemy nazwę pliku do pola **Nazwa pliku,** drugim przejdziemy do edycji nazwy – patrz rysunek 5.11.



Rysunek 5.11
Zmiana nazwy pliku

4. Zmieniamy nazwę starego pliku na nową. Można na przykład po numerze dodać kreskę i za nią literę **v** z numerem wersji – w naszym przypadku ***01-01-v01 Pierwszy program.*** Zmianę zatwierdzamy naciśnięciem klawisza ENTER.
5. Sprawdzamy, czy w polu **Nazwa pliku** jest odpowiednia nazwa nowego pliku, i kolejnym naciśnięciem ENTER lub naciśnięciem przycisku **Zapisz** wymuszamy zapisanie pliku.
6. System operacyjny *Windows* może nie zdążyć zauważyć, że poprzedni (stary) plik został przemianowany (patrz rysunek 5.12), stąd może być konieczne dodatkowe potwierdzenie. Otwierając ponownie okno **Zapisz jako** możemy upewnić się, że w folderze są oba pliki – wersja stara i nowa

Może już dało się zauważyć, że w oknie dialogu **Zapisać jako** w polu **Zapisać jako typ** nie jest napisane *Baltiego program,* ale *Baltiego projekt*. Jest tak dlatego, że w skład programu może wchodzić wiele innych plików, z których program korzysta (sceny, banki, tymczasowe plik itp.). Taką grupę plików nazywamy **projektem** i Baltie przy kopiowaniu i niektórych innych operacjach działa jednocześnie na wszystkich plikach projektu.

Jako **projekt** Baltie potraktuje **wszystkie pliki, których początek nazwy jest taki sam jak nazwa programu**. Mogą mieć różne rozszerzenia. Koniecznie trzeba sprawdzać, czy do projektu należą rzeczywiście wszystkie potrzebne pliki. Podczas zapisywania programu pod nową nazwą Baltie skopiuje wszystkie pliki projektu zmieniając początek nazw kopii każdego z plików na tekst podany jako nazwa nowego projektu.

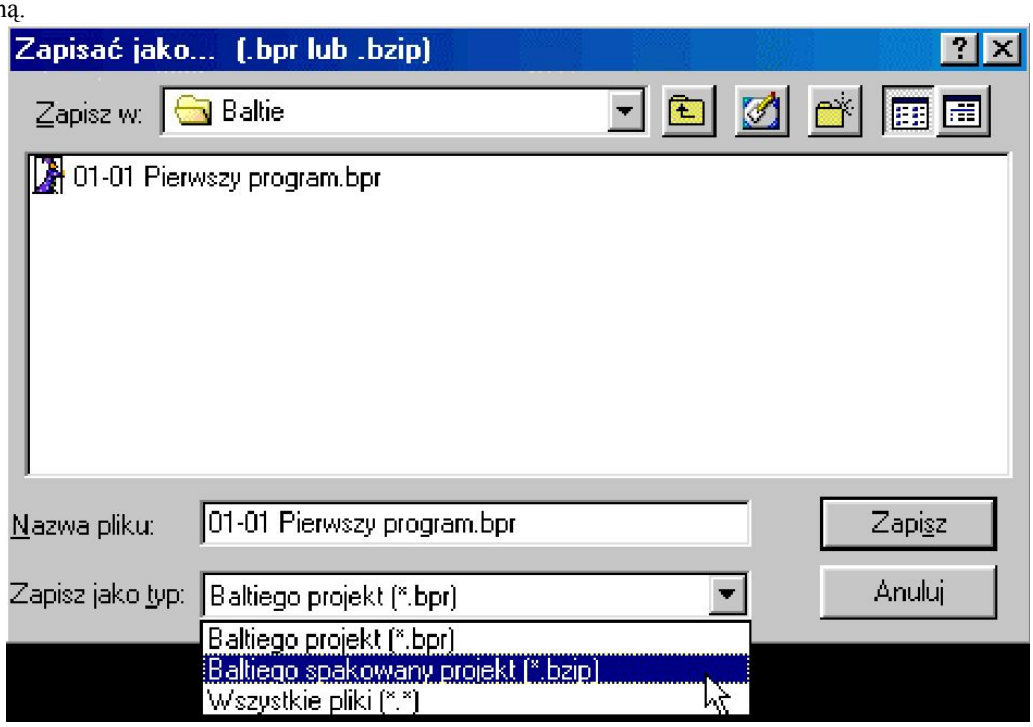
Jeśli na przykład mamy program ***Cześć,*** do projektu będą też należeć pliki ***Cześć Karol*** i ***Cześć Antek.*** Jeśli zechcemy zapisać swój program pod nową nazwą ***Czołem,*** zostaną utworzone kopie tych plików i będą nazywać się ***Czołem Karol*** i ***Czołem Antek***

Należy zatem swoje pojedyncze programy (lepiej – projekty) nazywać tak, aby różniły się już pierwszymi znakami nazwy – np. tak jak programy w tej książce. Uważać trzeba zwłaszcza podczas tworzenia nowych wersji programów.

Rysunek 5.12
Windows jeszcze nie wie, że już przemianowaliśmy plik

Zapisujemy spakowany program

Podczas tworzenia większych programów z mnóstwem scen wygodnie jest skorzystać z możliwości zapisania wszystkich plików należących do programu do jednego spakowanego (zgęszczonego) pliku z rozszerzeniem ***.zip.*** Zamiast wielu plików dostaniemy jeden, który ponadto zajmuje na dysku o wiele mniej miejsca niż oryginalne pliki. Plik taki jest rodzajem magazynu przechowującego pliki do przyszłego użycia lub przesłania. Korzystanie z plików spakowanych jest cenne przede wszystkim wtedy, gdy chcemy przenieść swoje programy na dyskietce lub wysłać pocztą elektroniczną.



Rysunek 5.13
Zapisanie spakowanego projektu

Zapisanie plików programu do pliku ***.zip*** jest proste. Wprowadzamy polecenie **Program → Zapisz jako.** W oknie dialogu **Zapisz jako… (.bpr lub .zip)** rozwijamy listę **Zapisz jako typ** i wybieramy **Baltiego spakowany projekt (.zip)** (patrz rysunek 5.13). Do pola **Nazwa pliku** wprowadzamy nazwę programu podobnie jak przy zwykłym zapisywaniu. Baltie utworzy plik, do którego powkłada (lepiej było by powiedzieć – wciśnie) wszystkie pliki, których nazwa zaczyna się tak, jak wprowadzono w polu **Nazwa pliku.**

Do tworzenia pakietów programów można wykorzystać tę samą zasadę, według której Baltie rozpoznaje pliki należące do projektu. Jeśli chcemy do pojedynczego pliku spakować razem kilka programów, możemy utworzyć nowy pusty program, którego nazwa będzie zgodna z początkiem wszystkich nazw plików, które chcemy zapisać razem. Wystarczy nawet jedna litera, np. **T**. Gdy potem zapiszemy ten program jako ZIP, do pliku spakowane zostaną wszystkie pliki z nazwą, która tak się zaczyna – w naszym przykładzie wszystkie pliki, których nazwa zaczyna się literą **T**.

Wysyłanie programu pocztą elektroniczną

Baltie zna i inne użyteczne sztuczki poza pakowaniem programu. Wyobraźmy sobie, ile czynności należałoby wykonać, by pochwalić się swoim programem koledze w innym mieście i wysłać mu program e-mailem poprzez Internet:

- musimy uruchomić program do obsługi poczty elektronicznej,
- utworzyć nową wiadomość,
- napisać adres,
- wpisać temat wiadomości**,**
- napisać wiadomość dla kolegi,
- dołączyć do niej wszystkie odpowiednie pliki, nie zapominając o żadnych elementach naszego programu,
- na koniec wysłać wiadomość.

Jak widać, jest tu trochę pracy.

Poleceniem **Program → Wyślij jako e-mail** możemy uprościć to zadanie. Baltie sprawdzi, czy w komputerze jest zainstalowany jakiś program dla wysyłania poczty elektronicznej. Jeśli znajdzie taki program, utworzy w nim nową wiadomość. Zapisze nasz projekt do pliku ***.zip***, dołączy ten plik do wiadomości. Wpisze też listę plików spakowanych do pliku ***.zip*** do tekstu wiadomości. Nam pozostanie tylko wypełnić pole adresowe, ewentualnie wpisać treść wiadomości i nacisnąć ***Wyślij*** w programie pocztowym.

Oczywiście aby korzystać z opisanej możliwości musimy mieć komputer podłączony do Internetu i konto poczty elektronicznej.

5.5 Nowy lub inny program

Być może podczas pracy nad programem okaże się, że jesteśmy zupełnie na złej drodze i trzeba cały program napisać od nowa. Wystarczy wtedy w panelu narzędzi nacisnąć przycisk **Wyczyścić program.** Naciśnięcie tego przycisku nie będzie mieć wpływu na pliki na dysku. Wyczyści tylko zawartość obszaru roboczego. Jeżeli mamy więc na dysku zapisaną poprzednią wersję programu, naciskając ten przycisk nie stracimy żadnych plików. Stanie się to wtedy, gdy przy zapisywaniu zastąpimy starą wersję programu nową – na razie pustą – wersją.

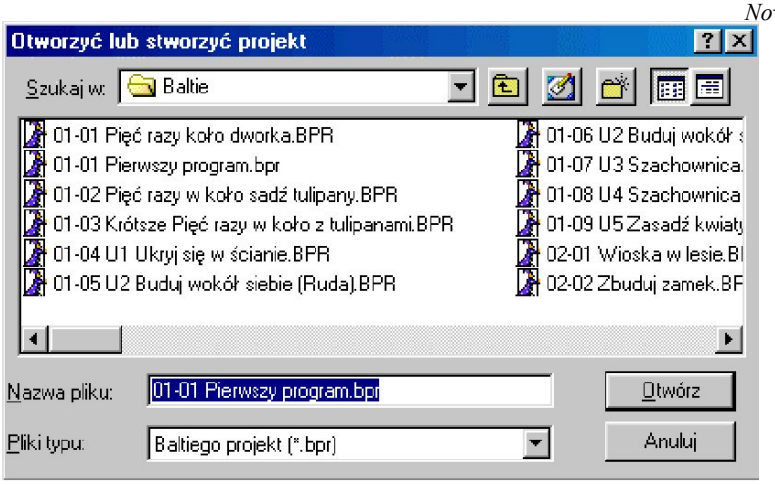
Rysunek 5.14
Wyczyszczenie programu

Żeby zakończyć pracę z programem i jednocześnie rozpocząć nowy program, naciskamy przycisk **Otwórz/Nowy program** lub wprowadzamy polecenie **Program → Otwórz/Nowy.** Otworzy się okno dialogu **Otworzyć lub stworzyć projekt,** w którym można znaleźć i otworzyć plik z nowym programem.

Program, z którym pracowaliśmy ostatnio, możemy spróbować odszukać na

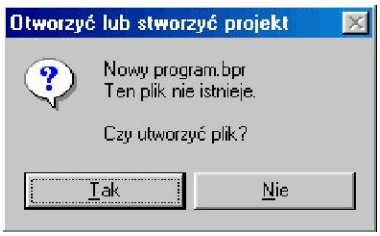
liście ostatnio otwieranych programów, która znajduje się na dole w menu **Program.** Jego nazwę znajdziemy pomiędzy pokazanymi programami. Wystarczy na nim kliknąć i program się otworzy. Natomiast jeżeli go tu nie ma, musimy wykonać operację, o której mówiliśmy powyżej. Gdy zadecydujemy się na stworzenie zupełnie nowego programu, przechodzimy się do folderu, do którego chcemy nowy program zapisać, wprowadzamy do pola **Nazwa pliku** jego nazwę i zatwierdzamy naciśnięciem ENTER lub naciśnięciem przycisku **Otwórz.** Jeśli wprowadzimy nazwę istniejącego pliku z programem, Baltie otworzy ten plik i wczyta program.

Rysunek 5.15
Nowy program



Rysunek 5.16
Okno dialogu Otworzyć lub stworzyć projekt

W przypadku podania nowej, tzn. nieistniejącej nazwy, Baltie spyta, czy ma stworzyć nowy plik. Potwierdzamy i możemy rozpocząć konstruowanie programu.



Rysunek 5.17
Pytanie przy tworzeniu nowego pliku

Rysunek 5.18
Pytanie przed zamknięciem zmienionego programu

Niezależnie od sposobu, w jaki otworzymy nowy plik z programem, Baltie zawsze najpierw sprawdzi czy zapisana została ostatnia wersję programu, z którym teraz pracujemy. Jeśli stwierdzi, że nie zapisano programu, spyta, czy zapisać program i zachowuje się zgodnie z naszą odpowiedzią.

Można skorzystać z tej cechy Baltiego, gdy niechcąco zepsujemy program. Jeśli każemy Baltiemu ponownie otworzyć otwarty plik i odpowiemy, że nie chcemy zapisywać zmian, Baltie otworzy ponownie program z pliku „gubiąc" wszystkie zmiany.

5.6 Czego nauczyliśmy się w tym rozdziale

Teraz już wiemy, co to jest program i potrafimy zbudować i poprawiać krótki program. Potrafimy program zapisać i później otworzyć go z pliku. Potrafimy nawet przy zapisywaniu zmienić nazwę poprzedniej wersji programu i zapisać nową wersję do pliku z poprzednią nazwą. Wiemy też, jaka jest różnica między programem i projektem. Potrafimy zapisać projekt z nową nazwą i wiemy, jakie niespodzianki mogą na nas przy tym spotkać.

Nauczyliśmy się też zapisywać cały projekt do jednego pliku ***.zip***. Wiemy, że można w ten sposób spakować cały projekt tak, by zajmował jak najmniej miejsca i było można go łatwiej przenosić na dyskietkach lub przesyłać pocztą elektroniczną. Potrafimy nawet cały projekt wysłać pocztą elektroniczną pod wybrany adres.

Na koniec nauczyliśmy się otwierać wcześniej zapisany program oraz jak utworzyć plik dla zupełnie nowego programu.

6. Zaczynamy naprawdę programować

W tym rozdziale wyjaśnimy, jak można skrócić wprowadzenie polecenia, które powtarza się kilka razy. Pokażemy, jak można zgrupować polecenia w blok, z którym będziemy pracować tak jak z jednym poleceniem. Opiszemy też, jak uczynić program czytelniejszym używając komentarzy. Powiemy, co to jest pętla i jak można używać jej w programach. Na koniec rozwiążemy kilka zadań.

6.1 Biegamy dookoła dworku

Wyobraźmy sobie, że chcemy, by Baltie zrobił coś bardziej skomplikowanego, np. by obiegł cały dostępny teren – nazwijmy go dworkiem. Musiałby więc dojść do przeciwnej ściany, tam zrobić w lewo zwrot, znów dojść do przeciwnej ściany i tak dalej aż obejdzie cały dworek.

Ponieważ dworek ma szerokość 15 pól, Baltie powinien najpierw zrobić 14 kroków, by dostać się do przeciwnej ściany. Program z 14 poleceniami ***Idź*** przestaje być czytelny (patrz rysunek 6.1), bo trudno policzyć tak wiele elementów.

Rysunek 6.1
Czternaście kroków do przodu

Powtarzanie polecenia

Baltie oferuje nam gotowe rozwiązanie: zamiast monotonnego powtarzania wstawiania elementu możemy przed ten element napisać liczbę oznaczającą, ile razy Baltie ma powtórzyć polecenie.

Naciśnijmy w panelu narzędzi przycisk **Wyczyść program,** (Baltie wyczyści obszar roboczy) i możemy rozpocząć od nowa.

Zbudujmy program przez wprowadzanie kolejno elementów **1** , **4** , **Idź** i **Czekaj** i wystartujmy go. Baltie przejdzie wzdłuż dolnej krawędzi do ściany przeciwnej.

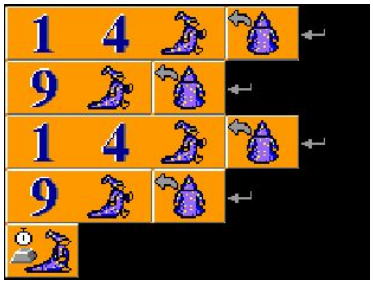
Teraz już możemy dołączyć następne elementy tak, byśmy nauczyli Baltiego, jak przejść dokoła całego dworka. Rozkażemy mu zawsze dojść do następnej ściany, gdzie zrobi **W lewo zwrot** . Program ten widać na rysunku 6.2.

Rysunek 6.2
Obejdź dworek

Zauważmy, jak Baltie łączy elementy tworzące jedno polecenie tak, że wyglądają one jako jeden szerszy element. Jeśli Baltie nie zachowuje się tak, trzeba włączyć odpowiednią opcję.

Dzielenie programu na wiersze

Można poprawić czytelność programu. Przede wszystkim przestaniemy wprowadzać wszystkie polecenia do jednego wiersza. Chwytamy element **Czekaj** i przesuwamy go gdzieś poniżej programu. Jak łatwo przewidzieć, Baltie przesunie go do lewej ściany i od dołu przyłączy go do poprzednich elementów. Zauważmy, że na poprzedniej pozycji elementu na końcu linijki pojawił się element **Koniec wiersza.** Został wprowadzony automatycznie – tym elementem powinny kończyć się wszystkie wiersze. Jedynym wyjątkiem jest ostatnia linijka programu.



Rysunek 6.3
Obejdź dworek – wersja ładniejsza

Jeśli wprowadzimy element **Koniec wiersza** do programu, zakończymy w tym miejscu wiersz i następne elementy zostaną przesunięte do następnego wiersza.

Spróbujmy wprowadzić element **Koniec wiersza** za każdym elementem **W lewo zwrot.** Program powinien wyglądać tak, jak na rysunku 6.3.

Rysunek 6.3 jest ostatnim rysunkiem, na którym wyświetlone są elementy **Nowy wiersz.** Od tej chwili będziemy je pomijać na rysunkach. Tak samo będziemy czasami pomijać ostatnie polecenie **Czekaj**.

Komentarze

Program jest teraz znacznie czytelniejszy, lecz można sprawić, że będzie jeszcze bardziej zrozumiały. Jak dotąd bez trudu orientujemy się w naszym programie. Tak prosty program prawie na pewno i za rok będzie łatwy do zrozumienia. Jednak przy bardziej skomplikowanych programach już nie będzie to takie oczywiste. Dlatego programiści wprowadzają do swoich programów *komentarze*, w których zapisują, co robi omawiana część programu. Komentarze są w programie tylko dla ludzi – komputer je zupełnie ignoruje podczas interpretacji programu.

Jeśli wprowadzimy do programu element **Komentarz** (przedostatni element w drugim wierszu panelu elementów), pojawi się kursor tekstowy i Baltie poczeka na wprowadzenie tekstu komentarza.

Spróbujmy wprowadzić do programu komentarze tak, by wyglądał jak program na rysunku 6.4.

Należy na bieżąco i w miarę szczegółowo komentować budowane programów! Ułatwia to późniejsze poprawianie programu.

Rysunek 6.4
Obejdź dworek – z komentarzami

Ważna zasada

W każdym programie jest przynajmniej jeden błąd

Pisząc programy trzeba być przygotowanym na to, że większość czasu programowania spędzimy na szukaniu i usuwaniu błędów. W miarę jak programy staną się bardziej złożone, zaczną pojawiać się w nich błędy. Niektóre ujawnią się natychmiast, część później. Zgodnie z prawem Murphy'ego poważne błędy ujawnią się dzień po tym, gdy zapomnimy, jak zbudowany jest program i co wykonuje dana jego część. Dlatego warto komentować swoje programy już podczas ich tworzenia. Czas spędzony przy wpisywaniu komentarzy pozwoli na późniejsze zaoszczędzenie wielokrotnie dłuższego czasu spędzanego zazwyczaj na „rozgryzanie" własnych programów. Na koniec coś dla tych, którzy lubią wiersze:

Trudno się twój program tworzy?
Użyj czasem komentarzy!
Czas w komentarz zapisany
W mig zmaleje, porównany
Z czasem, który szybko minie,
Gdy – dziś znane – z głowy zginie.

Edycja komentarzy

Jeśli wprowadzimy niepoprawny tekst komentarza, nie musimy go usuwać i pisać od nowa. Wystarczy wskazać komentarz myszą i nacisnąć **F2**. Baltie pozwoli na edycję tekstu. Podczas edycji tekstu komentarza obowiązują takie same reguły, jak przy pisaniu tekstu w polach okien dialogowych:

- jeśli zaczniemy pisać natychmiast po przejściu do edycji komentarza, poprzedni tekst zostanie usunięty i zastąpiony nowym,
- jeśli chcemy tylko zmienić tekst, najpierw trzeba nacisnąć jakiś klawisz sterujący kursorem tekstowym, najlepiej strzałkę w lewo lub w prawo. Zniknie zaznaczenie tekstu komentarza i wprowadzany znak zostanie wstawiony do istniejącego tekstu,

- jeśli tekst jest długi wygodniej jest korzystać z dodatkowych możliwości przesuwania kursora tekstowego za pomocą klawiszy:
    - klawisz HOME (END) przesuwa na początek (koniec) tekstu,
    - klawisze ze strzałkami używane przy naciśniętym klawiszem CTRL przesuwają na początek poprzedniego lub następnego słowa.

Opisane powyżej operacje klawiszowe działają w większości edytorów tekstu, nie tylko w Baltie.

6.2 Polecenia przyjaźnią się

Element i polecenie

Teraz spróbujemy nauczyć Baltiego, jak ma obejść dworek pięć razy dookoła. Oczywiście nikomu nie chce się ponownie (w dodatku cztery razy) wprowadzać wszystkich polecenia. Z pewnością można zrobić to tak, abyśmy nie musieli tyle pisać. Rzeczywiście. Aby jednak powiedzieć na ten temat więcej, powinniśmy wyjaśnić różnicę między terminami **element** i **polecenie.**

Element	„kafelek", który możemy przenieść z panelu elementów do programu. Element w programie może, lecz nie musi być poleceniem.
Polecenie	pełny rozkaz dla Baltiego. Może składać się z jednego lub kilku elementów. Baltie traktuje polecenie jako jeden obiekt.

Elementy **Idź**, **W lewo zwrot** i **Czekaj** są jednocześnie poleceniami. Liczby lub element **Komentarz** poleceniami nie są. Liczba nie połączona z innym elementem nic nie znaczy dla Baltiego. Komentarze są zawsze ignorowane.

Polecenie złożone i pętla

Jeśli przed poleceniem wprowadzimy liczbę, mówimy Baltiemu, ile razy powinien powtórzyć to polecenie. Liczba wspólnie z poleceniem tworzy w programie jedno polecenie złożone z kilku elementów. Na rysunkach od numeru 6.2 możemy zauważyć, że elementy, które wspólnie tworzą polecenie, są łączone przez Baltiego tak, że wyglądają jak jeden szerszy element. Nawiasem mówiąc jest to prosty sposób na rozpoznanie, gdzie się zaczyna i kończy polecenie.

Wróćmy jednak do naszego programu. Jak dotąd potrafimy Baltiemu rozkazać, by powtórzył jedno polecenie kilka razy. Teraz chcielibyśmy polecić mu powtórzenie grupy poleceń. Tu przydadzą się nawiasy poleceń. Za ich pomocą możemy oznaczyć grupę poleceń, do której Baltie ma się odnosić jak do jednego polecenia. Tak zaznaczoną grupę będziemy nazywać **blokiem poleceń** lub **poleceniem złożonym**. Baltie traktuje polecenie złożone tak, jak polecenie proste – po prostu wykona go.

Pamiętajmy, że **wszędzie, gdzie można (lub trzeba) użyć polecenia, można równie dobrze użyć polecenia złożonego**

Popatrzmy teraz, jak zbudować takie polecenie złożone:

1. Klikamy w panelu elementów na elemencie **Blok poleceń** ( – ostatni pomarańczowy element w drugim wierszu).
2. Przesuwamy wybrany element przed pierwszy element przyszłego bloku i kliknięciem umieszczamy tu element z nawiasem otwierającym.
3. Przesuwamy trzymany element za ostatni element przyszłego bloku i kliknięciem umieszczamy tu element z nawiasem zamykającym.

Na koniec dodajmy, że wszystko, co znajduje się między nawiasami poleceń, które ograniczają blok, to **treść bloku**.

Teraz już wiemy wszystko, czego potrzeba do zmiany naszego programu tak, by Baltie obiegł swój dworek pięć razy. Program może wyglądać na przykład tak, jak program na rysunku 6.5.

Rysunek 6.5
Pięć razy obejdź dworek

Część programu, która powtarza się, nazywamy **pętlą.** Polecenia, które są powtarzane, tworzą **treść pętli**. Liczba powtórzeń określona jest w **nagłówku pętli**.

Praca z blokami elementów

Powróćmy teraz na chwilę do narzędzi, które ułatwią nam na edytowanie programów. Jak dotąd wyjaśnialiśmy, jak manipulować pojedynczymi elementami. Teraz pokażemy, jak można manipulować całymi blokami elementów.

Zaznaczanie bloku

Aby mieć możliwość operowania blokiem elementów, powinniśmy najpierw taki blok zaznaczyć. Przepis jest następujący:

1. Naciskamy klawisz SHIFT lub wprowadzamy polecenie **Edycja → Zaznacz blok.**
2. Wskazujemy myszą na element zaczynający blok. Wskazujemy jego róg przeciwległy do kierunku, w którym będziemy zaznaczać blok. Jeśli więc będziemy zaznaczać blok na prawo na dół, wskazujemy na lewy górny róg pierwszego elementu.
3. Przeciągamy wskaźnik myszy do ostatniego elementu zaznaczenia. Gdy po drodze przesuniemy się wskaźnikiem myszy przez środek jakiegoś elementu, zostanie on zaznaczony razem ze wszystkimi elementami między nim i początkiem bloku – patrz rysunek 6.6. Zaznaczone elementy są przyciemnione.

Rysunek 6.6
Tak skończy się rozpoczęcie zaznaczania w złym narożniku pierwszego elementu.

Na rysunku 6.6 możemy też zauważyć, czym skończy rozpoczęcie zaznaczania w złym rogu pierwszego elementu. Ponieważ na początku wskaźnik nie został ustawiony w lewym górnym, lecz w prawym dolnym rogu, wskaźnik myszy nie przeszedł przez środek elementu **5** i tego brak go w zaznaczonym bloku.

4. Kiedy zaznaczone (ściemnione) są wszystkie elementy przyszłego bloku, zwalniamy przycisk myszy (i ewentualnie klawisz SHIFT).

Jeśli wcześniej wiemy, że w zaznaczanym bloku będą tylko całe wiersze, możemy zaznaczanie bloku trochę uprościć poleceniem **Edycja → Wybierz wiersze.** Wtedy nie będziemy wybierać bloku z pojedynczych elementów, lecz całymi wierszami.

Bloki można też zaznaczać w inny sposób. Skorzystamy z tego, że każdym dwukrotnym kliknięciem na elemencie powiększamy grupę zaznaczonych elementów:

– pierwszym dwukrotnym kliknięciem zaznaczymy całe polecenie, którego jest element członkiem,
– następnym dwukrotnym kliknięciem zaznaczymy cały wiersz,

– następne dwukrotne kliknięcia poszerzają grupę zaznaczonych wierszy.

Metody i inne porady dotyczące zaznaczania bloków opisane są w pomocy. Odpowiedni temat można otworzyć np. wyszukując w indeksie pomocy temat *blok.*

Jeśli podczas zaznaczania zrobimy błąd, wystarczy kliknąć gdziekolwiek na pustej czarnej przestrzeni i zaznaczenie zostanie anulowane.

Wcięcie bloku

Zanim zaczniemy ćwiczyć kopiowanie, wkładanie i usuwanie bloków, pokażemy, jak można „wcinać" bloki. Aby programy były czytelniejsze, programiści przesuwają wiersze poleceń znajdujące się wewnątrz bloku tworząc wtekście **wcięcia**. Łatwiej potem poznać, gdzie zaczyna się blok, nawet bez nawiasów poleceń.

Blok, na którym chcemy wykonać wcięcie, powinien zawierać tylko całe wiersze włącznie z elementami **Koniec wiersza.** Gdy chwycimy myszką taki blok, pojawi się wokół niego cienka ramka, którą można przesuwać w poziomie. Tak można ustawić odpowiednie wcięcie, na przykład na dwa elementy. Na rysunku 6.7 możemy zobaczyć nie tylko zaznaczony blok z ramką, lecz też program z wciętym blokiem.

Rysunek 6.7
Zaznaczenie bloku i jego wcięcie

Istnieje kilka metod wykonania wcięcia bloku. Różnią się umieszczeniem nawiasów poleceń. Należy uważać na fakt, że w przypadku powtarzania całego bloku poleceń nawias otwierający powinien być bezpośrednio za liczbą określającą liczbę powtórzeń. Polecenia w treści bloku mogą oczywiście znajdować się na następnych wierszach. Nawias zamykający blok możemy umieszczać na kilka sposobów. Dalej pokażemy kilka innych możliwości..

Teraz warto przećwiczyć kopiowanie, przesuwanie i usuwanie zaznaczonych bloków i pojedynczych elementów. Pamiętajmy jedynie, że błędną operacją można zrobić więcej szkody i że polecenia **Edycja→ Cofnij** można użyć naprawdę tylko raz natychmiast po błędnie wykonanej operacji.

Bloki zagnieżdżone

Jak już powiedzieliśmy, polecenie złożone jest przez Baltiego traktowane tak jak inne polecenia. Gdziekolwiek w programie może znajdować się polecenie, może też pojawić się polecenie złożone. To znaczy, że wewnątrz polecenia złożonego mogą znajdować się kolejne polecenia złożone. Pokażemy to na naszym programie.

Zanim jednak przejdziemy do pisania zajmiemy się pewną, być może uciążliwą cechą Baltiego, jaką może być zbyt wolne wykonywanie zadań. Szybkość wykonywania można ustawić. Używamy do tego elementu **Nastaw szybkość** ( – pierwszy od lewej w drugim wierszu), po którym wstawiamy liczbę. Ustawmy teraz szybkość 7, która jest największą szybkością, przy której Baltie na szybkich komputerach jest jeszcze widzialny. Przy większych szybkościach otrzymamy wynik szybciej, lecz nie będziemy mogli sprawdzić, czy Baltie postępuje tak, jak chcieliśmy. Aby dowiedzieć się więcej o nastawianiu szybkości, wystarczy przenieść element **Nastaw szybkość** na znak zapytania – pomoc (patrz rysunek 5.5).

Wróćmy do naszego programu. Chcemy, by Baltie chodząc wokół dworku wykonywał dodatkowe czynności – np. sadził obok siebie kwiaty. Nie może więc tylko iść, lecz powinien zawsze najpierw obrócić się na lewo, wyczarować kwiat, potem obrócić się z powrotem na prawo i dopiero potem iść do przodu (patrz rysunek 6.8).

Rysunek 6.8
Zasadź obok siebie kwiat

Oczywiście pamiętamy, że Baltie potrafi przed sobą czarować różne przedmioty. Chowa te przedmioty w tzw. **bankach przedmiotów,** które znamy z trybu *Budowanie.* Kilku takich gotowych banków otrzymujemy z programem, następne możemy tworzyć sami za pomocą programu *Paint.* Czarowanie przedmiotu programujemy klikając w panelu narzędzi na przycisku **Przedmioty**. Kolejnym kliknięciem wybieramy w banku odpowiedni przedmiot i wprowadzamy go na odpowiednie miejsce w programie. Jest to podobne do budowania.

Podczas chodzenia wokół dworku Baltie powinien zamiast prostego kroku powtarzać całą grupę poleceń. Zbudujemy więc z tej grupy polecenie złożone, które będziemy powtarzać. Cały program można zobaczyć na rysunku 6.9.

Rysunek 6.9
Obejdź dworek sadząc tulipany

Gdy uruchomimy program i ujrzymy już oczekiwany efekt, nie musimy czekać, aż program się skończy. Możemy przerwać działanie programu albo naciśnięciem klawisza ESC, albo naciśnięciem myszką przycisku z otwartymi drzwiami w prawym dolnym kącie okna, w którym program jest wykonywany.

Gdy doprowadzaliśmy program do postaci z rysunku 6.9 nietrudno było zauważyć, że umiejętność kopiowania bloków poleceń znacznie upraszcza tworzenie programu. Widać także, że dałoby się program jeszcze bardziej uprościć. Skorzystamy z tego, że Baltie wykonuje dwa razy to samo – zrobi zawsze 14 kroków poziomo, potem 9 pionowo. Ponownie zatem wykorzystajmy polecenie złożone i zmienimy program tak, by obowiązek powtarzania operacji zrzucić na komputer. Możliwe rozwiązanie przedstawiamy na rysunku 6.10.

Rysunek 6.10
Program do sadzenia tulipanów używający pętli

Zaletą takiego rozwiązania jest mniejsza liczba miejsc, w których w przyszłości konieczne mogą być zmiany. W tak prostym programie jest to w zasadzie nieistotne, lecz przy bardziej skomplikowanych programach trudno będzie zagwarantować, że wprowadzając zmiany w wielu miejscach nie popełnimy błędu. Zatem – im mniej miejsc, które trzeba zmieniać, tym mniejsza szansa wprowadzenia błędu do programu.

Przy okazji tworzenia poleceń złożonych zwróćmy uwagę na przyjemną właściwość Baltiego: nawiasy poleceń w poleceniu złożonym, które znajduje się w innym poleceniu złożonym, mają inny kolor. Ten drobiazg bardzo ułatwia orientowanie się w bardziej rozbudowanych programach.

6.3 Co jest i co nie jest ważne

Być może nasz program jest już doskonały. Zauważmy jednak, że Baltie ciągle sadzi kwiaty nie patrząc na to, czy w danym miejscu jest już kwiat. Dodatkowo jeszcze sadzi kwiaty w miejsca, przez które później przejdzie i na pewno je podepcze.

Pamiętajmy, że przy budowaniu programu nie jest aż tak ważne, jak wygląda rozwiązanie zadania – my przerabialiśmy program bo przyszło nam do głowy, jak go ulepszyć, lecz spokojnie moglibyśmy zostawić pierwszą wersję. Ważne jest, jak dokładnie program realizuje postawione zadanie. Nie można więc było uniknąć ciągłego obracania Baltiego w lewo i z powrotem, bo naszym zadaniem było zasadzić tulipany obok siebie.

Ta reguła zostałaby złamana także wtedy, gdybyśmy rozkazali Baltiemu iść o jeden wiersz wyżej i sadzić przed sobą – nie sadziłby wówczas kwiatów obok siebie, jak było to opisane w zadaniu.

Z drugiej strony mogliśmy pomyśleć, że z zadania nie wynika, jak Baltie ma zachowywać się w kącie. Mógł zasadzić kwiat przy krawędzi tak, jak my go zasadziliśmy. Mógł minąć kąt i dopiero potem zasadzić kwiat, wtedy kwiaty utworzyłyby prostokąt.

Przy rozwiązywaniu zadań warto przestrzegać następujących reguł:

- Trzeba uważnie przeczytać zadanie. Jeśli to możliwe, warto poprosić o wyjaśnienie wszystkich wątpliwości.
- Jeśli wydaje się, że autor zadania o czymś zapomniał czy coś błędnie sformułował, w miarę możliwości należy starać się to wyjaśnić. Może zadanie jest poprawne i tylko zabrakło nam umiejętności skojarzenia pewnych faktów.
- Elementy, które w zadaniu nie są szczegółowo sprecyzowane można wykonać dowolnie według własnego uznania.
- Elementy, które są w zadaniu opisane, powinny działać dokładnie jak podano w zadaniu.

Jeszcze wierszyk pomocny przy rozwiązywaniu zadań programistycznych:

Abyś dotrzeć mógł do celu
Droga skryta pośród wielu
Szybko łapać muchy można
Program raczej złóż z ostrożna

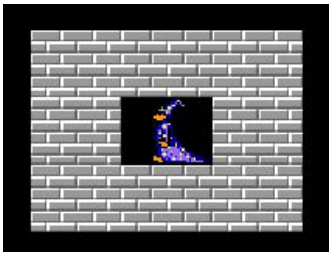
6.4 Kilka zadań do przećwiczenia

Na końcu tego rozdziału mamy kilka zadań, które warto spróbować samodzielnie rozwiązać. Zalecamy solidnie spróbować znaleźć rozwiązanie i dopiero potem porównać wyniki z rozwiązaniem przykładowym. Nie jest istotne, że znalezione rozwiązanie będzie różnić się w szczegółach od podanego. Ważne, by rozwiązać zadanie poprawnie. Z gotowego rozwiązania radzimy korzystać raczej jako z pomocy, gdy nie uda się znaleźć własnego rozwiązania.

Naprawdę warto spróbować rozwiązać zadanie samodzielnie. Tylko tak można nauczyć się budowania programów. Jeśli będziemy tylko przyglądać się cudzym rozwiązaniom, w pewnym momencie okaże się, że będzie brakować własnych doświadczeń.

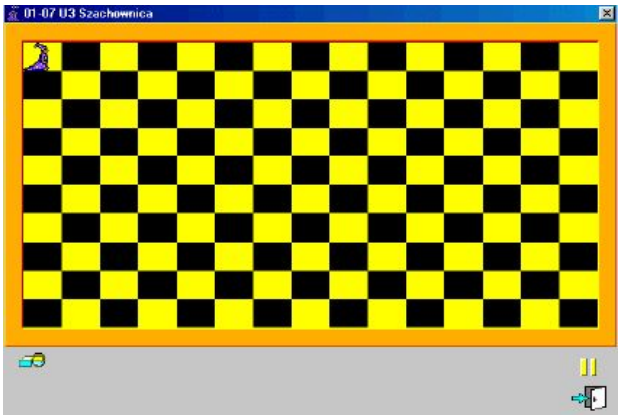
Dla tych, którzy chcą się pochwalić swoimi rozwiązaniami podajemy adres e-mail do autorów: ***rudolf@pecinovsky.cz***

Zadania



Rysunek 6.11
Zamurowany Baltie

1. Baltie potrzebuje schronienia, więc niech zbuduje sobie murową izbę i wejdzie do środka. Nie jest ważne, w którą stronę będzie obrócony w środku.
2. Zmienić poprzedni program tak, by Baltie zbudował izbę wokół pozycji, na której stoi i nie wchodził na pola, na których już postawił ścianę.
3. Pokryć cały dworek szachownicą z czarnych i żółtych pól (są w banku numer zero). Program można uprościć – zamiast czarnych pól nic nie wyczaruj. Możemy skorzystać z tego, że Baltie obrócony do ściany nie wykona polecenia **Idź** i nie zgłosi błędu.
4. Jak można zmienić poprzedni program tak, aby szachownica miała w lewym dolnym rogu żółte pole.
5. Ostatnie zadanie to kontynuacja zadania, w którym Baltie sadził tulipany podczas swojej wędrówki wokół dworku. Przypomnijmy, że Baltie sadził tulipany nawet wtedy, gdy na określonej pozycji był już kwiat. Poza tym Baltie chodząc deptał po niektórych już zasadzonych tulipanach. Zadanie jest proste. Niech znowu Baltie wędrując sadzi obok siebie tulipany, lecz niech nie sadzi ich tam, gdzie będzie chodził (aby ich nie podeptać) i niech każdy tulipan posadzi tyko raz. Potem, gdy posadzi wszystkie kwiaty, niech wędruje wokół dworka i pilnuje zasadzonych kwiatów.



Rysunek 6.12
Szachownica na całym dworku

Rozwiązania

Są już rozwiązania? Aby się czegoś nauczyć, na pewno warto spróbować je samodzielnie znaleźć i dopiero potem czytać dalej. Gotowe? Dobrze. Popatrzmy teraz na przykładowe rozwiązania. Z góry przepraszamy, że nie ma komentarzy w programach – komentarz znajdzie się wtekście.

Rysunek 6.13
Ukryj się w ścianie

1. Ten program zapewne nie sprawił problemów. Baltie obejdzie wnętrza przyszłej izby i po drodze wyczaruje ścianę. Na koniec wejdzie do środka. Porównajmy swoje rozwiązanie z programem na rysunku 6.13. Ten program napisany jest tak, by funkcjonował nawet w wyjściowej pozycji Baltiego, w której ma on pod sobą i za sobą ścianę krawędzi. Pozycja wyjściowa różni się zatem od pozycje końcowej. Ponieważ w zadaniu nie określono pozycji, rozwiązanie jest poprawne.
2. Zadanie z drugiego przykładu mówi już więcej o pozycji Baltiego. Ponieważ mamy budować wokół pozycji, na której stoi Baltie, musimy najpierw przesunąć go na środek dworku, by mógł budować wokół siebie i by nie przeszkadzała mu w tym ściana krawędzi. Ten kawałek jednak nie jest częścią programu – dlatego na rysunku 6.14 jest pokazany przed komentarzem.

Tym razem mamy pokazane dwa rozwiązania. Pierwsze polega na tym, że Baltie nie potrafi czarować na skos. Wejdzie więc najpierw w miejsce, gdzie ma być ściana i wyczaruje bloczek w rogu. Potem wróci i wyczaruje ścianę na pozycji, na której był przed chwilą. Obróci się i powtórzy operację przy następnym rogu.

Rysunek 6.14
Buduj wokół siebie tam, gdzie stoisz

Rysunek 6.15
Buduj wokół siebie wersja druga

Drugie rozwiązanie wydaje się lepsze. Baltie najpierw wymuruje uliczkę i potem do niej wstawi oba wejścia. Nie dajmy się zmylić w tym programie wcięciem drugiego wiersza – chodziło o umieszczenie podobnych poleceń koło siebie. Z powodów estetycznych wprowadzono także odstęp przed

poleceniem *Czekaj*.

3. Program pokrycia dworku szachownicą jest najtrudniejszy. Problem rozwiązany jest w ten sposób, że cały dworek zostanie pokryty pięcioma parami wierszy z polami ułożonymi na przemian.

   Na początku Baltie stoi zawsze plecami do ściany. To pole zostawi czarne i pierwsze żółte pole wyczaruje przed sobą. Potem przechodzi na właśnie wyczarowane pole i dalej na następne zostawiając puste (czarne) pole. Ponieważ dworek ma w kierunku poziomym nieparzystą liczbę pól, każdy wiersz będzie zaczynać i kończyć się tym samym polem. W pierwszym wierszu pierwszym i ostatnim będzie pole puste. Nad tym polem powinno być pole żółte. Baltie od razu go wyczaruje, przejdzie na niego i obróci się wstecz.

   Ponieważ kolejny wiersz, „wsteczny", będzie zaczynał się i kończył żółtym polem, musimy Baltiemu zorganizować podróż tak, by skończył na żółtym polu. Dlatego podczas budowania tego wiersza Baltie najpierw wkroczy na puste pole, wyczaruje następne i potem przejdzie na pole wyczarowane. Przejście wyżej na początek następnej pary wierszy jest już zupełnie proste. Co prawda po narysowaniu ostatniej, piątej pary wierszy już nie bardzo jest dokąd przejść. Tutaj skorzystamy z faktu, że Baltie stojący czołem do górnej ściany nie wykona polecenia **Idź**. Nie jest to rozwiązanie zbyt eleganckie, jednak znacznie uprości nam program.

Rysunek 6.16
Pokryj dworek szachownicą

4. Przy tworzeniu szachownicy z kolorowym polem w lewym dolnym rogu pominięto problem, jak wyczarować kolor na polu, na którym stoi Baltie. Baltie wyczaruje to pole w chwili, gdy przejdzie koło niego podczas czarowania górnego wiersza. Nie powinien wzrosnąć stopień skomplikowania program.

Rysunek 6.17
Szachownica z żółtym lewym dolnym narożnikiem

5. Podczas rozwiązywania ostatniego zadania zwróćmy uwagę na to, ile razy podczas wędrówki wzdłuż jednej krawędzi powtarzać się będzie operacja sadzenia tulipanów. Musimy jednocześnie uważać, by podczas mijania narożnika tulipan został zasadzony w kącie tylko raz oraz by Baltie nie sadził tulipanów tam, gdzie później będzie chodzić. Po zasadzeniu wszystkich tulipanów rozpoczniemy nieskończoną pętlę, w której Baltie będzie ciągle chodzić wokół dworka i pilnować, by nikt nie skrzywdził wysadzonych tulipanów.

Rysunek 6.18
Baltie sadzi tulipany wokół dworka i potem ich pilnuje

6.5 Czego nauczyliśmy się w tym rozdziale

Teraz już potrafimy uczynić swoje programy czytelniejszymi korzystając z komentarzy. Nauczyliśmy się grupować polecenia w bloki i nawet zagnieżdżać blok w bloku. Wiemy, co to jest pętla i potrafimy jej użyć w programie. Stworzyliśmy kilku prostych programów.

Jest tego stosunkowo dużo, prawda? Pora więc trochę odpocząć i dopiero potem zacząć czytać następny rozdział, w którym nauczymy się tworzyć całkiem złożone programy.

7. Tworzenie złożonych programów

Czego nauczymy się w tym rozdziale

W tym rozdziale wyjaśnimy, jak postępować przy tworzeniu złożonych programów. Nauczymy się dzielić (dekomponować) złożone problemy na serię problemów prostszych i z ich rozwiązań konstruować rozwiązanie problemu wyjściowego. Wszystko to pokażemy na dwóch projektach, które zbudujemy krok po kroku. Potem opowiemy, jak szukać błędów w programach i przede wszystkim jak je szybko usuwać. Powiemy o niektórych ulubionych sztuczkach przyśpieszających działanie programu. Na koniec nauczymy się, jak wydrukować gotowy program na drukarce lub jak zapisać go do pliku graficznego.

Pozostawmy już za sobą proste, chwilami banalne programy i zajmijmy się programowaniem rzeczy ciekawszych. Wymyślmy np. jak powinien wyglądać program, w którym Baltie zbuduje wioskę na polanie w lesie tak, jak pokazano na rysunku 7.1.

Rysunek 7.1
Wioska w lesie

7.1 Podstawowe zasady projektowania programu

Gdybyśmy już teraz mieli rozwiązać to zadanie, z pewnością powstałby długi i nieczytelny program. Początkujący często mają zwyczaj od razu po otrzymaniu zadania usiąść do komputera i rozpocząć programowanie. To jest jeden z największych błędów. Doświadczeni programiści potwierdzą, że takie działanie jest najgorszym z możliwych. Jeśli zadanie jest przynajmniej trochę skomplikowane, program będzie za chwilę miał pełno błędów. Autor spędzi więcej czasu szukając, co nie działa, zamiast wcześniej przemyśleć problem. Bardzo prawdopodobne, że wkrótce nie będzie w stanie zorientować się w swoim własnym programie.

Celowo przygotowaliśmy zadanie, którego raczej nie uda się rozwiązać w prosty sposób. Pokażemy na jego przykładzie jak postępować przy rozwiązywaniu złożonych zadań i jak sprawić, by ich rozwiązywanie nie trwało dłużej, niż trzeba.

Pierwszą i najważniejszą zasadą jest to, że powinniśmy nad każdym problemem zastanowić się. Jak w wierszu:

Słuchaj mowy mądrych głosów:
Najpierw pomyśl, w jaki sposób.
Próbuj rzecz oglądać różnie,
By nie musieć skreślać później.

Podstawową techniką rozwiązywania złożonych problemów jest **dekompozycja,** czyli podzielenie problemu na mniejsze. Skomplikowany problem podzielimy najpierw na kilka prostszych problemów i od razu wymyślimy, jak znich z powrotem poskładać rozwiązanie pierwotnego zadania.

Jeśli okaże się, że któryś z tych mniejszych problemów nadal będzie zbyt skomplikowany, znowu podzielimy go na jeszcze mniejsze kawałki. Można to działanie powtarzać aż dojdziemy do zadań na tyle prostych, że potrafimy je rozwiązać. Jest to w rzeczywistości komputerowa zastosowanie starego rzymskiego *divide et impera*, czyli *dziel i rządź.* Używa się także określenia **programowanie zstępujące** lub **dekompozycja**.

Pokażemy to na przykładzie zadania z wioską w lesie. Cały rysunek składa się z dwóch odrębnych części: z lasu i z wioski. Wygodniej będzie zatem rozkazać Baltiemu najpierw wyczarować las a potem wioskę wewnątrz niego lub odwrotnie – na tym etapie jeszcze wszystko jedno.

Las łatwo zbudować – wystarczy do programu dla obchodzenia dworku dołączyć w każdym kroku wyczarowanie drzewka. Czarowaliśmy już tulipany, więc drzewka nie sprawią nam problemu.

Wioska składa się z trzech szeregów domków. Najlepiej będzie nauczyć Baltiego zbudowania szeregu domków, kazać mu przejść do czarowania następnego szeregu. To działanie wystarczy trzy razy powtórzyć.

Szereg domków składa się z czterech identycznych domków. Nauczmy Baltiego zbudować domek i przejść do „parceli" następnego domu i cztery razy powtórzyć takie budowanie. Budować domek już umiemy, więc można rozpocząć pracę.

7.2 Do pracy

Najpierw utworzymy nowy projekt. Nazwijmy go ***02-01 Wioska w lesie***. Do pustego programu wprowadzimy komentarze, które na razie będą zastępować fragmenty przyszłego programu zbudowanego tak, jak to już opisaliśmy. Nie powinniśmy jednak zapominać o tym, że Baltie musi czasami przesunąć się z miejsca na miejsce. Szkic programu można zobaczyć na rysunku 7.2.

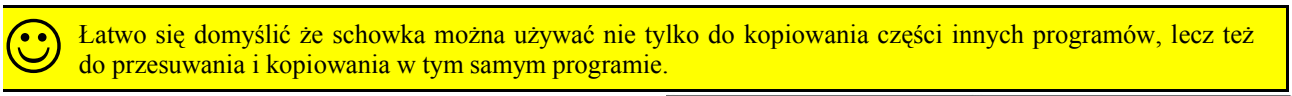
Rysunek 7.2
Szkic programu „Wioska w lesie"

Wykorzystywanie części innych programów

Zaczniemy od wyczarowania otaczającego lasu. Możemy tu użyć programu zbudowanego w poprzednim rozdziale. Ponieważ Baltie potrafi korzystać ze schowka *Windows,* więc możemy w jednym programie zaznaczyć blok, wprowadzić go do schowka, otworzyć inny program i do wkleić niego blok ze schowka. Pokażemy to na naszym przykładzie:

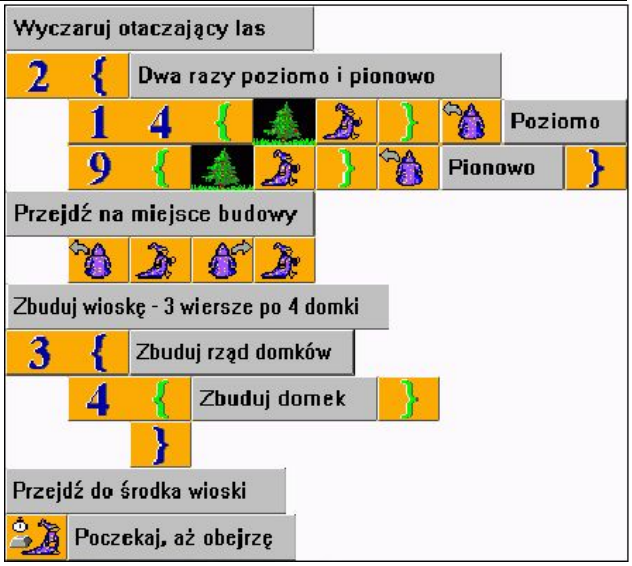
1. Zapisujemy program.
2. Otwieramy program, z którego chcemy coś skopiować, na przykład wybierając plik z listy na końcu menu**Program**.
3. Znajdujemy odpowiedni fragment i zaznaczamy go jako blok. W naszym przykładzie zaznaczymy tylko trzy środkowe wiersze, w których dwa razy powtarzamy przesunięcie poziome i pionowe.
4. Wprowadzamy polecenie **Edycja** → **Kopiuj** lub naciskamy skrót klawiszowy CTRL+C. W ten sposób umieszczamy kopię zaznaczonego bloku do schowka. Warto zapamiętać skrót klawiszowy CTRL+C – działa on we wszystkich programach *Windows*.
5. **P**onownie otwieramy program, nad którym pracujemy – w naszym przypadku wioskę. Jeśli Baltie wyświetli komunikat, że w programie, z którego kopiowaliśmy do schowka, wprowadzono zmiany, nie należy mu wierzyć i nie należy niczego zapisywać.

6. Przesuwamy się na miejsce, w którym chcemy wkleić zawartość schowka, i wprowadzamy polecenie **Edycja** → **Wklej** lub naciskamy skrót klawiszowy CTRL+V. Skrót ten działa we wszystkich programach *Windows*.
7. Wskaźnik myszy będzie wyglądać jak ręka trzymająca element. Umieszczamy element w programu tak, jak zwykły element i klikamy.
8. Jeżeli zawartość schowka chcemy wkleić jeszcze raz w innym miejscu, wystarczy powtórzyć dwa ostatnie punkty. Po wklejeniu schowek nie jest opróżniany.


Łatwo się domyślić że schowka można używać nie tylko do kopiowania części innych programów, lecz też do przesuwania i kopiowania w tym samym programie.

Zakończenie projektu

Wprowadzony blok jest za bardzo wcięty (przesunięty w prawo). Skorzystajmy z tego, że po wklejeniu elementów ze schowka blok pozostaje zaznaczony. Przesuwamy go teraz w lewo. Trzeba jeszcze zmienić „czar" rzucany przez Baltiego przy każdym kroku. Przedtem sadził obok siebie kwiat, teraz ma przed sobą zasadzić drzewko. Dodajmy jeszcze przejście przed pierwszy domek przyszłej wioski i nasz program będzie wyglądać tak, jak na rysunku 7.3.



Rysunek 7.3
Program po zakończeniu części poświęconej sadzeniu lasu

Pozostaje jeszcze zbudować wioskę. Zajmiemy się tym, jak zbudować domek. Należałoby jeszcze wyjaśnić, jak będziemy układać pojedyncze elementy, ale pora na trochę swobody. Spróbujmy samodzielnie napisać fragment programu budujący domek. Prawdopodobnie będzie wyglądać jak na rysunku 7.4.

Rysunek 7.4
Część programu do budowania domku

Uruchomimy go i sprawdzimy, jak działa. Ponieważ następny domek będzie o jeden krok dalej, Baltie powinien zbudować domki w pierwszym szeregu i następne budować na końcu szeregu aż do lasu. A może Baltie powinien po zakończeniu szeregu przejść o parę pól wyżej i zacząć tam stawiać nowy szereg domków? Jeśli Baltie pójdzie w drugą stronę będzie obracać się przy budowaniu dachu w złym kierunku.

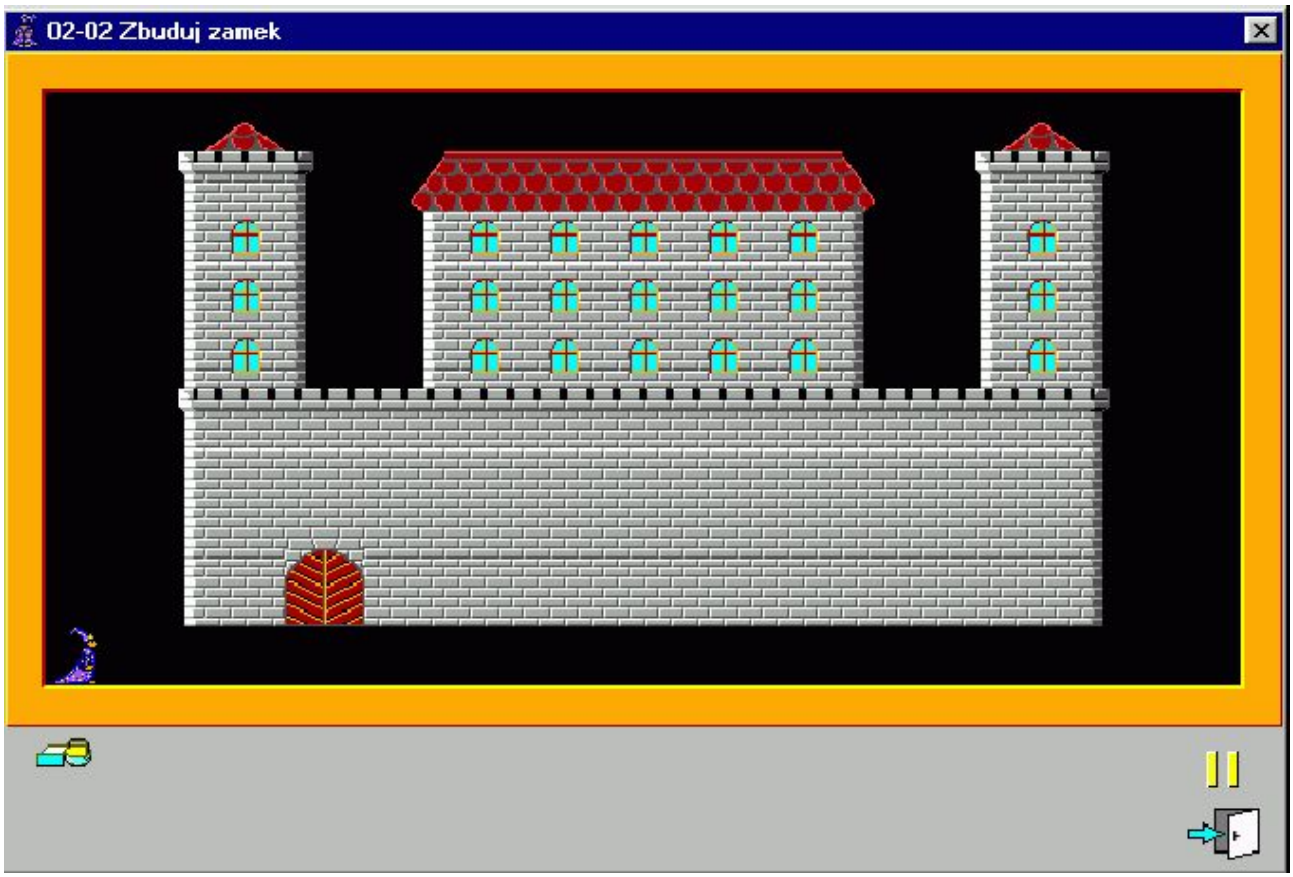
Rację będzie mieć każdy kto stwierdzi, że najprościej byłoby przejść z Baltiem z powrotem i ustawić go na początku następnego szeregu. W ten sposób sprawimy, że będzie budować drugi szereg tak, jak pierwszy. Jeśli na końcu dodamy jeszcze przejście na środek wioski, program będzie wyglądać podobnie jak ten z rysunku 7.5.

Zakończyliśmy projekt. Przypomnijmy, że zbyt wczesne zbudowanie domku prowadziło do błędnego rozwiązania. Nasze rozwiązanie okazało się proste i błędy tak widoczne, że łatwo było je usunąć. Na ogół jednak w złożonych programach nie jest to tak łatwe. Podczas tworzenia programów najlepiej jest problem rozdrabniać przechodząc od ogółu do szczegółu. Pozwala to na sprawniejsze skonstruowanie poprawnie działającego programu.

Rysunek 7.5
Końcowa wersja „Wioski w lesie"

7.3 Budujemy zamek

Spróbujemy teraz rozwiązać inne zadanie. Mamy zbudować zamek taki, jak na rysunku 7.6. Spróbujmy najpierw zastanowić się jak taki zamek zbudować. Warto przygotować samodzielnie projekt programu na kartce i potem porównać z zaproponowanym w dalszej części projektem. Zanim zaczniemy, jeszcze jedna rada: czasami wygodniej jest stworzyć najpierw przybliżone rozwiązanie problemu i dopiero potem doprowadzić je do końcowej postaci.



Rysunek 7.6
Zamek

Jeżeli mamy już gotowy projekt, przejdźmy do rozważenia przykładowego rozwiązania. Tak jak ostatnio zaczniemy od projektu programu. Na podstawowe założenia chyba się zgodzimy: Baltie po ustawieniach początkowych (fachowo nazywa się to **inicjalizacja programu**) zbuduje najpierw mur obronny, potem budynek i wieże. Na koniec posprząta i poczeka, aż obejrzymy wynik jego pracy.

Spróbujmy teraz przemyśleć poszczególne elementy projektu. Zacznijmy od muru obronnego. Składa się z trzech warstw ścian, nad którymi jest jeszcze warstwa palisady. Niestety nie można użyć pętli do ich zbudowania, bo dolne dwie warstwy zawierają bramę, więc różnią się od górnych.

Tu skorzystamy z możliwości stworzenia obiektu w dwóch etapach: najpierw zbudujemy mur a bramę wbudujemy dodatkowo. Zbudujemy mur z trzech części: najpierw zbudujemy podstawowy mur bez bramy (skomplikowałaby program), potem zbudujemy palisadę i na koniec wbudujemy pominiętą na początku bramę.

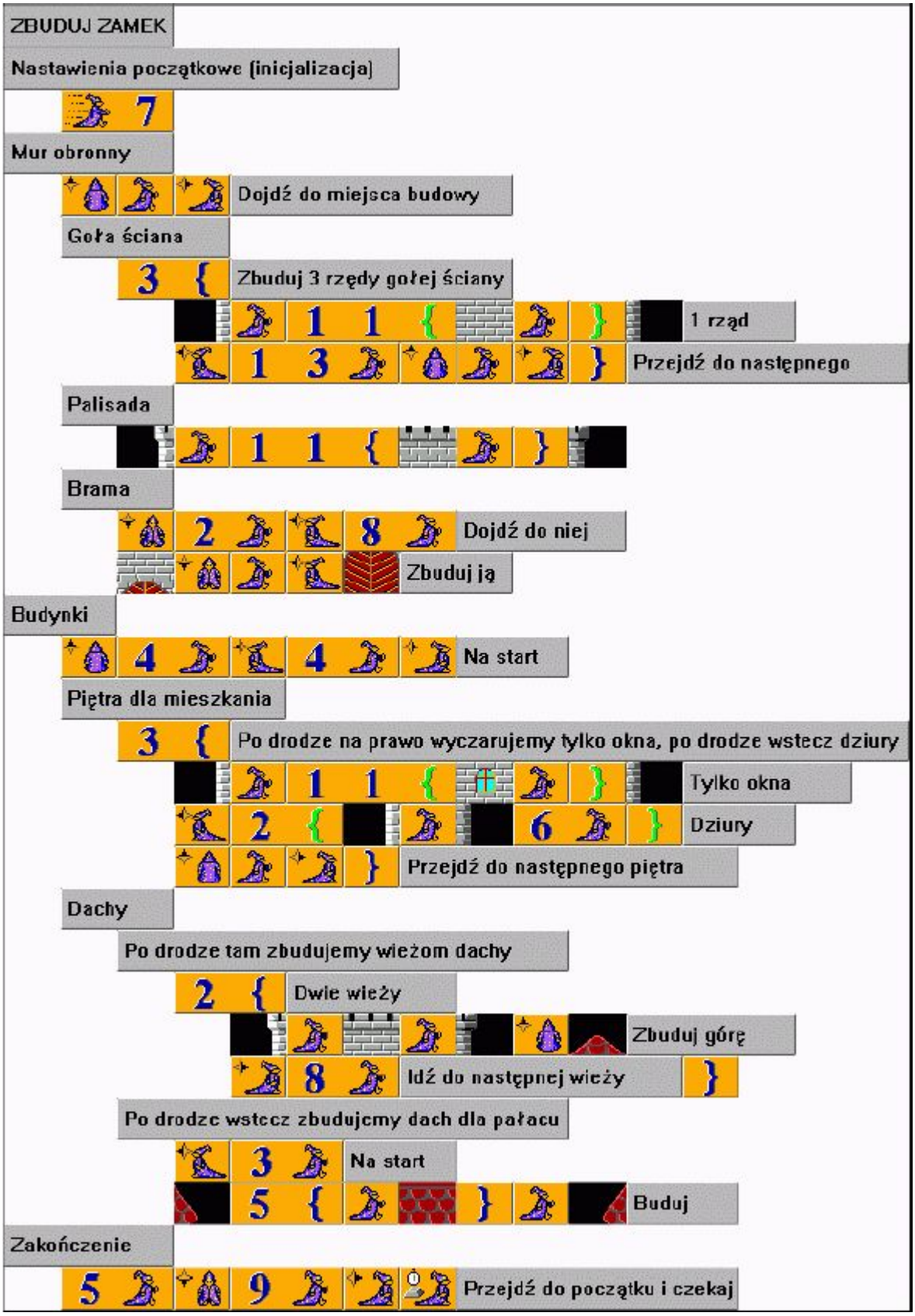
Podobny problem trzeba rozwiązać podczas rysowania budynków za murem. Gdybyśmy mieli od razu czarować piętro w pełni wykończone, musielibyśmy najpierw wyczarować krawędź wieży, okno, drugą krawędź wieży, krawędź budynku, pięć okien, drugą krawędź budynku i ponownie wieżę.

My jednak zrobimy inaczej. Idąc w prawo narysujemy jeden wielki budynek z oknami a wracając wyczarujemy dziury między wieżami i budynkiem. Możemy więc wędrując tam i z powrotem użyć pętli. Idąc „tam” będziemy powtarzać operację wyczarowania okna, wracając zrobimy dwie równe dziury. Znów wykorzystamy fakt, że Baltie ignoruje polecenia, których wykonywanie uniemożliwia mu ściana.

Podobnie można podzielić na mniejsze kroki rysowanie dachów i górnej palisady. Idąc „tam” narysujemy dwa równe daszki na wieży wykorzystując możliwość wysłania Baltiego za krawędź obszaru roboczego. Wracając zajmujemy się dachem budynku. Na koniec pozostanie nam tylko przesunąć Baltiego do lewego dolnego narożnika i obrócić go twarzą na wschód. Szkic programu zbudowanego według tego projektu widać na rysunku 7.7.

Teraz spróbujmy samodzielnie dokończyć program. Gotowe rozwiązanie można porównać z programem z rysunku 7.8.

Rysunek 7.7
Wstępny projekt programu budującego zamek



Rysunek 7.8
Program rysujący zamek

7.4 Testowanie programu

Człowiek jest istotą omylną. Każdy programista może zrobić błąd w swoim programie. Już teraz przyjmijmy, że w każdym programie jest przynajmniej jeden błąd. Im bardziej rozbudowany jest program, tym więcej błędów może w nim pozostać. Proces testowania programu prowadzący do odszukania i usunięcia błędów nazywa się **debugowaniem**.

Musimy zatem nauczyć się nie tylko budować programy ale i je testować. Jest to nieodłączny element pracy programisty. Dalej opiszemy parę przydatnych sztuczek, które ułatwią nam testowanie programu.

Przyśpieszenie gotowych części programu

Gdy uczymy Baltiego rysować czy opowiadać proste opowieści, wkrótce zacznie nam przeszkadzać, że przy testowaniu kolejnych fragmentów programu musimy zawsze czekać, aż Baltie wykona wszystko, co jest wcześniej. Budując wieś zawsze musieliśmy czekać, aż Baltie posadzi cały otaczający las. Podczas budowania zamku musieliśmy czekać na zbudowanie muru obronnego zanim możliwe było sprawdzenie poprawności części programu dotyczącej budowania wieży.

Czekania można uniknąć ustawiając na początku programu nieskończoną szybkość wykonywania. W tej szybkości wykonamy poprawną, sprawdzoną już część programu. Dla części programu, którą chcemy sprawdzić dokładniej, nastawimy odpowiednio mniejszą szybkość.

Szybkości nieskończonej można użyć tylko tam, gdzie podczas pracy programu nie zmieniamy szybkości jego działania. W takim przypadku poczekajmy na rozdział dotyczący poziomu *Zaawansowany,* w którym dowiemy się, jak powiększyć szybkość działania programu w takim przypadku.

Użycie komentarzy wierszowych

W programach często znajdują się fragmenty, których programista już nie potrzebuje, lecz z jakiegoś powodu nie chce ich wymazać. Są to np. części pomocne do testowania lub poprzednia wersja tego fragmentu programu, której nie chcemy wyrzucić, dopóki nie sprawdzimy nowej wersji rozwiązania.

W klasycznych językach programowania zamienia się takie elementy w komentarz. Podobne rozwiązanie oferuje nam Baltie. Gdy wprowadzimy w dowolne miejsce programu element **Komentarz wierszowy** (//), reszta wiersza stanie się komentarzem. Wszystko, co będzie w tym wierszu umieszczone za elementem **Komentarz wierszowy**, Baltie będzie ignorował podczas wykonywania programu. Baltie przekreśli wszystkie ignorowane elementy żeby wyraźnie zaznaczyć pomijane elementy.

Rysunek 7.9
Elementy w komentarzu zostały przekreślone

Ukrywanie części programu

Podczas testowania program i jego debugowania często zachodzi potrzeba pominięcia większej części programu podczas jego wykonywania. Przeważnie zdarza się to wtedy, gdy wykonywanie tej części przeszkadza podczas debugowania i wykonywanie danego fragmentu nie jest istotne w procesie poszukiwania błędu.

Taką możliwą do pominięcia częścią programu było np. początkowe sadzenie lasu w naszym programie budowania wioski w lesie. Możliwość ukrycia tej części byłaby tym cenniejsza, że po jej skończeniu Baltie stoi w tym samym miejscu, z którego rozpoczynał wędrówkę. Nie byłoby zbyt mądre usuwać tę część, bo przecież za chwilę będziemy jej potrzebować. Oczywiście można wprowadzić na początek każdego wiersza element **Komentarz wierszowy**, który zamaskuje resztę wiersza. Przy dłuższych częściach wymaga to jednak sporo pracy, bo musimy zająć się każdym wierszem z osobna i do tego dwukrotnie – raz podczas wprowadzania komentarza wierszowego i ponownie usuwając go.

Rysunek 7.10
Ukrywanie części programu

Istnieje jednak jeszcze jeden sposób: możemy skorzystać z bloków zagnieżdżonych i możliwości określenia liczby powtórzeń ich wykonania. Dotychczas wykonywaliśmy blok dwa, cztery lub więcej razy. Ale gdy cały obszar „niechcianego" fragmentu programu zamkniemy w bloku, który będzie powtarzany **zero** razy, to nie wykona się w ogóle. Część programu budowania wioski z ukrytą częścią początkowego sadzenia lasu pokazano na rysunku 7.10. Dla podkreślenia, że jest to rozwiązanie tymczasowe, można uzupełnić odpowiednim komentarzem wiersze z maskującymi nawiasami blokowymi.

Żeby ponownie „ożywić" wyłączoną część programu nie musimy od razu usuwać nawiasów blokowych. Wystarczy zastąpić zero – jedynką i cały blok wykona się jeden raz przy wykonywaniu programu. Gdy później będziemy chcieli ponownie wyłączyć ten fragment z użycia, zechcemy, wprowadzimy ponownie zamiast jedynki zero i Baltie będzie omijać cały blok.

Wprowadzanie punktu stopu

Ulubionym przez wielu sposobem śledzenia działania programów jest wstawianie do programów punktów stopu (po angielsku *breakpoint*). Punkty stopu wstawiamy w programie w miejscu, w którym chcemy sprawdzić stan programu.

W naszych programach, do tej pory mimo wszystko dość prostych, zawsze było jasne, gdzie w danym momencie wykonywania zadania znajduje się Baltie. W programach bardziej złożonych już nie będzie to tak oczywiste. W razie potrzeby sprawdzenia w pewnym miejscu programu, gdzie teraz znajduje się Baltie i co już zdążył zrobić, możemy wstawić do programu w to właśnie miejsce element **Czekaj**. Może zdarzyć się, że takich miejsc będzie więcej i jeśli dodatkowo niektóre z nich umieścimy wewnątrz pętli trudno będzie rozpoznać, który z wstawionych elementów**Czekaj** zatrzymał Baltiego. Możemy rozwiązać ten problem na przykład rozkazując Baltiemu piknąć kilka razy zanim zatrzyma się w danym punkcie. Możemy wtedy po liczbie piknięć poznasz, w którym punkcie stopu się zatrzymał. Nie wolno jednak przesadzać, bo trudno rozróżnić 4 piknięcia od 5, a 10 piknięć od 11 ?

Drugim problemem, który może się pojawić, jest ustawienie widzialności Baltiego. Jeżeli właśnie Baltie jest niewidzialny, bo tak ustaliliśmy w programie, trzeba przed element **Czekaj** wprowadzić element **Widzialny** a za element **Czekaj** element **Niewidzialny**. Baltie pokaże się, zatrzyma, a potem znowu zniknie. Gdy Baltiego nie widać, bo działa w nieskończonej szybkości, musimy go spowolnić – wybrać dowolną szybkość między jedynką i dziewiątką. Po skontrolowaniu stanu znowu trzeba go przyśpieszyć. Widać, że ustawianie punktów stopu w Baltie wymaga sporej uwagi, na tyle jednak ułatwia śledzenie działania programu, że warto je stosować.

Gdy punkt stopu spełni już swoje zadanie i sprawdziliśmy, że w tym miejscu programu Baltie działa poprawnie, nie trzeba od razu usuwać punktu. Wystarczy przed niego wprowadzić element **Komentarz wierszowy**. W przyszłości łatwiej będzie ponownie uruchomić ten punkt stopu. Wystarczy np. przesunąć element **Komentarz wierszowy** na koniec wiersza i punkt stopu znowu zacznie zatrzymywać Baltiego.

Na rysunku 7.11 możemy zobaczyć ustawione dwa punkty stopu. W pierwszym wierszu jest aktywny punkt, który przed zatrzymaniem Baltiego zmniejszy szybkość wykonywania programu do 8 (Baltie stanie się widzialny), by potem znów przyśpieszyć go do nieskończonej szybkości. Element **Komentarz wiersza** przygotowany jest już na końcu wiersza na ten moment, gdy już nie będzie potrzebny punktu stopu. Wystarczy przesunąć element **Komentarz wiersza** przed punkt stopu. W drugim wierszu jest nieaktywny punkt, który najpierw pokazywał Baltiego i pikał. Po naciśnięciu klawisza lub kliknięciu myszą Baltie znikał i kontynuował wykonywanie programu.

Rysunek 7.11
Punkty stopu w programie

Śledzenie

Czasami programy zachowują się tak dziwnie, że nie jesteśmy w stanie pojąć, jaka może być tego przyczyna. W takich przypadkach programiści sięgają po narzędzie ostatecznego ratunku i każą komputerowi wykonywać program jedno polecenie po drugim i szukają miejsca, które jest przyczyną błędnego zachowywania programu.

Podobną możliwość ma także Baltie. Wystarczy nastawić szybkość na 0 (zero) i Baltie przed każdym rozkazem czeka na naciśnięcie klawisza lub przycisku myszy. Jeżeli Baltie ma przy szybkości zero zrobić pięć kroków, musimy pięć razy nacisnąć klawisz, zanim zostanie wykonane całe polecenie.

Gdy program jest bardziej rozbudowany, takie śledzenie będzie bardzo uciążliwe, jeśli równocześnie nie będziemy mogli obserwować treści programu przechodzonego krok za krokiem. Jeżeli nie mamy wydrukowanego programu, (za chwilę pokażemy, jak się to robi), musimy wygląd ekranu ustawić tak, byś podczas śledzenia widoczna była podejrzana część programu i jednocześnie efekty działania Baltiego. Po uruchomieniu programu trzeba przesunąć okno z dworkiem

Baltiego trochę w bok tak, by widać było zarówno dworek, jak i program. Teraz możemy przechodzić program krok za krokiem.

7.5 Drukowanie programów

Nasze ostatnie programy były już tak rozbudowane, że nie mogliśmy zobaczyć ich na ekranie w całości. Jest to uciążliwe zwłaszcza wtedy, gdy szukamy błędów w programie.

Baltie pozwala wydrukować cały program lub tylko zaznaczony blok. Możemy przy tym wydrukować go na drukarce lub zapisać jako obraz do pliku graficznego z rozszerzeniem ***.bmp***. Plik taki można później obrabiać dowolnym edytorem graficznym, np. *Paint,* który jest częścią *Windows*. W ten sposób zostały stworzone prawie wszystkie rysunki w tym podręczniku.

Jeśli chcemy wydrukować program lub zaznaczony blok, wprowadzamy polecenie **Program → Drukuj.** Otworzy się menu, w którym ustalamy, czy chcemy drukować cały program, czy też tylko zaznaczony blok (o ile jest zaznaczony). Po kliknięciu na odpowiednie polecenie otworzy się okno dialogu **Druk.**

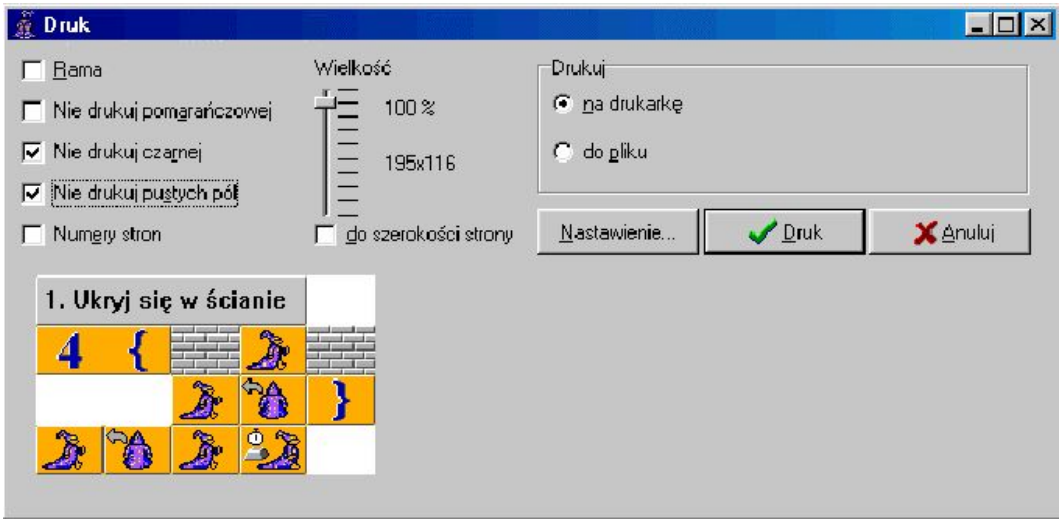
W lewej górnej części tego okna znajduje się kilka pól zaznaczania, za pomocą których możemy dostosować wygląd drukowanego programu. Ich znaczenie jest jasne i nie trzeba go opisywać. Spróbujmy włączyć/wyłączyć opcje i sprawdzić zmiany, które natychmiast pojawiają się poniżej w części z wzorem drukowanego programu.

Jedynym polem, przy którym nie widać efektu zmiany zaznaczenia, jest pole **Numery stron**. Pojawią się dopiero przy drukowaniu na drukarce. Suwak **Wielkość** także ma znaczenie głównie przy druku na papierze. Możemy nim dostosować wielkość drukowanych elementów. Optymalne ustawienie zależy od drukarki i trzeba samodzielnie poeksperymentować. Przy powiększeniu 400% rysunki na drukarce są tylko trochę mniejsze niż na ekranie. Przy drukowaniu do pliku zalecamy pozostawić domyślne ustawienie 100%.

Przełącznikiem w prawej części określamy miejsce druku. Przyciskiem **Nastawienie** można ustawić właściwości drukarki, na której chcemy drukować program.

Gdy już wszystko jest ustawione, wystarczy nacisnąć przycisk **Druk.** Jeśli drukujemy do pliku, otworzy się dodatkowe okno dialogu, w którym trzeba podać nazwę i lokalizację pliku, do którego chcemy zapisać obraz.

Niektóre drukarki postscriptowe niezbyt lubią się z Baltiem i odmawiają drukowania. Przeważnie można ustawić taką drukarkę jako do niepostscriptową.



Rysunek 7.12
Okno dialogu dla drukowania programów i ich części

7.6 Czego nauczyliśmy się w tym rozdziale

Wiemy już, jak postępować przy tworzeniu dłuższych programów. Potrafimy rozłożyć problem na kilka mniejszych problemów. Wiemy, jak z rozwiązań problemów składowych złożyć rozwiązanie pierwotnego problemu. Spróbowaliśmy stworzyć schemat programu, który stopniowo uzupełnialiśmy. Potrafimy sterować przebiegiem programu tak, by Baltie szybko wykonał sprawdzone fragmenty a wolniej działał tam, gdzie zachowuje się inaczej, niż oczekiwaliśmy. Znamy kilka sztuczek, jak chować niektóre części programu i jak je ponownie ożywić. Potrafimy też program wydrukować na drukarce lub zapisać do pliku graficznego.

8. Opowiadamy bajkę

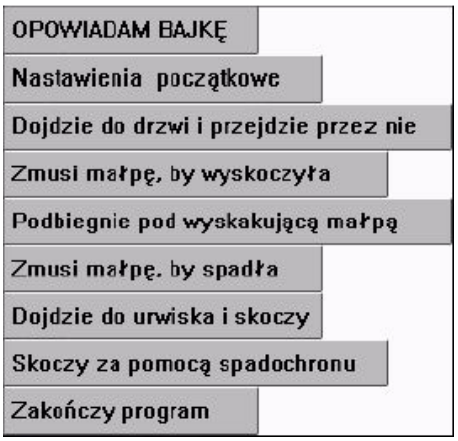
Czego nauczymy się w tym rozdziale

W tym rozdziale nauczymy Baltiego przedstawić prostą historyjkę. Dowiemy się, jak przygotować taką opowieść. Pokażemy, jak Baltie może ożywić przedmioty w swojej okolicy i przede wszystkim jak sprawić, by Baltie mógł wpływać na zachowywanie się przedmiotów w innym miejscu dworku niż to, w którym stoi. Przy okazji nauczymy się wyczarowywać przedmioty tak, by nie przesłaniały zawartości pola, na które je czarujemy.

Początkujący użytkownicy lubią za pomocą Baltiego opowiadać różne historyjki. Spróbujmy zaprogramować jedną bardzo prostą opowieść. Oto jej treść.

Baltie podejdzie do ściany, w której są drzwi. Otworzy drzwi i spotka tam małpę. Zaczaruje ją, małpa wyskoczy i Baltie podbiegnie pod nią. Ale małpa okaże się za ciężka, załamie się pod nią podłoga i małpa wpadnie na dół. Baltie pójdzie dalej aż do urwiska. Ma spadochron, więc nie boi się i skoczy z urwiska ze spadochronem. Po wylądowaniu zmieni się na młodego chłopca i odejdzie.

Najpierw za pomocą komentarzy zbudujemy podstawowy szkielet przyszłego programu. Wyznaczymy w nim podstawowe wydarzenia, które później oprogramujemy. Nie wolno nam jednak zapominać o tym, że przed rozpoczęciem opowiadania musimy przygotować środowisko, scenę, na której nasza historyjka będzie grana, i ustawić niektóre parametry. Od tego będzie się teraz zaczynać każdy nasz program.

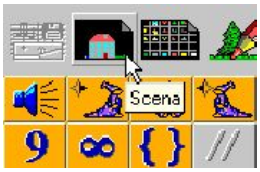


Rysunek 8.1
Szkielet programu

8.1 Przygotowujemy scenę

Przedstawianie historyjek jest typowym zadaniem, które rozwiązujemy nie tylko za pomocą programu, lecz używamy także scen i innych obiektów tworzonych poza programem.

Ponieważ często wykorzystuje się w programie sceny, w Baltie można szybko przejść do trybu *Budowanie* i z powrotem po naciśnięciu przycisku **Scena** w panelu narzędzi. Nie trzeba wykonywać serii poleceń z menu.



Rysunek 8.2
Przycisk Scena

Zanim zaczniemy rysować scenę, zwróćmy uwagę na przycisk **Przenieś scenę** w panelu narzędzi (8.3). Początkowo znajdują się w nim trzy znaki zapytania, tak jak na rysunku. Oznacza to, że scena jeszcze nie jest zapisana do pliku więc scena nie ma nazwy. Zostawmy na razie sprawę nazwy – gdy zapiszemy scenę, Baltie przydzieli jej nazwę zgodną z nazwą tworzonego przez nas programu.

Rozszerzenie nazwy (za kropką) zaczyna się od litery **s** (jak **s**cena), za którą są dwa zera. Określają numer tworzonej sceny. Baltie pozwala tworzyć sceny z numerami od 00 do 99. Numer sceny można zmienić klikając na niego i wprowadzając liczbę z klawiatury. Można też skorzystać z przycisków z żółtymi strzałkami, które są po prawej stronie i służą do zmiany numeru sceny o plus lub minus jeden.

Rysunek 8.3
Przycisk Przenieś scenę

Naciskamy przycisk **Zapisać scenę**. Baltie otworzy okno dialogu **Zapisz scenę,** w którym zaproponuje dla zapisywanego pliku nazwę zgodną z nazwą programu. Zaproponowane rozszerzenie będzie zawierać numer sceny, który został nastawiony na przycisku **Przenieś scenę.** Pozostaje nam zatwierdzić decyzję.

Teraz już na przycisku **Przenieś scenę** powinna być widoczna nazwa właśnie zapisanego pliku. Jeśli nazwa jest zbyt długa i nie zmieści się, Baltie wyświetli zamiast całej nazwy tylko 8-znakowy skrót, który używany jest w *Windows* dla systemów, które nie pozwalają długie nazwy plików (np. stary DOS).

Umiemy już budować scenę, więc przygotujmy scenę, która jest tłem naszej opowieści. Widać ją na rysunku 8.4. Małpa znajduje się w trzecim banku w piątym wierszu druga od lewej. Gotową scenę zapisujemy.

Rysunek 8.4
Scena, po której będzie chodzić Baltie

Otworzenie sceny w programie

Główną funkcją przycisku **Przenieś scenę,** o którym mówiliśmy przed chwilą, nie jest pokazywanie nazwy pliku – można ją przeczytać w całości w nagłówku okna. Ten przycisk służy przede wszystkim do wygodnego przeniesienia sceny do programu i określenia miejsca w programie, w którym scena zostanie wyświetlona. Naciśnijmy go. Baltie natychmiast przełączy się do trybu *Programowanie* i zmieni wygląd wskaźnika myszy na rękę trzymającą element. Umieszczamy ten element za komentarzem *Nastawienia początkowe* (patrz rysunek 8.5). Baltie wprowadzi element **Wyświetl scenę** z numerem **0** ma wskazane miejsce w programie. Skopiujmy potem przed niego element **Koniec wiersza** przesuwając go do nowego wiersza.

Rysunek 8.5
Umieszczenie elementu Wyświetl scenę

Jeżeli za wstawionym właśnie elementem dodamy jeszcze ustawienie szybkości działania na wartość 3 będziemy mieć już gotową część programu poświęconą ustawieniom początkowym (jak na rysunku 8.6).

Rysunek 8.6
Ustawienia początkowe

W dalszym tekście nie będziemy już pokazywać fragmentów budowanego programu. Odpowiednią część można będzie znaleźć na rysunku zawierającym cały program. Nie będziemy też specjalnie wyjaśniać znaczenia poszczególnych elementów. Jeśli nie wystarczy krótkie wyjaśnienie w pasku stanu zawsze można przeciągnąć element nad znak zapytania i przeczytać informacje podane w pomocy. Idziemy więc dalej.

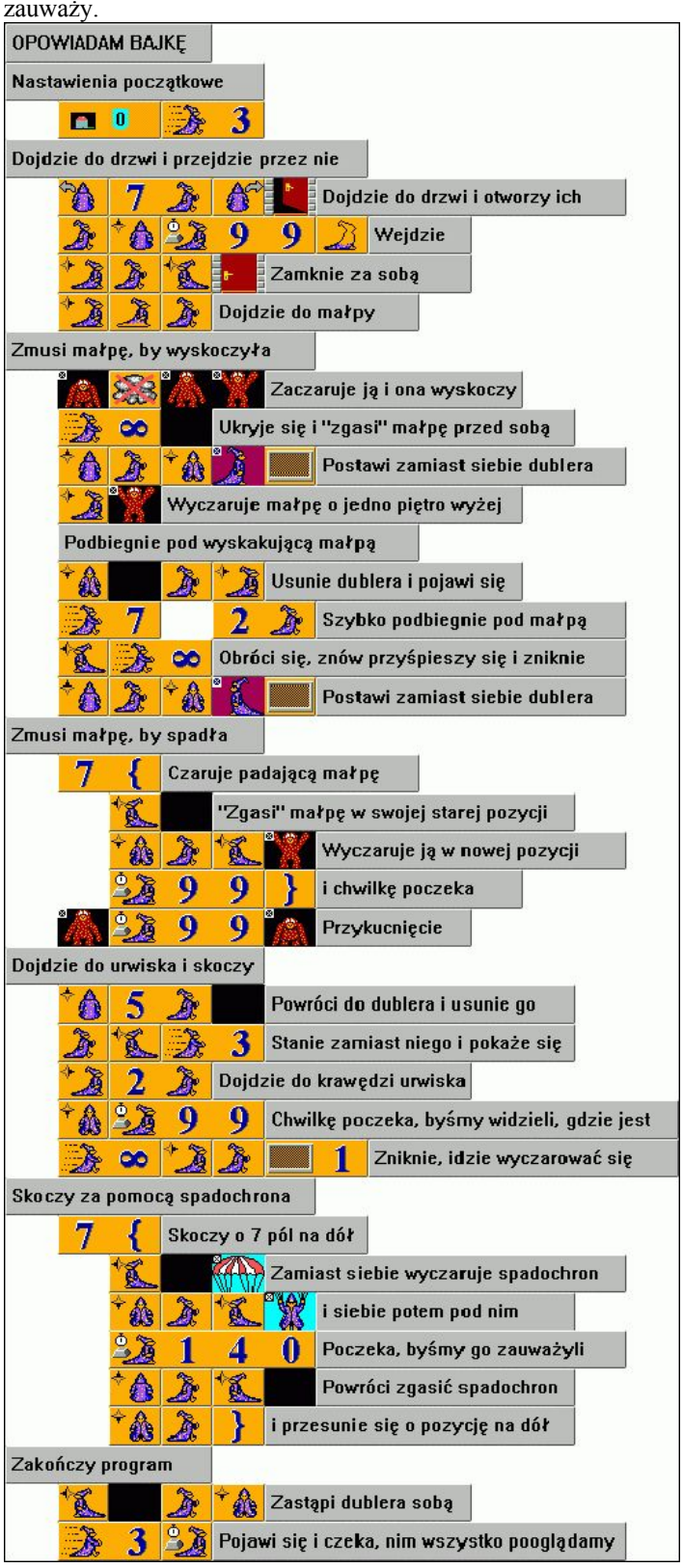
Proste animacje

Baltie najpierw musi podejść do drzwi i je otworzyć. Potrafimy już napisać, jak dojść do drzwi. Zupełnym nowicjuszom podpowiedzmy, że Baltie otworzy drzwi wyczarowując na miejscu zamkniętych drzwi otwarte.

Trochę trudniejsze jest wejście w drzwi. By wszystko wyglądało wiarygodnie, Baltie dojdzie przed drzwi i obróci się twarzą do nich. Potem chwilę poczeka i zniknie. To powinno wywołać wrażenie, że przeszedł przez drzwi.

Popatrzmy na odpowiedni fragment programu i spróbujmy poeksperymentować z różnymi długościami czasu czekania, zanim Baltie zniknie. Wypróbujmy, jakie wrażenie wywołują różne czasy czekania.

Następnie Baltie musi zamknąć drzwi za sobą. Łatwo zgadnąć, że wystarczy wyczarować zamiast drzwi otwartych znowu zamknięte. Musimy jednak najpierw odejść, bo Baltie potrafi czarować tylko przed sobą. Ponieważ właśnie jest niewidzialny, nikt tego nie zauważy.



Rysunek 8.7
Bajka o Baltiem i małpie

Po zamknięciu drzwi pozostaje tylko przesunąć Baltiego do miejsca, gdzie można go znowu zobaczyć, tam pokazać go i doprowadzić do czekającej małpy.

Teraz Baltie powinien zmusić małpę do wyskoczenia. W naszym rozwiązaniu na miejscu małpy wyczaruje znów tę samą małpę. Naszym celem jest żeby na małpie ”wybuchła” chmura dymu i po to wstawiamy czarowanie.

Po rozproszeniu dymu zaczniemy małpę podnosić. W tej chwili pojawiające się chmurki psułyby efekt, więc wyłączymy czarowanie z chmurą. Teraz możemy po kolei czarować podnoszącą się małpę. Tak wywołamy wrażenie wstającej małpy.

Teraz małpa ma podskoczyć. Do tego trzeba wyczarować małpę o jedno pole wyżej. Baltie musiałby przemieścić się, co znowu psułoby efekt. Z tych problemem poradzimy sobie przydzielając Baltiemu nieskończoną szybkość. Efektem ubocznym będzie zniknięcie Baltiego. Aby widz niczego nie zauważył, Baltie wyczaruje na swoim miejscu dublera.

Jeśli popatrzymy na program, zauważymy, że Baltie po zmianie szybkości najpierw przed sobą wyczaruje puste pole. Musiał z tego pola usunąć stojącą małpę, aby mógł ją wyczarować o jedno pole wyżej. Tak wywoła u widza wrażenie, że małpa podskoczyła.

Idąc na miejsce wyczarowania podskakującej małpy Baltie wyczaruje na swojej poprzedniej pozycji dublera, by widzowi wydawało się, że nie poruszał się. Jeśli zrobimy to wystarczająco szybko (ustawiona jest największa szybkość), widz niczego nie zauważy. Warto sprawdzić!

Przezroczystość

Dziwne może się wydawać, że w programie za rysunkiem dublera Baltiego umieszczony jest się element **Przezroczystość.** Na pasku stanu możemy przeczytać informację, że tym poleceniem Baltie przezroczyście czaruje przedmiot 3071 (numer przedmiotu z dublerem Baltiego). Co to znaczy?

Rysunek 8.8
Czaruj przezroczyście

Niektóre przedmioty mają ustalony kolor przezroczysty. Przedmioty takie można poznać po specyficznym oznaczeniu. W banku przedmiotów i w programie mają w one narożniku przekreślony kwadracik, w którym jest kolor ustalony jako przezroczysty (w rozdziale *Możliwości środowiska* podano co ustawić, jeśli Baltie nie pokazuje tych kwadracików). Jeśli taki przedmiot zostanie wyczarowany przezroczyście to zobaczymy prześwitujące tło wszędzie, gdzie w przedmiocie występuje kolor oznaczony jako przeźroczysty.

Tego właśnie potrzebujemy, bo dubler Baltiego stoi na fioletowym tle i ten kolor psuje nam wygląd obrazka. Na szczęście kolor fioletowy jest w tym przedmiocie kolorem przezroczystym. Wyczarujemy więc dublera przezroczyście. Wszystko, co jest w przedmiocie fioletowe, nie będzie widoczne i zobaczymy zamiast tego tło – to jest właśnie to, czego chcieliśmy.

Po uruchomieniu programu Baltie czaruje nieprzezroczyście. Każdy przedmiot wyczaruje dokładnie tak, jak go widać w banku. Jeśli jednak ustawimy czarowanie przezroczyste, poprzez fragmenty namalowane kolorem przezroczystym będzie widać tło, które było przed czarowaniem.

Czarowanie przezroczyste dla jednego konkretnego przedmiotu ustawia się umieszczając w programie za przedmiotem element **Przezroczystość** tak, jak widać w naszym programie. Jeśli chcemy określić, że Baltie ma od tej chwili czarować wszystko przezroczyście, wprowadzamy do programu osobny element **Przezroczystość**, tzn. nie będzie on za żadnym czarowanym przedmiotem. Jeśli będziemy chcieli przełączyć Baltiego z powrotem na czarowanie nieprzezroczyste, wprowadzamy do programu element **Przezroczystość** z numerem zero.

Animacje

Wróćmy do naszego programu. Po wyczarowaniu dublera Baltie podejdzie do małpy i wyczaruje ją o jedno pole wyżej. Wywoła to u widza wrażenie, że małpa podskoczyła. Potem Baltie szybko przebiegnie na dół, usunie dublera, stanie zamiast niego i najwyższą widzialną szybkością (tzn. szybkością 7) przebiegnie pod podskakującą małpką i obróci się, by zobaczyć, co małpiszon będzie robić dalej.

Łatwo przewidzieć następny krok. Baltie, aby mógł poruszać małpą, musi stanąć w innym miejscu. Ponownie zatem wyczaruje na swojej pozycji dublera i ukryty podejdzie do małpy.

Zobaczmy w programie, jak zrealizowany został upadek małpy. Baltie usuwa ją ze starej pozycji, przechodzi o jedno pole w dół i wyczarowuje małpę na nowej pozycji. Potem chwilę poczeka, by można było małpkę zauważyć i cały cykl powtarza się aż do chwili, gdy małpa upadnie na ziemię. W rzeczywistości małpa spadnie jeszcze o jedno pole niżej.

Uzyskamy w ten sposób efekt załamania posadzki pod lądującą małpą. Dla lepszego efektu Baltie zmusi małpiszona do przykucnięcia.

Zwróćmy uwagę na sposób, w jaki jest zaprogramowana pętla. Jeśli będziemy w swoich programach używać pętli, musimy pamiętać, by program na końcu pętli zawsze znajdował się w dokładnie tym stanie, w jakim powinien znaleźć się w następnym kroku.

Gdy Baltie doprowadzi małpiszona na same dno przepaści, wróci i przejmie sterowanie swojego opuszczonego ciała. Podejdzie do krawędzi urwiska i obróci się twarzą do nas. Pozostawi nas zatroskanych jego losem i odejdzie (z nieskończoną szybkością czyli niewidzialnie) wyczarować swój skok ze spadochronem. Przedtem jeszcze ustawi czarowanie przezroczyste, by nie przeszkadzało nam jasnobłękitne tło przedmiotów, z których będziemy składać postać Baltiego na spadochronie.

Rozwiązanie jest ponownie oprogramowane za pomocą pętli. Nie będziemy tego szczególnie opisywać, bo komentarze opisują pętlę wystarczająco.

Zwróćmy tylko uwagę na sposób, w jaki program ustawia Baltiego po każdym obrocie pętli dokładnie tam, gdzie oczekujemy go w następnym kroku.

Końcówka programu jest już jasna i nie potrzebuje komentarzy. Zalecamy uważne przestudiowanie części, która wyświetla Baltiego opadającego na spadochronie. Warto spróbować narysować pojedyncze fazy ruchu, by łatwiej zrozumieć, dlaczego są zaprogramowane właśnie w ten sposób.

Teraz pora wymyślić własną bajkę i nauczyć Baltiego ją przedstawiać. Ciekawe prace chętnie opublikujemy po przesłaniu ciekawych rozwiązań do naszego katalogu, który znajduje się na stronachwww.baltik.cz.

8.2 Czego nauczyliśmy się w tym rozdziale

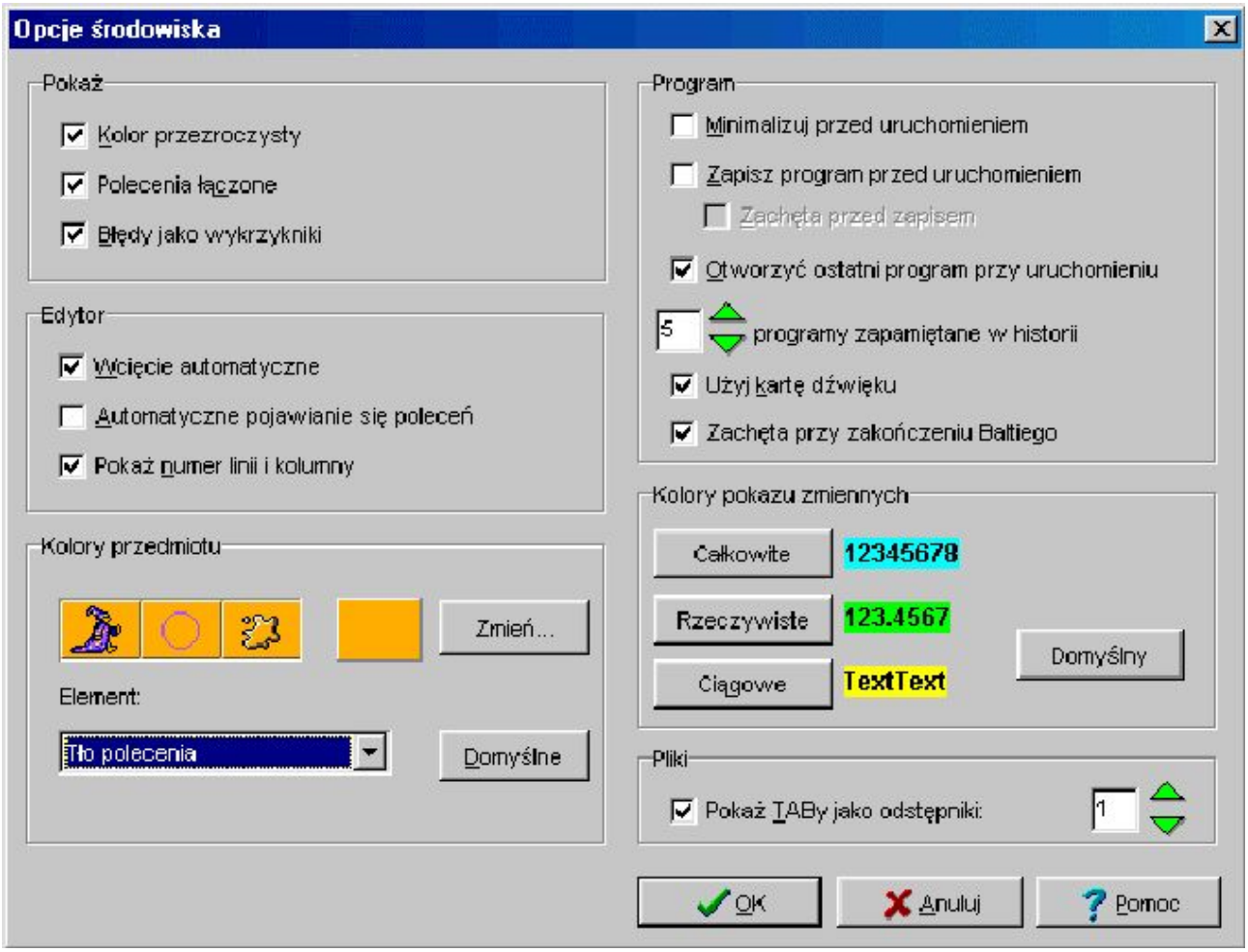
Teraz już potrafimy wyjaśnić Baltiemu, jak ma przedstawić prostą historyjkę. Potrafimy polecić mu nie tylko, by przedmioty czarował, lecz potrafimy z jego pomocą przedmioty ożywiać. Potrafimy zrobić to tak, by widz nie zauważył, że Baltie opuścił swoją pozycję i wędruje gdzieś po dworku. Dodatkowo potrafimy wywołać wrażenie ruchu nie tylko pojedynczych przedmiotów, lecz nawet rysunków złożonych z kilku przedmiotów. Nauczyliśmy się ustawić przezroczystość, czyli cechę przedmiotu decydującą o tym, czy wyczarowane przedmioty zakryją całe swoje pole, czy też pod nimi będzie widać tło, czyli poprzednią zawartość tego pola.

9. Możliwości środowiska

Czego nauczymy się w tym rozdziale

W tym rozdziale pokażemy, jak zmienić niektóre parametry środowiska programistycznego Baltiego. Dowiemy się, jak wyświetlać przezroczystość czarowanych przedmiotów i jak to wyświetlanie wyłączyć. Zobaczymy jak włączyć i wyłączyć efekt łączenia elementów tworzących jedno polecenie, jak ustawić kolory, w których są wyświetlane komentarze i tło poleceń, jak włączyć automatyczne zapisywanie programu przed jego uruchomieniem i szereg innych przydatnych własności, które ułatwią pracę z Baltiem.

Parametry środowiska Baltiego są ustawiane są w oknie dialogu **Możliwości środowiska**, które pokazano na rysunku 9.1. Otwieramy je poleceniem **Możliwości → Środowisko.** Omówimy po kolei ustawienia, z których można skorzystać pracując na poziomie *Nowicjusz.*



Rysunek 9.1
Okno dialogu Możliwości środowiska

Pole Pokaż

W polu **Pokaż** znajdują się trzy pola zaznaczania, z których można użyć dwóch pierwszych.

- Zaznaczeniem pola **Kolor przezroczysty** rozkażemy Baltiemu, by dla przedmiotów, które mają określony kolor przezroczysty, zaznaczał ten kolor. W lewym górnym rogu przedmiotu wyświetlany będzie przekreślony kwadracik z kolorem przezroczystym tego przedmiotu. Jeśli ta opcja zostanie wyłączona, znacznik nie będzie wyświetlany. Więcej informacji na temat przezroczystości podaliśmy w rozdziale *Przezroczystość*.

- Włączając pole **Polecenia łączone** polecimy Baltiemu łączyć elementy tworzące w programie jedno polecenie tak, by wyglądały jak jeden szeroki element.
- Pola **Błędy jako wykrzykniki** użyjemy dopiero na poziomie *Zaawansowany*. Na poziomie *Nowicjusz* nie można popełnić błędu syntaktycznego (niepoprawnie zapisać polecenia), można jedynie coś Baltiemu źle wyjaśnić i wtedy zrobi po prostu coś innego niż spodziewaliśmy się.

Pole Edytor

Również w polu **Edytor** znajdują się trzy pola wyboru i tylko dwa z nich są przeznaczone dla poziomu *Nowicjusz*.

- Zaznaczając pole **Wcięcie automatyczne** rozkażemy Baltiemu, by przy wprowadzeniu elementu **Koniec wiersza** automatycznie wsuwał następny wiersz tworząc wcięcie takie, jak w poprzednim wierszu. Zachęcamy do włączenia tej opcji, bo w ten sposób łatwiej jest przestrzegać konwencji (zwyczajów) tworzenia wcięć w tekście programu.

  Ta właściwość przeszkadza trochę przy przesuwaniu i kopiowaniu bloków, bo Baltie dla nich też doda wcięcia. Jeżeli przesuwany lub kopiowany blok miał już wcięcie, to końcowe wcięcie jest za duże. Ponieważ jednak blok elementów zostaje po przesunięciu czy skopiowaniu zaznaczony, wystarczy chwycić go myszą i zmienić wcięcie – mówiliśmy o tym w rozdziale *Wcięcie bloku*.
- Pola **Automatyczne pojawianie się poleceń** użyjemy dopiero w trybie *Zaawansowany*.
- Włączając opcję **Pokaż numer wiersza i kolumny** spowodujemy, że po prawej stronie paska stanu Baltie będzie pokazywał numer wiersza i kolumny, w której znajduje się element wskazywany myszą. Warto pozostawić tę opcję włączoną.

Pole Kolory elementów

Kontrast szczególnych kombinacji kolorów może różnić się dla niektórych monitorów. Inna kombinacja kolorów może okazać się optymalna dla monitorów notebooków a inna dla monitorów komputerów biurkowych. Ten blok umożliwia zmianę niektórych kolorów środowiska programistycznego Baltiego zgodnie z własnymi upodobaniami.

W liście **Element** Baltie oferuje ustawialne kolory dla trzech elementów środowiska: kolor tła poleceń, kolor tła i kolor tekstu komentarzy. Po ustawieniu koloru, który chcemy zmienić, naciskamy przycisk **Zmień**. Otworzy się okno dialogu **Kolor,** w którym określamy odpowiedni kolor klikając na wybrane pole. Wybór zatwierdzamy naciśnięciem **OK.**

Jeśli nie odpowiada nam żaden z kolorów proponowanych w polu **Kolory podstawowe** możemy nacisnąć przycisk **Definiuj swoje kolory** i wybrać inny kolor. Zalecamy jednak pozostać przy kolorach podstawowych.

Jeśli będziemy chcieli powrócić do domyślnego zestawu, wystarczy nacisnąć przycisk **Domyślne** i wszystkie obiekty będą mieć te kolory, które zaprojektowali twórcy Baltiego.

Rysunek 9.2
Okno dialogu Kolor

Pole Program

W tym polu znajdują się elementy określające, jak Baltie ma traktować całe programy. Jak widać, jest ich sporo.

- Włączając opcję **Minimalizuj przed uruchomieniem** sprawimy, że przed każdym uruchomieniem programu zostanie zminimalizowane okno aplikacji, w którym piszemy program. Jest to przydatne wtedy, gdy nie chcemy, żeby pod dworkiem Baltiego pokazywało się okno z programem. Przy wyłączeniu tej opcji okno z programem zostaje otwarte po uruchomieniu programu, więc po odpowiednim rozmieszczeniu na ekranie okna aplikacji i okna programu można przeglądać program podczas jego wykonywania i tym ułatwić sobie szukanie błędów.
- Pole **Zapisz program przed uruchomieniem** powoduje zapisywanie zmienionego programu przed jego uruchomieniem. Pozwala to na uniknięcie utraty pracy w przypadku np. zaniku zasilania. Jeśli nie chcemy automatycznie zapisywać programu przed uruchomieniem, bo np. tylko tak sobie ćwiczymy i nie chcemy zastępować poprzedniej wersję programu, wyłączamy to pole.

- Kompromisowym rozwiązaniem jest włączenie opcji **Zapisz program przed uruchomieniem** razem z opcją **Zachęta przed zapisem.** Baltie spyta przed każdym zapisaniem zmodyfikowanego programu czy naprawdę go zapisać. Czasami te pytania przeszkadzają.
- Zaznaczając pole **Otworzyć ostatni program przy uruchomieniu** spowodujemy, że Baltie po uruchomieniu bez pytania otworzy program, który był otwarty w momencie ostatniego wyłączenia Baltiego. Ta opcja przydaje się zwłaszcza wtedy, gdy pracujemy nad dłuższym projektem, bo wtedy otwieramy zawsze ten sam program.
- W polu numerycznym **Programy zapamiętane w historii** określamy liczbę ostatnio używanych plików, które Baltie będzie pamiętać i pokazywać na końcu menu **Program.** Większa liczba niekoniecznie musi być lepsza, ale każdy może ustawić wartość dla siebie optymalną.

Pozostałe opcje

Pozostałych opcji użyjemy dopiero na poziomie *Zaawansowany*, więc teraz je pominiemy.

9.1 Czego nauczyliśmy się w tym rozdziale

Potrafimy teraz ustawiać parametry środowiska, w którym pracujemy. Wiemy, jak włączyć i wyłączyć zaznaczanie przezroczystości, łączenie elementów tworzących polecenie, automatyczne ustawianie wcięć i inne przydatne właściwości. Potrafimy zmienić kolor komentarzy i tła elementów, określić liczbę plików i szereg innych parametrów. Krótko mówiąc, potrafimy dostosować środowisko Baltiego do własnych potrzeb.

Część 3:

Programujemy naprawdę

Programowanie na poziomie
Zaawansowany

10. Zostajemy zaawansowanymi programistami

Czego nauczymy się w tym rozdziale

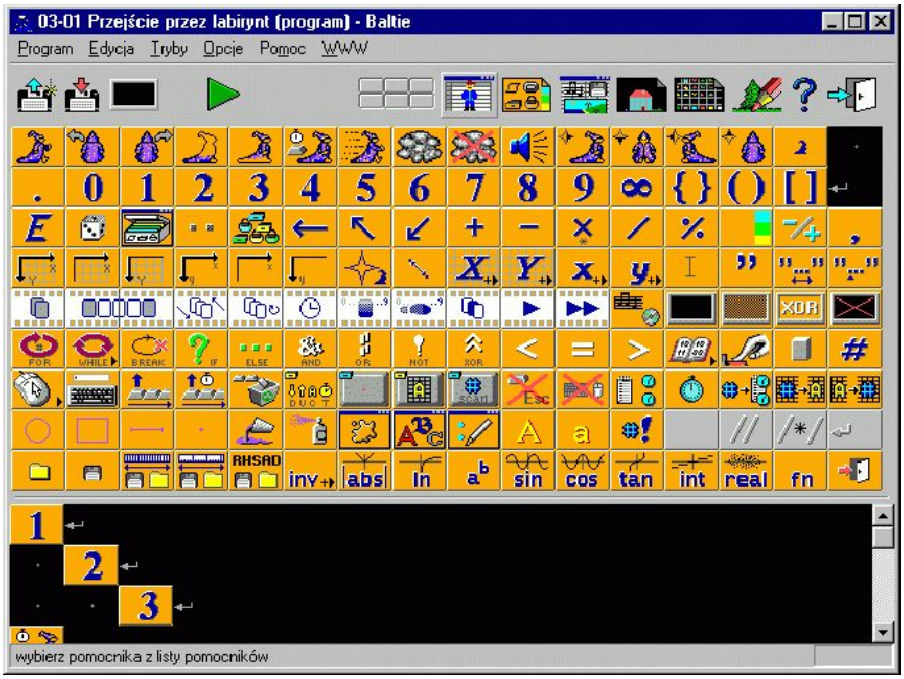
Ten rozdział będzie bardzo krótki. Nauczymy się w nim ustawiać poziom programowania *Zaawansowany* i pokażemy, jak w nim pracować. Przy okazji wyjaśnimy niektóre różnice pomiędzy tym poziomem a trybem *Nowicjusz*.

10.1 Przełączenie na poziom Zaawansowany

W swoich programach wyczerpaliśmy już większość możliwości, które oferuje nam poziom *Nowicjusz*. Najwyższy czas przejść do poziomu *Zaawansowany*. Jak wiemy włącza się go poleceniem **Tryby → Programowanie → Zaawansowany.**

Rysunek 10.1
Przełączenie na poziom Zaawansowany

Po wprowadzeniu polecenia znacznie powiększy się panel elementów, które używane są do tworzenia programów. Elementów jest tak dużo, że przy rozdzielczości ekranu 640×480 pikseli (kto jeszcze używa takich rozdzielczości?) na pisanie programu zostanie tylko trzy wiersze, jak widać na rysunku 10.2. Na takiej małej przestrzeni oczywiście nie można wygodnie programować. Na razie tylko pokażemy, co będziemy mieć do dyspozycji na tym poziomie.



Rysunek 10.2
Początkowy wygląd okna w poziomie Zaawansowany

10.2 Panel elementów

Rysunek 10.3
Zmniejszenie panelu elementów

Kiedy już przejrzymy wszystkie nowe możliwości, możemy zmniejszyć panel elementów do rozmiaru, który będzie nam odpowiadać. Wystarczy wskazać myszą na jego dolną krawędź i gdy wskaźnik myszy zmieni się tak jak widać na rysunku 10.3 na dwie kreski poziome z przyłączonymi dwoma pionowymi strzałkami chwycić krawędź i przesunąć ją do góry.

Możemy ukryć nawet cały panel elementów. Otrzymamy wtedy dodatkowe dwa wiersze przestrzeni dla programu. Jeśli później zmienimy zdanie, wystarczy chwycić dolną krawędź panelu i powiększyć go. Większość użytkowników preferuje jak największą przestrzeń dla programu i pozostawia panel elementów na stałe ukryty.

Nie musimy mieć panelu elementów rozwiniętego aby wprowadzać elementy z panelu do programu. Wystarczy rozwinąć go tylko podczas wybierania elementu – kliknięciem myszką na pustą czarną przestrzeń po prawej stronie programu. Panel pokaże się, poczeka, aż klikniemy na wybranym elemencie i potem ponownie się ukryje. Wybrany element pozostanie chwycony i następnym kliknięciem umieszczamy go w odpowiednim miejscu w programie. Trochę przypomina to umieszczanie przedmiotów na scenie, prawda?

Spróbujmy opisać to krok po kroku:

1. Jeśli panel elementów nie jest ukryty, chwytamy go za dolną krawędź (patrz rysunek 10.3) i przesuwamy krawędź do góry tak, by nie było widać ani jednego wiersza z panelu elementów.
2. Klikamy na dowolnym miejscu czarnej przestrzeni – pokaże się panel elementów.
3. Klikamy w panelu np. na elemencie **Idź** chwytając element. Po kliknięciu panel elementów zniknie, ale wybrany element jest ciągle „trzymany w ręce”.
4. Przesuwamy wskaźnik myszy na wybrane miejsce programu i klikamy upuszczając element w wybranym miejscu programu.

Jeśli niechcący klikniemy na czarnej przestrzeni i nie chcemy wybrać żadnego elementu, możemy rozwinięty panel elementów zawinąć klikając w dowolne miejsce poza nim, nawet w programie. Panel zamknie się i możemy kontynuować pracę.

10.3 Menu lokalne i kopiowanie elementów

Rysunek 10.4
Menu lokalne dla elementu

Jedną z przyjemnych nowości poziomu *Zaawansowany* jest menu lokalne. Menu otwiera się kliknięciem prawym przyciskiem. Zależnie od miejsca, w które klikniemy, będziemy różne menu, zależnie od tego, co możemy w danym miejscu zrobić. Na rysunku 10.4 możemy zobaczyć menu, które otworzy się po kliknięciu na elemencie w programie.

Ponieważ chcemy jak najszybciej zacząć naprawdę programować nie będziemy teraz opisywać tych menu i wrócimy do nich później w odpowiedniej do tego chwili. Mówimy o nich przede wszystkim dlatego, że takie zachowanie Baltiego może zaskoczyć nas podczas kopiowania elementu. Jeśli ktoś jest przyzwyczajony do kopiowania elementu za pomocą kliknięcia prawym przyciskiem i przesunięcia, nie będzie musiał uczyć się kopiować inaczej. Trzeba tylko pamiętać, że gdy chcemy element skopiować na nową pozycję, nie wystarczy teraz tylko kliknąć prawym przyciskiem – otworzy się menu lokalne. Musimy element chwycić, tzn. nacisnąć prawy przycisk myszy i nie zwalniać go. Puszczamy przycisk dopiero w miejscu, na które chcemy skopiować element.

Dla osób z drżącą ręką mamy jeszcze jedną możliwość: po otworzeniu menu lokalnego klikamy polecenie **Kopiuj**. Kopiujemy element do *Schowka,* z którego możemy go potem w odpowiednim momencie wkleić do programu, i to kilka razy.

10.4 Schowek

Przed chwilą napisaliśmy słowo *Schowek* z wielkiej litery i do tego jeszcze kursywą. A to dlatego, że *Schowek* jest częścią *Windows*, z którego usług korzystamy.

Schowek jest bardzo przydatnym miejscem, do którego możemy wstawić każdy potrzebny nam obiekt – tekst, rysunek, część programu, dźwięk – jednym słowem cokolwiek. Pamiętajmy jednak, że w danym momencie może być w *Schowku* tylko jedna rzecz. Gdy wstawimy do niego inny obiekt, poprzednia zawartość *Schowka* zostanie usunięta. Istnieją programy, które umożliwiają korzystanie z większej liczby schowków. Korzystanie ze *Schowka* docenimy wtedy, gdy zmieniając program będziemy przesuwać lub kopiować duże bloki elementów z jednego miejsca programu na inne lub nawet do innego programu.

Wyobraźmy sobie, że mamy gotowy genialny program (nazwijmy go *źródłowym*) i chcemy użyć fragment tego programu w innym ze swoim programie (nazwijmy go *docelowym*). Jak to zrobić?

1. Otwieramy w Baltiem plik z programem źródłowym.
2. Zaznaczamy blok elementów, które chcemy skopiować do programu docelowego.
3. Klikamy na zaznaczonym bloku prawym przyciskiem myszy i w menu lokalnym wybieramy polecenie **Kopiuj** kopiując zaznaczony blok do *Schowka*.
4. Otwieramy w Baltiem plik z programem docelowym.
5. Znajdujemy w nim miejsce, w które chcemy wkleić skopiowaną część programu i klikamy w to miejsce prawym przyciskiem myszy.
6. W menu lokalnym wybieramy polecenie **Wklej tu.** Program wklei blok ze *Schowka* przed przedmiot, na który przed otwarciem menu wskazywała mysz.
7. Jeśli będziemy chcieli wkleić blok elementów do innego programu, możesz ponownie powtórzyć działania od punktu **4**, bo po wklejeniu zawartości *Schowka* jego zawartość pozostaje. Zostanie usunięta dopiero po wprowadzeniu innego obiektu do *Schowka*.

W podobny sposób możemy przesuwać części programów. W takim przypadku przesuwaną część programu do *Schowka* nie kopiujemy, lecz wybieramy polecenie **Wytnij.** Reszta operacji jest dokładnie taka sama.

Ze *Schowka* można skorzystać także wtedy, gdy potrzebujemy wielokrotnie skopiować część programu. Po prostu wklejamy zawartość *Schowka* na jedno miejsce po drugim.

Spróbujmy otworzyć dowolny ze swoich programów, zapisać go do dwóch nowych plików i w tych plikach przećwiczyć to, co opisaliśmy, włącznie z kopiowaniem i przesuwaniem części programu z jednego pliku do drugiego. Po tych eksperymentach można usunąć pliki, by nie przeszkadzały na dysku.

Kopiowanie i przesuwanie części programu za pomocą *Schowka* możliwe jest też na poziomie *Nowicjusz*. Ponieważ nie ma tutaj menu lokalnego, więc trzeba skorzystać z menu **Edycja.**

Ze *Schowka* można korzystać także przy przesuwaniu i kopiowaniu w obrębie jednego programu. Jednak można przesuwać i kopiować bloki w programie także bez wprowadzania ich do *Schowka*. Jeśli zaznaczymy blok, w menu lokalnym pojawią się polecenia **Przesuń zaznaczony blok** i **Skopiuj zaznaczony blok.** Wystarczy wybrać polecenie i zaznaczony blok przesunie się lub skopiuje przed element, na który wskazywała mysz przed otworzeniem menu lokalnego.

10.5 Czego nauczyliśmy się

W tym rozdziale nauczyliśmy się uruchamiać poziom *Zaawansowany* i korzystać z niektórych jego właściwości. Potrafimy np. zmieniać wielkość widocznej części panelu elementów i pracować z ukrytym panelem. Potrafimy otworzyć menu lokalne. Wiemy też, jak kopiować elementy nie otwierając przy tym menu lokalnego. Nauczyliśmy się przesuwać i kopiować części programu za pomocą *Schowka*. W następnym rozdziale zakończymy ćwiczenia ze środowiskiem i zajmiemy się samym programowaniem.

11. Literały i typy danych

Czego nauczymy się w tym rozdziale

W tym rozdziale nauczymy się wprowadzać do programu wartości numeryczne i tekstowe. Zapoznamy się z terminem *literał* i nauczymy się używać elementu z o tej nazwie. Powiemy trochę o typach danych i możliwościach wyświetlenia różnych wartości na ekranie. Nauczymy się używać wyrażeń matematycznych i wyjaśnimy, ile rodzajów operacji dzielenia oferuje nam Baltie. Na końcu rozdziału pokażemy, jak sprawić, by długie teksty nie uciekały poza ekran.

Na poziomie *Nowicjusz* wprowadzaliśmy liczbą powtórzeń poszczególnych poleceń i całych bloków za pomocą elementów z cyframi. Takie wartości wpisywane do programu to **literały**.

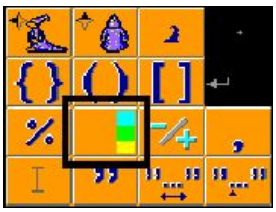
Na poziomie *Zaawansowany* użycie *literałów* nie jest ograniczone do wprowadzania liczby powtórzeń. Możliwości, które oferuje ten poziom są nieporównanie większe. Jedną z nich jest np. wyświetlenie liczby na ekranie. Na rysunku 11.1 widzimy program, który wyświetli liczbę 123 w lewym górnym narożniku dworku Baltiego i poczeka na naciśnięcie klawisza lub kliknięcie przyciskiem myszy.

Rysunek 11.1
Literał złożony z elementów z cyframi

11.1 Element Literał

Wprowadzanie liczb za pomocą elementów z cyframi nie należy do najwygodniejszych. Musimy wprowadzać do programu jeden element po drugim. W dodatku zajmują one w programie dużo miejsca. Poza tym takimi elementami można zapisać tylko liczby. Aby wprowadzić do programu tekst trzeba by dysponować elementami ze wszystkimi literami – tyle miejsca nie mamy.

Możemy rozwiązać te problemy używając elementu **Literał** (na rysunku 11.2 odznaczony jest czarną ramką). Pozwala on do programu wstawiać nawet długie liczby używając jednego elementu. Pozwoli nam też oprócz liczb całkowitych wprowadzać także liczby rzeczywiste a nawet teksty. Element **Literał** zachowuje się podobnie jak element **Komentarz** – też pojawia się w nim (po wstawieniu do programu) kursor tekstowy i Baltie czeka, aż wpiszemy do elementu wartość. Podczas wprowadzania wartości element automatycznie wydłuża się tak, by wprowadzana wartość się zmieściła. Wprowadzanie kończymy jak zawsze klawiszem ENTER lub kliknięciem myszy gdzieś poza elementem.



Rysunek 11.2
Element Literał

11.2 Typy danych

Podczas wpisywania danych do elementu **Literał** może automatycznie zmienić się kolor tła pola, do którego wpisujemy kolejne znaki. Kolor określa **typ** wprowadzanej wartości. **Typ** zaś określa, **gdzie** można użyć tego *literału* i **jak** będzie ta wartość traktowana.

Baltie rozróżnia trzy **typy danych.**

- **Liczby całkowite** mogą mieć wartość od -2 147 483 648 do 2 147 483 647, np. 123.
- **Liczby rzeczywiste,** do których należą liczby całkowite oraz liczby z niezerową częścią dziesiętną (czyli ułamki dziesiętne, np. 123.33). Baltie pamięta te liczby dokładnie do 15 cyfr – dalej wpisuje same zera. Na przykład liczba 12345678901234567890 będzie zapamiętana jako 12345678901234500000.
- **Ciągi znaków** (napisy), które mogą mieć praktycznie nieograniczoną ilość znaków.

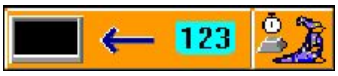
Podczas wpisywania wartości *literału* Baltie postępuje następująco:

1. Dopóki jest w stanie potraktować wprowadzany tekst jako **liczbę całkowitą**, kolor tła to **cyjan** (można powiedzieć jasnobłękitny). Jeżeli wprowadzana liczba przekroczy ograniczenia typu całkowitego lub będzie ułamkiem dziesiętnym będzie traktowana jako **liczba rzeczywista**.

2. Dopóki jest w stanie traktować wprowadzany tekst jako **liczbę rzeczywistą** (lecz już nie całkowitą), kolor tła będzie **zielony**.
3. Jeśli nie jest w stanie traktować tekstu jako liczby, potraktuje wprowadzany tekst jako **ciąg znaków** i kolor tła będzie **żółty.**

Jeśli w programie zajdzie potrzeba potraktowania wprowadzonej liczby całkowitej jako liczby rzeczywistej lub by liczbę potraktować jak tekst, wystarczy podczas edycji kliknąć na odpowiednim kolorze po prawej stronie elementu **Literał**. Baltie zmieni zdanie co do charakteru wprowadzanej wartości. Można zmusić go jednak, jak łatwo możemy sprawdzić, tylko do możliwych konwersji typów. Nie uda się natomiast zmusić Baltiego, by napis **Cześć** potraktował jako numer.

11.3 Piszemy na ekranie



Rysunek 11.3
Wyświetl wartość literału

Na początku tego rozdziału obejrzeliśmy prosty program wyświetlający wprowadzoną liczbę na ekranie. Na rysunku 11.3 możemy zobaczyć ten sam program, w którym do wprowadzenia liczby użyto elementu **Literał**.

Rysunek 11.4
Wyświetl tekst przed Baltiem

Do wyświetlenia na ekranie ciąg znaków nie musimy wprowadzać elementu **Ekran** ani strzałki przypisania. Gdy w programie pojawi się ciąg znaków, Baltie od razu wyświetli go przed sobą. Wypróbujmy to w programie z rysunku 11.4. Przesuńmy potem Baltiego bliżej środka dworku i sprawdźmy, czy Baltie ponownie wyświetli tekst przed sobą.

Gdyby do wyświetlenia tekstu użyć tej samej metody co do wyświetlenia liczby, tekst pokazałby się również w lewym górnym rogu ekranu. Jeśli wyświetlimy kolejno kilka napisów lub liczb, nie będą wyświetlane na sobie, ale będą wyświetlane jeden za drugim. Wypróbujmy to w programie z rysunku 11.5.

Rysunek 11.5
Wyświetl tekst w lewym górnym rogu

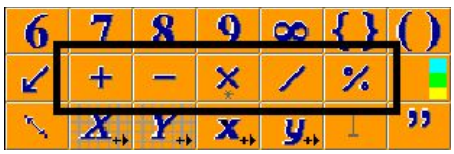
Jeśli wpiszemy program dokładnie tak, jak na rysunku, teksty będą przyklejone do czwórki – patrz rysunek 11.6 z wycięciem lewego górnego rogu dworku. Warto zapamiętać, że **liczby zawsze są wyświetlane bez dodatkowych odstępów.** Jeśli więc chcemy liczby opisać tekstem, należy przygotowując ten tekst pamiętać o odpowiednich odstępach. W naszym przykładzie powinniśmy więc dodać odstęp na końcu pierwszego ciągu znaków za słowem *babka* i na początku drugiego napisu przed słowem *jabłka*.

Rysunek 11.6
Wyświetlony tekst z rys. 11.5

11.4 Wyrażenia matematyczne

Ponieważ najprawdopodobniej wszystko, co tu będzie powiedziane, jest znane, umówmy się, że umieszczamy ten materiał dla uporządkowania pewnych pojęć – zawsze można pominąć ten fragment.

Jak łatwo przewidzieć, na poziomie *Zaawansowany* liczby służą nie tylko do wprowadzania liczby powtórzeń, lecz pozwalają nam też czasami coś obliczyć. Podstawowe operatory matematyczne znajdują się w panelu elementów pod cyframi (patrz rys. 11.7). O dodawaniu, odejmowaniu i mnożeniu nie trzeba specjalnie mówić. Parę niejasności może spowodować dzielenie, więc poświęcimy mu więcej uwagi.



Rysunek 11.7
Operatory arytmetyczne

Zanim przejdziemy dalej, przypomnijmy dwa ważne terminy:

Operand liczba, z którą chcemy wykonać działanie matematyczne.

Operator znak, który odznacza działanie matematyczne. W programowaniu używa się dla oznaczenia mnożenia i dzielenia trochę innych operatorów, niż te znane z lekcji matematyki. Dla mnożenia używa się znaku * (gwiazdka) a dla dzielenia znaku / (łamane).

Specyfika dzielenia, konwersja liczby całkowitej na rzeczywistą

Dzieląc dwie liczby rzeczywiste lub liczbę rzeczywistą przez całkowitą zawsze otrzymamy oczekiwany wynik. Jednak przy dzieleniu dwóch liczb całkowitych sytuacja się komplikuje, bo mogą się pojawić trzy różne wyniki (w nawiasach podamy wartości, które otrzymalibyśmy w wyniku operacji 7 : 2):

- iloraz rzeczywisty (3,5),
- iloraz całkowity (3 – jak pamiętamy 7 podzielone przez 2 daje wynik 3 i resztę 1),
- resztę z dzielenia (1).

Wszystkie trzy wyniki możemy otrzymać w Baltie. Wynik zależy od typu operandów i operatora. Przypomnijmy jeszcze, że pierwszy operand nazywa się **dzielna** a drugi to **dzielnik**.

- Dzieląc liczbę całkowitą przez liczbę całkowitą z użyciem operatora / otrzymamy zawsze **iloraz całkowity**, dlatego operacja ta nazywa się **dzielenie całkowite**. Dla liczb, których nie można podzielić bez reszty, nie sprawdza się formuła **X = (X / Y) * Y.** Przy dzieleniu całkowitym okazuje się, że 5/3=1, 7/3=2 itd. Odrzucamy przecież resztę. Po pomnożeniu ilorazu przez dzielnik otrzymamy zatem wyniki postaci (5/3)*3=3 , bo przecież wynikiem dzielenia było 1, (7/3)*3=6 , bo wynikiem dzielenia było 2, itd., co niezupełnie jest zgodne z intuicją.
- Gdy chcemy otrzymać **iloraz rzeczywisty**, przynajmniej jeden z operandów musi być liczbą rzeczywistą. Jeśli jednym z operandów jest literał, nie ma problemu – po prostu podczas wprowadzania jego wartości klikniemy na zielony kwadracik (patrz radę z rozdziału 11.2). Jeśli operandem jest jednak jakieś wyrażenie matematyczne (a w zasadzie jego wynik), musimy wynik jawnie przetłumaczyć (mówimy: dokonać konwersji) na liczbę rzeczywistą. Do tego służy element **Liczba rzeczywista** – znajduje się przy dolnej krawędzi panelu elementów, trzeci element od prawej strony – patrz rysunek 11.8. Element umieszczamy przed liczbą lub wyrażeniem, dla którego chcemy dokonać konwersji.

Rysunek 11.8
Element Liczba rzeczywista

- Jeżeli interesuje nas **reszta z dzielenia** dwóch liczb całkowitych, powinniśmy użyć operandu **%**. Podaje on resztę z dzielenia pierwszego operandu (dzielnej) przez drugi (dzielnik). Napiszmy program podobny jak na rysunku 11.9 żeby to wszystko wypróbować. Wypróbujmy też przykłady z innymi liczbami. Warto wprowadzić przy wyświetlaniu liczb średnik lub inny znaku oddzielający liczby, aby oddzielić liczby od siebie na ekranie.

Rysunek 11.9
Trzy rodzaje dzielenia

Błędnie wprowadzony typ jest jednym z bardzo częstych błędów programisty. W poprzednim przykładzie widzieliśmy, jak prosta zmiana typu może zmienić cały wynik. Uważajmy więc w swoich programach na to, czy tam, gdzie powinna być liczba całkowita, jest rzeczywiście liczba całkowita. Nie wystarczy, że jest tam np. liczba 2, powinna być jeszcze umieszczona na polu w kolorze cyjan a nie zielonym. Sprawdzajmy dwukrotnie, czy tam, gdzie powinna być liczba rzeczywista, jest liczba rzeczywista (tło zielone), a gdzie powinien być napis jest rzeczywiście ciąg znaków (tło żółte).

Konwersja liczby rzeczywistej na całkowitą

W programach będziemy często dokonywać konwersji liczby rzeczywistej na całkowitą. Metod konwersji jest sporo. Decyzja, którą zastosować, zależy od tego, która metoda jest w danym przypadku najlepsza. Najczęściej używaną metodą jest proste odcięcie części dziesiętnej czyli ułamkowej. Może to dziwne, ale w programowaniu używa się go częściej, niż klasycznego zaokrąglenia. Tę operację załatwi element **Część całkowita,** który znajduje się przy dolnej

krawędzi panelu elementów na lewo od elementu **Liczba rzeczywista** (jest na nim napisane **int**).

Oprócz odcięcia części dziesiętnej używa się też innych sposobów konwersji, na przykład poprzez zaokrąglenie. Dowiemy się więcej o konwersjach lub innych funkcjach upuszczając element konwersji na znak zapytania (pomocy). Pomoc otworzy się dokładnie tam, gdzie znajdują się potrzebne informacje.

Wyświetlanie liczby przed Baltiem – konwersja liczby na ciąg znaków

Rysunek 11.10
Element Konwersja na ciąg znaków

Może wydawać się to nie w porządku, że liczby nie mogą zostać wyświetlone wprost przed Baltiem. Jeśli chcielibyśmy wyświetlić jedną konkretną liczbę, nie byłoby problemu – podczas wprowadzania wartości odpowiedniego literału powiedzielibyśmy po prostu Baltiemu, że nie jest to liczba, ale ciąg znaków (patrz rada na końcu części 11.2). Jak jednak postąpić, gdy zechcemy wyświetlić wynik jakiegoś obliczenia?

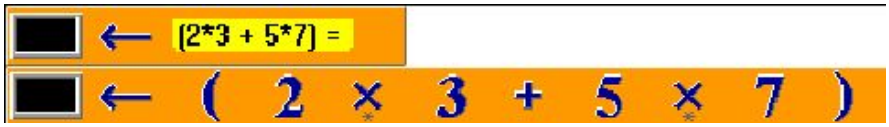
Rozwiązanie jest dość proste: użyjemy elementu **Konwersja na ciąg znaków,** który w panelu elementów znajduje się zaraz pod elementem **Literał** (patrz rysunek 11.10). Element ten wprowadzimy przed liczbę. Jeśli jednak nie chcemy konwertować pojedynczej liczby ale bardziej skomplikowane wyrażenie, powinniśmy najpierw zamknąć to wyrażenie wnawiasy ( ). Wtedy Baltie najpierw obliczy wartość wyrażenia i dopiero potem dokona konwersji liczby (wyniku) na ciąg znaków.

Rysunek 11.11
Wyświetlenie liczby przed Baltiem

Program z przykładem konwersji znajdziemy na rysunku 11.11. Wpisując program do komputera pamiętajmy o umieszczeniu na końcu programu polecenia czekania na klawisz. Nie zapomnijmy też sprawdzić, czy Baltie traktuje rzeczywiście liczby jako liczby, czyli czy tło jest cyjanowe lub zielone.

Wyświetlenie liczby przed Baltiem nie jest jednak najważniejszym zastosowaniem konwersji liczby na ciąg znaków. Najprawdopodobniej nikt tego nie używa – chcieliśmy tylko pokazać jedną z ciekawych operacji.

Znacznie bardziej przydatne jest dokonywanie konwersji liczby na tekst w innych sytuacjach. Jedną z nich jest wyświetlenie na ekranie. Gdy zechcemy np. wyświetlić wartość wyrażenia (2*3 + 5*7), możemy to zrobić albo za pomocą dwóch poleceń, jak jest to pokazane na rysunku 11.12, albo za pomocą jedynego polecenia, korzystając z możliwości konwersji liczby na ciąg znaków, jak można to zobaczyć na rysunku 11.13.



Rysunek 11.12
Wyświetlenie wyrażenia za pomocą dwóch poleceń

Drugi sposób wydaje się znacznie bardziej czytelny, można go używać np. w kontrolnym wypisywaniu wartości podczas szukania błędów w programie.

Rysunek 11.13
Wyświetlenie wyrażenia za pomocą jednego polecenia

11.5 Łączenie napisów i dzielenie wiersza

Przy wyświetlaniu napisów (ciągów znaków) możemy skorzystać z tego, że dwa sąsiednie literały tekstowe zostaną automatycznie połączone i Baltie będzie je traktować jak pojedynczy napis.

Rysunek 11.14
Źle podzielony długi napis

Ta właściwość często się nam przyda. Długie napisy, które nie mieszczą się na ekranie („wypływają" z obszaru roboczego), będziemy mogli podzielić na dwa ciągi znaków, z których każdy zostanie wyświetlony w kolejnym wierszu. Należy uważać tylko, by nie napisać programu podobnie jak na rysunku 11.14. Jeśli podzielimy napis elementem **Koniec wiersza**, otrzymamy dwa niezależne polecenia:

- Pierwsze polecenie wyświetla ciąg znaków za ostatnio wyświetlonym tekstem.
- Drugie polecenie jest osobnym napisem, co jest, jak pamiętamy, poleceniem dla Baltiego, żeby wyświetlił ten tekst przed sobą. Warto wypróbować, czym to się skończy.

Jeśli chcemy skorzystać z możliwości podzielenia napisu na dwa ciągi znaków w dwóch wierszach, powinniśmy zamiast elementu **Koniec wiersza** wprowadzić element **Podzielenie wierszy** (znajduje się między komentarzami) tak jak to widać na rysunku 11.15.

Rysunek 11.15
Poprawnie podzielony długi napis

Zauważmy, że oprócz innego znacznika końca wiersza inaczej wygląda też odstęp – nazywa się tutaj **Odstępnik komentarzowy**. Element **Odstęp** dzieli polecenia tak samo jak **Koniec wiersza** , a przy zmianie znacznika końca wiersza Baltie zmienia też odstępy tworzące wcięcia. Elementu **Odstępnik komentarzowy** nie można znaleźć w panelu elementów. Baltie sam wstawi go na początek wiersza.

Rysunek 11.16
Długi napis podzielony na więcej wierszy

W ten sposób można podzielić nie tylko ciąg znaków, lecz także długie wyrażenie matematyczne lub inne elementy programu. Typowym przykładem są długie polecenia wyświetlenia. Dzieląc wiersze w programie pamiętajmy, by program pozostał jak najbardziej czytelny. Nie zapominajmy o możliwości wcięcia wiersza (jeżeli to konieczne, Baltie sam wstawi odstępy komentarzowe). Czytelny program jest znacznie bardziej zrozumiały, poza tym czytelność znacznie przyśpiesza wyszukiwanie i usuwanie błędów.

To, że jakiś fragment programu nie mieści się na ekranie, nie ma znaczenia w programie. Jeżeli chcemy zobaczyć długie wiersze elementów, można np. skorzystać z paska przewijania poziomego. Lepiej jednak podzielić długie wiersze tak, by były widoczne wszystkie elementy programu.

Czasami po podzieleniu wierszy Baltie wyświetli na początku nowego wiersza wykrzyknik odznaczający błąd. W większości przypadków nie chodzi o błąd, lecz o rozdzielenie elementów tworzących razem jedno polecenie.

11.6 Czego nauczyliśmy się

W tym rozdziale po raz pierwszy zaczęliśmy pracować z danymi i nauczyliśmy się wprowadzać do programu wartości numeryczne i tekstowe. Wyjaśniliśmy, co to jest literał, i pokazaliśmy, jak można tego elementu użyć do wprowadzania liczby i tekstów do programu. Przy okazji zobaczyliśmy, że Baltie rozróżnia trzy typy danych: liczby całkowite, liczby rzeczywiste i ciągi znaków czyli napisy. Nauczyliśmy się wyświetlać wartości literałów na ekranie i pokazaliśmy różnicę między traktowaniem wartości liczbowych i tekstowych. Poznaliśmy umiejętności matematyczne Baltiego i pokazaliśmy trzy sposoby dzielenia liczb całkowitych. Skorzystaliśmy z możliwości konwersji między liczbami całkowitymi i rzeczywistymi oraz pomiędzy liczbami i napisami. Na koniec wypróbowaliśmy, jak można w programie podzielić długi ciąg znaków tak, by nie uciekał poza krawędź obszaru roboczego.

Pamiętajmy o kontrolowaniu typów wpisywanych danych. Zawsze musimy zwracać na to szczególną uwagę kontrolując pojawiający się kolor tła.

12. Używamy współrzędnych ekranowych

Czego nauczymy się w tym rozdziale

W tym rozdziale nauczymy się posługiwać współrzędnymi ekranu. Najpierw powiemy, co to są współrzędne ekranowe i jakich rodzajów współrzędnych można używać w systemie Baltie. Potem wyjaśnimy, jak wskazać, w którym miejscu ekranu wyczarować przedmiot, bez względu na to, gdzie w obrębie dworku znajduje się Baltie. Pokażemy, że współrzędne można wprowadzać nie tylko podając liczbę, lecz też ustalać je np. na podstawie bieżącej pozycji kursora myszy lub z położenia innych obiektów. Tę część zakończymy prostym programem, który pozwoli dwóm graczom zagrać w kółko i krzyżyk. Potem skorzystamy ze współrzędnych, by umieścić na ekranie w wybranych przez nas miejscach różne informacje. Powiemy, jakie dodatkowe możliwości wyświetlania przedmiotów na ekranie oferuje Baltie. Na koniec pokażemy, że współrzędne można nie tylko ustawiać, lecz także otrzymywać od innych obiektów i spróbujemy zbudować program, w którym skorzystamy z tej możliwości.

W poprzednim rozdziale pokazaliśmy, jak wyświetlać liczby i teksty na ekranie. Tekst komunikatu mogliśmy wyświetlić albo w lewym górnym rogu ekranu, za ostatnio wypisanym tekstem, albo przed Baltiem. Teraz pokażemy, jak umieścić teksty i liczby praktycznie gdziekolwiek na obszarze dworku Baltiego.

12.1 Najpierw trochę teorii

Jak już wiemy, cała przestrzeń Baltiego (żeby było sympatyczniej nazywamy ją dworkiem) jest podzielona na pola, które tworzą 10 wierszy po 15 kolumn, jak na szachownicy. Poszczególne pola mają rozmiar taki, jak pojedynczy przedmiot. Jeśli popatrzymy na rysunki przedmiotów w powiększeniu, zobaczymy, że przedmioty składają się z niewielkich pikseli poukładanych w szachownicę liczącą 29 wierszy i 39 kolumn. Nietrudno więc obliczyć, że przestrzeń Baltiego liczona w *pikselach* składa się z 290 linii poziomych i 585 pionowych. Zapisujemy to jako rozmiar 585 x 290 co oznacza, że możemy postawić w poziomie 585 kropek, ale w pionie tylko 290. Rozmiar dworku liczony w polach wynosi 15 x 10 pól.

Każde pole i każdy piksel mają swoje **współrzędne** czyli uporządkowaną parę liczb, określającą pozycję na dworku („uporządkowany" oznacza, że ważna jest kolejność liczb):

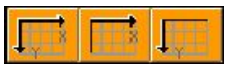
- pierwsza liczba jest **współrzędną poziomą**, która oznacza się literą **X** lub **x** i określa pozycję elementu w wierszu liczoną od lewej strony,
- druga liczba jest **współrzędną pionową**, którą oznacza się literą **Y** lub **y** i określa pozycję elementu w kolumnie, liczoną od góry ekranu.

Pozycję zaczynamy obliczać zawsze od zera. **Lewy górny narożnik** dworku ma zatem współrzędne **(0, 0)**.

Jeśli będziemy w tekście podawać współrzędne, zawsze będziemy ujmować je w nawiasy ( ) i oddzielać przecinkiem.

Baltie rozróżnia dwa rodzaje współrzędnych:

- Współrzędne **pola** oznaczane będą **wielkimi literami (X, Y)** i do ich określania używane będą elementy z podziału dworku na pola (patrz rysunek 12.1).



Rysunek 12.1
Współrzędne pola

- Współrzędne **piksela** oznaczane będą **małymi literami (x, y)** (można to łatwiej zapamiętać gdy zauważymy, że piksel jest mniejszy od pola) i do ich ustalenia używane będą elementy bez podziału dworka (patrz rysunek 12.2).

Rysunek 12.2
Współrzędne piksela

12.2 Czarujemy na współrzędnych

Teraz pokażemy na przykładach, jak wykorzystać współrzędne w programach. Zaczniemy od „czarującego” Baltiego. Do tej pory Baltie potrafił wyczarować przedmiot tylko bezpośrednio przed sobą. Za pomocą współrzędnych możemy nauczyć go czarować gdziekolwiek na obszarze dworku bez względu na to, w którym miejscu Baltie stoi. Wystarczy w programie za przedmiotem wprowadzić element współrzędnych i jedną lub dwie współrzędne.

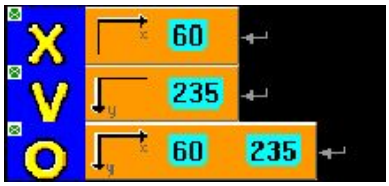
Współrzędne określone za pomocą liczb

Uruchamiając programu z rysunku 12.3 możemy sprawdzić, że zamiast współrzędnej, której nie podamy, Baltie użyje odpowiedniej współrzędnej pola, które jest przed nim. Jeśli nie wprowadzimy żadnych współrzędnych, wyczaruje przedmiot wprost przed sobą (to już znamy). Jeśli Baltie jest podczas uruchomienia naszego programu w pozycji początkowej, to:

- przedmiot **X** umieszczony zostanie w piątej (liczone od zera) kolumnie ostatniego wiersza (wiersza, w którym stoi Baltie, tzn. wiersz numer 9), czyli w polu o współrzędnych (5, 9),
- przedmiot **Y** zostanie umieszczony w piątym wierszu (znów liczymy od zera) w kolumnie przed Baltiem (kolumna 1), czyli w polu o współrzędnych (1, 5),
- przedmiot **O** umieszczony zostanie w polu o współrzędnych (5, 5).

Rysunek 12.3
Przedmioty na współrzędnych pola

Do czarowania przedmiotów nie musimy używać tylko współrzędnych pól. Jak widać w programie 12.4, Baltie potrafi też wyczarować przedmiot mając podane współrzędne piksela. Trzeba przy tym zwrócić uwagę na to, że **współrzędną piksela dla przedmiotu to współrzędna piksela lewego górnego rogu przedmiotu**.



Rysunek 12.4
Przedmioty na współrzędnych piksela

W pierwszym programie celowo użyto literałów złożonych z elementów cyfrowych, czyli konstrukcji, do stosowania której byliśmy zniechęcani w rozdziale o literałach. Chcieliśmy tu wyraźnie pokazać, że wprowadzając dwie sąsiednie liczby musimy tutaj użyć elementu **Przecinek**, bo inaczej Baltie połączy elementy i będzie je traktować jak jedną liczbę (w naszym przypadku byłoby to 55).

W drugim programie wykorzystaliśmy literały w inny sposób. Jak widać w ostatnim wierszu programu, podczas wprowadzania liczb za pomocą elementu **Literał** nie potrzebujemy przecinka do ich rozdzielenia. Jeśli jednak go wprowadzimy nic złego się nie stanie. Baltie bez problemu rozróżnia obie liczby.

Współrzędne określone za pomocą myszy

Współrzędne, na których ma być wyczarowany przedmiot, można określać także inaczej niż liczbą. Możemy np. rozkazać Baltiemu, by wyczarował przedmiot w miejscu, gdzie ostatnio kliknęliśmy myszą. Używa się do tego elementu **Mysz**, który znajduje się przy lewej krawędzi panelu elementów (patrz rysunek 12.5).

Rysunek 12.5
Element Mysz

Gdy popatrzymy uważniej na ikonę elementu, zauważymy, że po prawej stronie pod rysunkiem myszy jest mała czarna strzałka na prawo. Ta strzałka oznacza trochę inne zachowywanie się elementu niż to, którego się spodziewamy. Elementy ze strzałką po „upuszczeniu” nad miejscem w programie nie włożą się do programu, lecz najpierw rozwiną paletę kilku wyspecjalizowanych pod-elementów, z których wybierzemy ten, który zostanie ostatecznie wprowadzony do programu.

Rysunek 12.6
Paleta myszy

Element **Mysz** oferuje paletę sześciu możliwości (patrz rysunek 12.6), z których wybieramy jedną w zależności od tego, czy chcesz użyć bieżącej pozycji wskaźnika myszki, czy też współrzędnych punktu, w którym został naciśnięty jeden z przycisków myszy. Nie będziemy tu opisywać szczegółowo znaczenia wszystkich pod-elementów – na dowolny z nich można wskazać myszą i przeczytać opis w pasku stanu lub przenieść element **Mysz** na znak zapytania pomocy.

Zmontujmy teraz prosty program, który będzie w nieskończonej pętli ciągle wyczarowywać jakiś przedmiot (np. żółtą literę **x** z banku 7) na pozycji, na której klikniemy lewym przyciskiem myszy

(element w lewym górnym rogu palety). Przykładową wersję można zobaczyć na rysunku 12.7.

Spróbujmy przetestować program i sprawdzić, czy Baltie zawsze wyczaruje przedmiot na pozycji, na którą wskazywała mysz podczas ostatniego kliknięcia. W tym punkcie powinien znaleźć się lewy górny narożnik przedmiotu. Czarowanie powinno się udać nawet wtedy, gdy część przedmiotu nie mieści się w dworku Baltiego. Zauważmy przy tym, że podczas czarowania na współrzędnych piksela nie pojawia się znajoma chmurka czarowania.

Rysunek 12.7
Czaruj na współrzędnych myszy

Teoretycznie w programie nie było konieczne użycie elementu **Współrzędne piksela**. Gdy Baltie w programie znajdzie za przedmiotem element z „mysiej rodziny", przypuszcza, że przedmiot określa współrzędne, na których ma czarować. Będzie zachowywał się tak, jakby określone zostały współrzędne piksela. Polecamy jednak nie korzystać z takich możliwości w programach i pisać je tak, by były czytelniejsze.

Bardzo szybko zauważymy, że czas pozornie zaoszczędzony przy tworzeniu programu poprzez niewstawianie niektórych elementów nie da się porównać z czasem, który możemy stracić szukając błędów w nieczytelnym programie pełnym sztuczek i trików.

Zmienimy teraz program zastępując element **Współrzędne piksela** elementem **Współrzędne pola**. Różnica jest na pierwszy rzut oka niedostrzegalna (patrz rysunek 12.8), lecz program zachowuje się inaczej. Przedmiot jest wyczarowany dokładnie na polu, na którym klikamy myszą. Podczas czarowania pojawia się chmurka!

Rysunek 12.8
Wyczaruj przedmiot na kliknięty polu

Program ten ujawnia pewną cechę, która mogła zostać niezauważona w poprzednim przykładzie. Baltie nie czeka, aż klikniemy myszą. Bez końca czaruje i czaruje przedmiot na polu, na którym **ostatnio** kliknęliśmy myszą. Powinniśmy pozwolić mu odpocząć przez chwilę. Wystarczy rozkazać, by najpierw poczekał na kliknięcie myszy i dopiero potem wyczarował przedmiot.

12.3 Gra w kółko i krzyżyk

Wiemy już tyle, że możemy napisać program, który pozwoli dwóm graczom grać na komputerze w kółko i krzyżyk. Będą wykonywać ruchy po kolei. Każdy z nich kliknie myszą na polu, na którym Baltie powinien umieścić jego znak.

By gracze mogli lepiej orientować się w grze trzeba zaznaczyć pozycje umieszczania krzyżyków i kółek. Jedną z możliwości jest wyczarowanie na całym dworku przedmiotu z kropką w środku – weźmy np. przedmiot, który jest w banku 7 między dużym i małym **O** (przedmiot numer 7105). Przypominamy, że numer przedmiotu pojawi się na lewej krawędzi paska stanu, gdy wskażemy na przedmiot w banku.

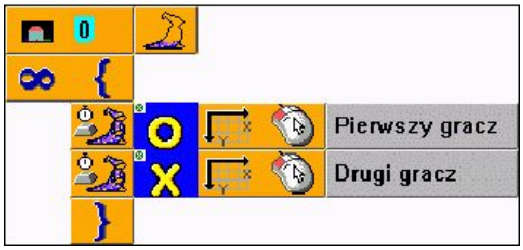
Obszar roboczy możemy wypełnić przedmiotem w dwojako:

– albo stworzymy odpowiednią scenę i wstawimy ją na początku do programu,

– albo przed uruchomieniem gry rozkażemy Baltiemu z nieskończoną szybkością przebiec cały dworek i wyczarować na każdym polu odpowiedni przedmiot.

W przykładzie na rysunku 12.9 wykorzystana jest pierwszą możliwość, bo daje nam krótszy program. Jeśli jednak pamiętamy, jak wypełnić całą przestrzeń, można wybrać drugą metodę. Jej zaletą jest możliwość łatwej zamiany krzyżyków i kółek na przykład na bańki z czerwoną i błękitną wodą (z banku 1) a kropki pustych pół na puste bańki.

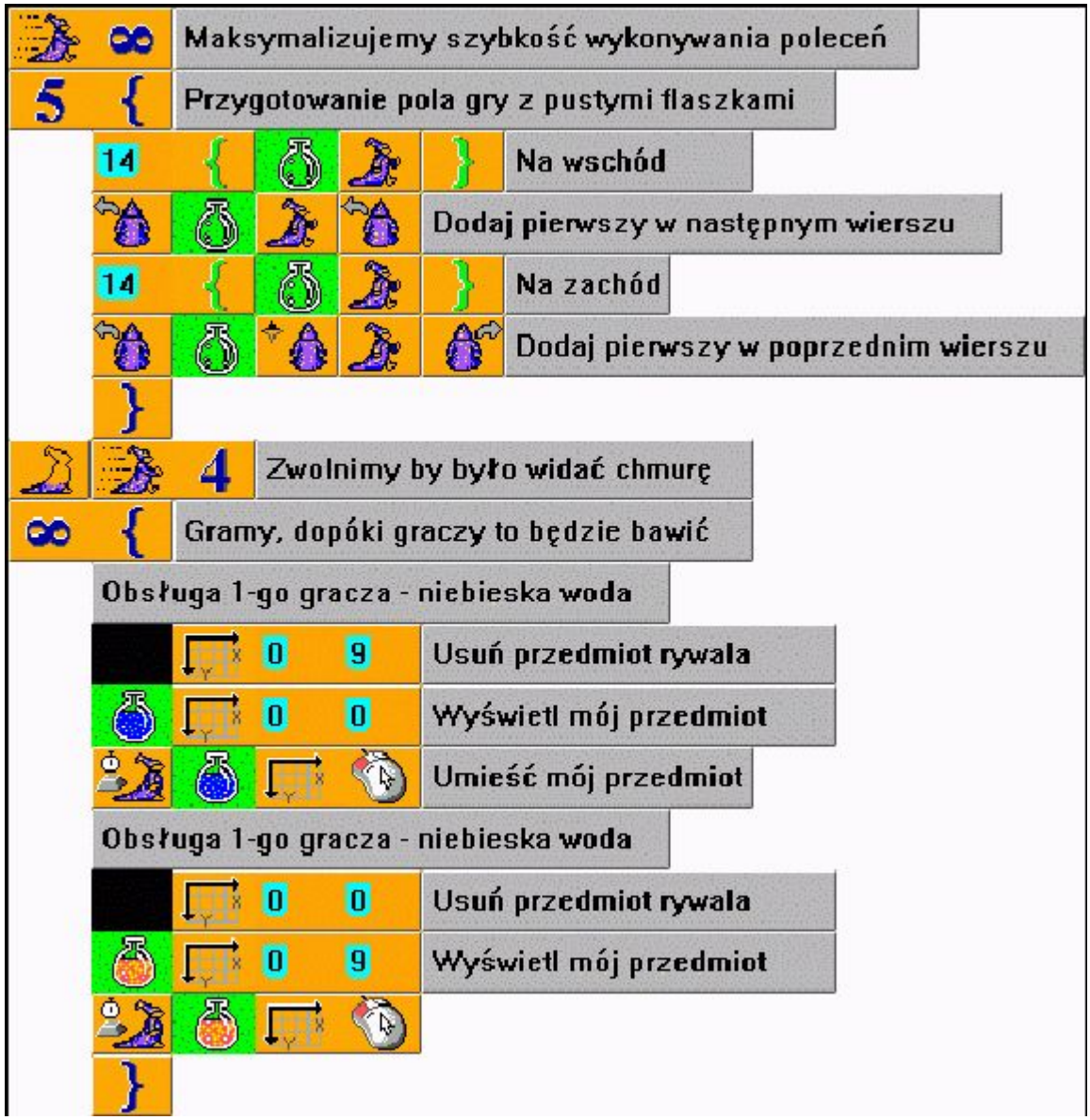
Na koniec wykładu o czarowaniu na współrzędnych zadanie domowe: napiszmy program gry w kółko i krzyżyk tak jak poprzednio. Program jednak nie może używać sceny, ale Baltie na wypełni sam całą przestrzeń. Pominie jednak pierwszą kolumnę, w której przed każdym ruchem będzie wyświetlać znak gracza, który ma wykonać ruch.



Rysunek 12.9
Kółko i krzyżyk

Żeby nie było to za proste, znak pierwszego gracza będzie umieszczany na górnej krawędzi dworku a znak drugiego gracza przy dolnej krawędzi dworku. Pamiętajmy, że zawsze ma być widoczny tylko jeden znak tak, by gracze mogli rozpoznać, czyja kolej.

Jeszcze raz przypominamy o pilnowaniu przy wprowadzaniu literałów kolorów tła – liczby całkowite są jasnobłękitne. Baltie nie lubi napisów podanych jako współrzędne. Spróbujmy teraz wszystko oprogramować samodzielnie i możesz porównać rozwiązanie z wersją na rysunku 12.10.



Rysunek 12.10
Gra w kółko i krzyżyk
(wszystkie liczby są całkowite – błękitne tło!)

12.4 Wyświetlanie wartości na współrzędnych

Czarowanie już wypróbowaliśmy, teraz zobaczmy, jak z pomocą współrzędnych dostosować sposób wyświetlenia wartości na ekranie (czasem mówimy: współrzędne wyjścia). Podczas wprowadzania współrzędnych miejsca wyświetlania obiektu na ekranie są aż dwa miejsca w programie, gdzie można wstawić elementy opisujące współrzędne wyjścia:

– pierwsze to miejsce pomiędzy elementem **Ekran** i strzałką przypisania,
– drugie to koniec polecenia wyświetlenia.

Rysunek 12.11
Możliwości wprowadzenia współrzędnych wyjścia

Wybieramy miejsce, które w tym momencie bardziej nam odpowiada.

Trzy sposoby wyświetlania

Rysunek 12.12
Wyświetlanie liczby na współrzędnych myszy

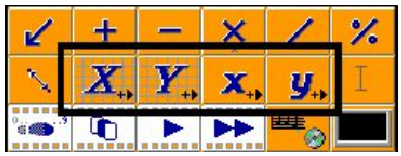
Ponieważ zwykłe wyświetlanie będzie już dla nas nudne, wypróbujemy wyświetlanie ruchome – będziemy wyświetlać na bieżących współrzędnych myszy. Odpowiedni element wybierzemy z palety pokazanej na rysunku 12.6. Przygotujmy scenę z tłem (dowolnym, tylko nie czarnym) i spróbujmy uruchomić program na rysunku 12.12. Gdy przebiegniemy przez dworek Baltiego wskaźnikiem myszy, będzie za nim podążać czarna linia z wyświetlaną liczbą.

Teraz spróbujemy program zmodyfikować dodając na końcu polecenia wyświetlenia (przed nawiasem zamykającym) element **Przezroczystość**. Po uruchomieniu zobaczymy, że podczas poruszania myszy już nie rysuje się czarny ślad, lecz liczba jest wyświetlany bezpośrednio na tle dworku. Teraz za myszą pozostaje biały ślad po wyświetlanych liczbach.

Czy nie można sprawić, by za myszą nie było żadnego śladu? Można. Wystarczy za element **Przezroczystość** wprowadzić liczbę całkowitą większą od 1 (patrz rysunek 12.13). Zmieni to wyjście przezroczyste na animowane i gotowe.

Rysunek 12.13
Wyświetlanie animowane na współrzędnych myszy

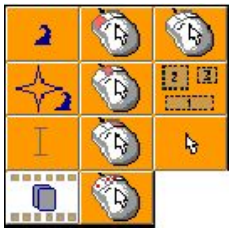
12.5 Otrzymywanie współrzędnych



Rysunek 12.14
Elementy do otrzymania współrzędnych

Współrzędne można nie tylko podawać, lecz także otrzymywać. Służą do tego cztery elementy, które znajdują się w panelu elementów pod elementami operatorów matematycznych (patrz rysunek 12.14). Jak widać, ikony wystarczająco określają przeznaczenie elementów – elementy z wielkimi literami służą do otrzymania współrzędnej X lub Y pola, elementy z małymi literami do otrzymania współrzędnej x lub y piksela.

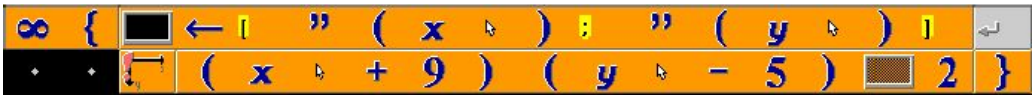
Po umieszczeniu elementu w programie Baltie rozwinie paletę pod-elementów określających obiekt, którego współrzędne chcemy otrzymać (patrz rysunek 12.15).



Rysunek 12.15
Współrzędne czego?

Jak widać oprócz znajomych elementów z myszą jest tu kilka nowych. Ich znaczenie można znaleźć na pasku stanu po wskazaniu elementu myszą.

Teraz już wiemy wszystko potrzebne co jest potrzebne do napisania program, który wyświetla obok wskaźnika myszy jego współrzędne – np. w formie **[x ; y]** (tzn. otwierający nawias kwadratowy, współrzędna x, średnik, współrzędna y, zamykający nawias kwadratowy). Spróbujmy go napisać. Przykładowe rozwiązanie można zobaczyć na rysunku 12.16. Jak wyglądałby program wypisujący współrzędne w postaci **(x, y)**?



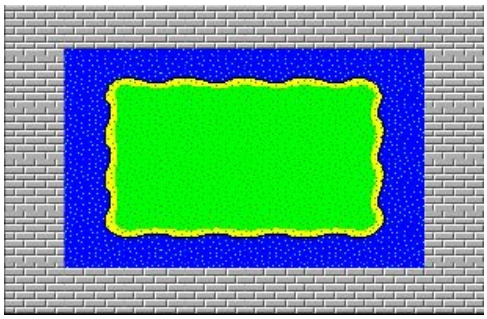
Rysunek 12.16
Wyświetlaj obok wskaźnika myszy jego współrzędne w formie [x ; y]

Jak widać nie wyświetlamy tekstu dokładnie na pozycji wskaźnika myszy, lecz nieco obok – por. (x+9, y-5). Spowodowane to jest rozmiarem wskaźnika myszy, który nieco zasłoniłby wypisywany tekst.

Jeżeli napiszemy polecenie bez nawiasów w pierwszym wierszu, wyświetli się coś innego, niż spodziewaliśmy się. Teoretycznie nawiasów tam nie trzeba, ale bez nich Baltie myli polecenia.

Sytuacja, gdy komputer traktuje polecenie inaczej, niż się spodziewamy występuje stosunkowo często, i to praktycznie bez względu na używany język programowania. Najczęściej problemy takie można rozwiązać odpowiednim użyciem nawiasów w wyrażeniach. Często wyjaśnia to także strukturę wyrażenia nawet samemu autorowi. Przeważnie później okazuje się, że zapomnieliśmy o niektórych regułach.

Współrzędne pobrane z pozycji Baltiego



Rysunek 12.17
Basen z wyspą

Jak już zauważyliśmy, do obiektów, na których współrzędnych możemy coś umieścić i których współrzędne można otrzymać, należy też Baltie i pole przed Baltiem. Gdy znamy te dwa pola, łatwo otrzymać współrzędne pola, które jest określoną liczbę pól przed Baltiem i potem umieścić przedmiot na tym polu.

Spróbujmy rozwiązać proste zadanie: wyobraźmy sobie, że Baltie ma w domu basen. Na środku tego basenu jest wyspa, na której Baltie chciałby zasadzić kwiaty (patrz rysunek 12.17 – wodę i wyspę znajdziemy w banku 1 po lewej stronie na dole). Baltie nie chce zamoczyć nóg. Nie byłby jednak prawdziwym czarodziejem gdyby nie potrafił czarować poprzez wodę. Nauczymy go, jak to zrobić.

Wyczarowanie kwiatów dwa pola przed Baltiem mogłoby nam jeszcze stwarzać problemy. Spróbujmy jednak napisać już swoje własne rozwiązanie. W międzyczasie podamy jeszcze kilka informacji o współrzędnych.

Potrzebujemy formuły, która ze współrzędnych Baltiego i współrzędnych pola przed nim (nic innego nie mamy) wyprowadzi współrzędne pola w odpowiedniej odległości przed Baltiem. Jeśli wymyślimy formułę, która bez względu na kierunek potrafi poprawnie obliczyć z tych wartości współrzędne odpowiedniego pola, wygraliśmy.

Jak wiadomo, współrzędne pola przed Baltiem różnią się od pola Baltiego o jedynkę na jednej współrzędnej, X lub Y. To, czy w grę wchodzi współrzędna X czy współrzędna Y, zależy od kierunku, w jakim Baltie jest obrócony. Od kierunku ustawienia zależy też, czy współrzędna jest większa czy mniejsza od współrzędnej pola przed Baltiem. Współrzędne wszystkich pół przed Baltiem różnią się od współrzędnych Baltiego według tej samej reguły, różnice dotyczą tylko odległości.

Innymi słowy: jeśli weźmiemy różnicę między współrzędną X Baltiego i współrzędną X pola przed nim, dla pola za tym polem będzie to różnica dwukrotna, dla następnego pola trzykrotna itd.

Jeśli więc Baltie będzie obrócony na wschód, współrzędna pola przed Baltiem będzie o jedynkę większa (różnica wyniesie +1), współrzędna następnego pola będzie większa o dwa (+2), itd.

Jeśli będzie obrócony na zachód, współrzędna X pola przed Baltiem będzie o jedynkę mniejsza (różnica wyniesie -1), współrzędne następnego pola mniejsza o dwa (-2), następnego o trzy (-3) itd.

Jeśli Baltie będzie obrócony na północ lub południe, współrzędne X wszystkich pół przed Baltiem będą zgodne ze współrzędną X Baltiego (różnica wyniesie zero). To nadal jest zgodne z naszą regułą „mnożącej się różnicy", bo 2×0=0.

Teraz już nie powinno być problemem napisanie formuły otrzymania współrzędnej pola znajdującego się*N* pól przed Baltiem. Jeżeli para liczb (X_B, Y_B) określaja współrzędne Baltiego, a para (X_P, Y_P) współrzędne pola przed Baltiem i para liczb (X_N, Y_N) współrzędne pola N pól przed Baltiem, to:

$$X_N = X_B + N * (X_P - X_B)$$

$$Y_N = Y_B + N * (Y_P - Y_B)$$

Dla Czytelników, którzy jeszcze nie są biegli we wzorach, pokażemy to wszystko na przykładzie. Załóżmy, że Baltie stoi obrócony na północ na współrzędnych

$$(X_B, Y_B) = (3, 6)$$

Współrzędne pola przed Baltiem opisuje para liczb

$$(X_{P_N}, Y_{P_N}) = (3, 5)$$

Jeśli stojąc na tej samej pozycji Baltie będzie obrócony na południe, wtedy współrzędne pola przed Baltiem to

$$(X_{Ps}, Y_{Ps}) = (3, 7).$$

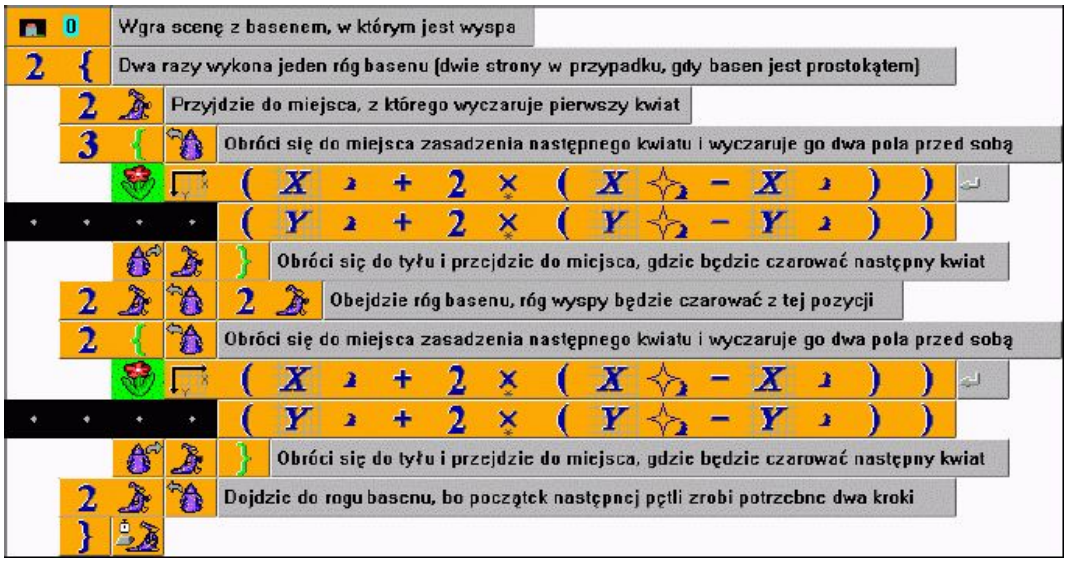
Kiedy w naszych formułach zastąpimy współrzędne obliczonymi wartościami, zobaczymy, że współrzędne pola odległego o 2 pola przed Baltiem będą przy obróceniu na północ wynosiły

$$(X_{2N}, Y_{2N}) = (3 – 2*(3-3) \ , \ 6 – 2 * (5 – 6)) = (3, 4)$$

a przy obróceniu na południe współrzędne wyniosą

$$(X_{2s}, Y_{2s}) = (3 – 2 * (3 – 3) \ , \ 6 – 2 * (7 – 6)) = (3, 8)$$

Spróbujmy jeszcze raz wszystko przeanalizować i napisać własną wersję programu. Gdy będzie gotowa, możemy swoje rozwiązanie porównać z rozwiązaniem z rysunku 12.18.



Rysunek 12.18
Baltie sadzi kwiaty na wyspie w basenie

12.6 Ustawianie Baltiego na współrzędnych, mierzenie czasu, liczba losowa

Współrzędne można nie tylko otrzymać od Baltiego, lecz też ustawiać jego pozycję. Wystarczy wprowadzić za element **Baltie** element **Współrzędne** i potem podać współrzędne, na które chcemy Baltiego przesunąć – np. program na rysunku 12.19 próbuje ciągle umieszczać Baltiego na współrzędnych określonych położeniem myszki.

Rysunek 12.19
Program umieszczający Baltiego na pozycji wskaźnika myszy

Gdy uruchomimy program z rysunku 12.19 zobaczymy, że trochę różni się od programów, które wyczarowują przedmioty. Przede wszystkim umieszcza Baltiego zawsze na współrzędnych pola, i to pomimo faktu, że w poleceniu są współrzędne piksela. W dodatku Baltie nie zostawia za sobą śladu w przeciwieństwie do przedmiotów.

Spróbujmy teraz oprogramować taką prostą grę: Baltie będzie w sposób przypadkowy (mówimy: losowo) zmieniać swoją pozycję. Zadaniem gracza będzie złapać go klikając na niego myszą. Gdy to się uda, gra będzie skończona.

Stoper

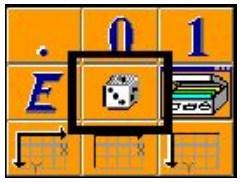
Podczas tworzenia tego programu nauczymy się jeszcze czegoś nowego. Pokażemy, jak można mierzyć czas trwania pewnego zdarzenia. Na pewno każdy z nas widział *stoper* i wie, jak go używać: pierwsze naciśnięcie przycisku uruchamia pomiar czasu, następne naciśnięcie zatrzyma pomiar i pozwala odczytać czas. Gdy naciśniemy przycisk szybko dwa razy stoper wyzeruje się, jeśli naciśniemy szybko trzy razy wyzeruje się i znów uruchomi pomiar.

Podobnie zachowuje się też stoper Baltiego. Do jego sterowania potrzebujemy dwóch elementów: elementu **Stoper** i elementu **Naciśnij przycisk stopera**, które znajdują się w panelu elementów nad elementami komentarzowymi – patrz rysunek 12.20.

Rysunek 12.20
Elementy stopera

Liczba losowa

Teraz pokażemy, jak do swoich programów dodać przypadkowość. Baltie ma wbudowany generator liczb losowych (przypadkowych), którego możemy użyć w swoich programach. Jeśli zawołamy element **Liczba losowa** (w panelu elementów znajdziemy go pod zerem – patrz rysunek 12.21), otrzymamy jako wartość przypadkową liczbę rzeczywistą wylosowaną pomiędzy zerem i jedynką – mówimy o *liczbie losowej*. Jeśli jednak za ten element wprowadzimy liczbę całkowitą otrzymamy liczbę całkowitą pomiędzy zerem a liczbą o jedynkę mniejszą, niż liczba, którą wprowadziliśmy za ten element. Podając np. 4 otrzymamy wylosowaną liczbę 0, 1, 2 lub 3. Program będzie zachowywać się tak, jakbyśmy podczas każdego przejścia programu przez element **Liczba losowa** wprowadzali w tym miejscu liczbę z określonego przedziału.



Rysunek 12.21
Liczba losowa

Jeśli będziemy chcieli zagrać w grę, do której trzeba kostki i nie będziemy mogli znaleźć kostki, doskonale zastąpi ją program z rysunku 12.22. Za każdym razem poczeka, aż ktoś naciśnie jakiś klawisz lub kliknie myszą, piknie i w środku ekranu pokaże się liczba losowa od jedynki do szóstki.

Rysunek 12.22
Automatyczna kostka

Wartość bezwzględna

W zaproponowanej wersji programu łapania Baltiego znajdziemy jeszcze jeden nieużywany do tej pory element: **Wartość absolutna** (w panelu elementów znajduje się w środku dolnego wiersza – patrz rysunek 12.23). Po wykładach z matematyki wiemy, co to jest *wartość bezwzględna* liczby: najkrócej mówiąc jest to wartość powstała poprzez odrzucenie znaku od liczby, którego wartość bezwzględną chcemy otrzymać. Odznacza się to kreskami pionowymi:

Rysunek 12.23
Element Wartość absolutna

$|3| = 3$ $\quad\quad$ $|-3| = 3$

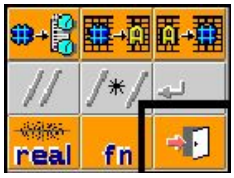
$|5| = 5$ $\quad\quad$ $|-5| = 5$

Wartość bezwzględną liczby oblicza w Baltie element **Wartość absolutna**. Oznaczenie abs pochodzi od angielskiego *absolute*. Element ten oczekuje za sobą liczby, której wartość bezwzględną chcemy otrzymać i podaje wartość bezwzględną tej liczby. Jeśli chcemy otrzymać wartość bezwzględną wyrażenia pamiętajmy o tym, że trzeba wyrażenie zamknąć w nawiasy tak, jak na rysunku 12.24.

Rysunek 12.24
Wyświetl odległość Baltiego od wskaźnika myszy

Przerwanie działania programu

Ostatnim nowym elementem, który znajdziemy w omawianym programie, jest element **Zakończ program**. Wprowadza się go do programu w miejsce, gdzie chcemy przerwać działanie programu. Użycie tego elementu jest jedynym sposobem, w jaki można opuścić pętlę nieskończoną. Odpowiedni element można znaleźć w prawym dolnym rogu panelu elementów (patrz rysunek 12.25).



Rysunek 12.25
Element Zakończ program

Program Złap Baltiego

Teraz już wiemy wszystko, czego potrzebujemy i możemy zacząć budowę programu. Warto spróbować stworzyć cały program *Złap Baltiego* samodzielnie. Dla mniej odważnych program jest na rysunku 12.26.

Program najpierw uruchomi stoper (ten Baltiego uruchamia się trzykrotnym naciśnięciem przycisku stopera). Potem

rozpocznie pętlę nieskończoną. W niej najpierw przesunie Baltiego na losową pozycję na dworku i poczeka chwilę, by gracz mógł go złapać. Program czeka co najwyżej pół sekundy, lecz łatwo można zmienić czas czekania (500 to pół sekundy, 1000 to sekunda itd.). Po odczekaniu wyświetli się w lewym górnym rogu ekranu czas ze stopera.

Rysunek 12.26
Program Złap Baltiego

Teraz przechodzimy do polecenia kluczowego dla całego programu. Sprawdzimy, czy współrzędne pola myszy w momencie kliknięcia i współrzędne pola Baltiego są różne. Jeśli będą takie same, niczego od jedynki nie odejmiemy i jeden raz wykona się treść pętli, w której Baltie piknie, poczeka aż przeczytamy czas ze stopera i po następnym naciśnięciu klawisza czy kliknięciu myszy zakończy program (wyjdzie drzwiami).

Jeśli współrzędne myszy będą się różnić od współrzędnych Baltiego różnica tych współrzędnych będzie niezerowa. Jeśli więc współrzędne różnią się, wtedy bez względu na to, która z nich jest większa, zawsze od jedynki odejmiemy pewną wartość dodatnią. Wtedy pętla zostanie wykonana mniej niż raz czyli zero razy. Nie dojdzie więc do zakończenia programu i nieskończona pętla musi zostać wykonana ponownie. Gdy na koniec uda nam się kliknąć w Baltiego, współrzędne Baltiego i wskaźnika myszy będą zgodne, od jedynki nic nie odejmiemy, program wejdzie do treści pętli na samym końcu. Tam piknie, poczeka, nim przeczytamy zmierzony czas i gdy naciśniemy jakiś klawisz lub przycisk myszy zakończy działanie.

12.7 Czego nauczyliśmy się

Dowiedzieliśmy się, jakich współrzędnych można używać w Baltiem i pokazaliśmy elementy, których będziemy do tego potrzebować. Nauczyliśmy się wyczarować określony przedmiot na określonych współrzędnych. Pokazaliśmy, jak wyczarować przedmiot na współrzędnych myszy i skonstruowaliśmy dwie wersje prostego programu do grania w kółko i krzyżyk na komputerze.

W następnej części przeszliśmy od czarowania do wyświetlania i pokazaliśmy, jak na określonych współrzędnych wyświetlić napis. Potem nauczyliśmy się nie tylko ustawiać współrzędne, ale też je pobierać. Pokazaliśmy, jak można z takich umiejętności skorzystać w swoich programach.

Na końcu nauczyliśmy się mierzyć czas za pomocą stopera i korzystać z wbudowanego generatora liczb losowych. Za pomocą tych wszystkich umiejętności zaprogramowaliśmy prostą grę, w której gracz próbuje jak najszybciej złapać Baltiego losowo przemieszczającego się po dworku.

13. Dzielimy pracę między pomocników

Czego nauczymy się w tym rozdziale

W wielu naszych programów było do tej pory kilka podobnych, jeśli wręcz identycznych części. Pokażemy teraz jeden z największych wynalazków w historii programowania: możliwość skonstruowania pomocnika, który będzie zajmować się jednym konkretnym zadaniem. Nauczymy się używać pomocników i budować z ich pomocą nawet skomplikowane programy jako programy stosunkowo czytelne i łatwe do zmiany. Przypomnimy reguły tworzenia złożonych programów i zbudujemy kilka takich programów.

13.1 Do czego potrzebni są pomocnicy

Programiści od samego początku walczyli z tym, że w różnych miejscach programu potrzebowali wykonać prawie te same operacje. Kopiowanie odpowiedniego kawałka kodu programu było uciążliwe. Potem jakiś spryciarz wymyślił jeden z największych wynalazków programistycznych: **procedury** (nazywane też **podprogramami**). Ponieważ zwykłemu człowiekowi słowo procedura przeważnie nic nie mówi używamy w Baltie zamiast słowa**procedura** terminu **pomocnik**.

Pomocnika można wyobrazić sobie jako kolegę, któremu przydzielimy do wykonywania pewną konkretną czynność. Musimy go jednak najpierw nauczyć tej czynności tak jak uczymy Baltiego. Dlatego też pomocnik posiada część programu, gdzie opisano jak powinien działać. Kiedykolwiek będzie potrzebna nam opisana czynność po prostu zawołamy pomocnika i on wykona zadanie.

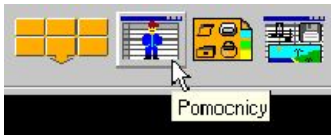
Jak już powiedzieliśmy, z pomocników korzystamy wtedy, gdy potrzebujemy w kilku miejscach programu wykonać tę samą czynność. Nie jest wygodnie programować czynność tyle razy, ile razy chcemy ją wykonać. Znacznie korzystniej jest zdefiniować pomocnika, „nauczyć" go wykonać tę akcję i w odpowiednich miejscach zawołać go, by wykonał potrzebne czynności. Znacznie zwiększa to czytelność programu.

Jeśli potrzebujemy później zmienić definicję czynności wykonywanej przez pomocnika, czy dlatego, że zrobiliśmy błąd w programie, czy też tylko chcemy inaczej wykonać daną akcję, wystarczy zmienić definicję pomocnika. Nie trzeba wyszukiwać wszystkich miejsca w programie, gdzie opisaliśmy zmienianą czynność.

Drugim powodem, który powinien zachęcić nas do używania pomocników, jest większa czytelność programu. Wygodnie jest podzielić zadanie na szereg zadań prostszych i dla każdego z nich zdefiniować jego osobnego pomocnika. Zamiast jednego długiego programu otrzymamy w ten sposób kilka krótszych programów, w których łatwiej się zorientować. Zrobimy w nich mniej błędów. Poza tym błędy, które niechcąco zrobimy, łatwiej znajdziemy i usuniemy. Pomocnik doskonale sprawdza się przy programowaniu *metodą zstępującą.*

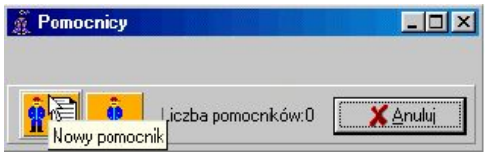
13.2 Definicja pomocnika

Część programu, w której wyjaśniamy pomocnikowi, co i jak powinien robić, nazywamy **definicją pomocnika**. Definicja pomocnika musi zaczynać się elementem **Nowy pomocnik**, który można wprowadzić do programu z **Tablicy pomocników**. Otworzymy ją naciśnięciem klawisza **Pomocnicy** w panelu narzędzi po prawej stronie obok ikony panelu elementów (patrz rysunek 13.1).



Rysunek 13.1
Otworzenie tablicy pomocników

Przy pierwszym otwarciu tablica będzie pusta. Przesuwamy się w niej na przycisk **Nowy pomocnik** (patrz rysunek 13.2) przy jego dolnej krawędzi i naciskamy go. Tablica zamknie się i na końcu programu pojawi się nowy wiersz z elementem **Nowy pomocnik**.



Rysunek 13.2
Tablica pomocników – element Nowy pomocnik

Baltie może mieć bardzo wielu pomocników – ich liczba jest ograniczona tylko wielkością pamięci komputera. By Baltie (i programista) mógł się zorientować w tym tłumie, każdy pomocnik musi mieć swoje odznaczenie. Pomocnika oznaczamy wprowadzając za elementem **Nowy pomocnik**:

- numer pomocnika,
- przedmiot oznaczający pomocnika,
- nazwę pomocnika (tzn. ciąg znaków).

Rysunek 13.3
Pomocnik oznaczony numerem

Nie zalecamy oznaczania pomocników numerami (patrz rysunek 13.3). Łatwo zapomnieć, który pomocnik czym się zajmuje. Numeracja przyda się tylko w niektórych specjalnych przypadkach – na końcu tego rozdziału pokażemy jeden z nich.

Oznaczanie pomocników przedmiotami (patrz rysunek 13.4) jest lepsze zwłaszcza jeśli wymyślimy jakąś zasadę, konwencję, której będziemy przestrzegać (np. część inicjalizacji programu będziemy zawsze oznaczać słoneczkiem itp.). W prostszych programach wystarczy taka metoda oznaczania, bo jest z nim najmniej pracy i w wielu przypadkach przedmiot wywołuje skojarzenie z zadaniem pomocnika. W bardziej skomplikowanych programach zawierających wielu pomocników zaczną się jednak poszczególne przedmioty mylić.

Rysunek 13.4
Pomocnik oznaczony przedmiotem

Dla długich i skomplikowanych programów polecamy jako najwygodniejszy sposób oznaczać pomocniki napisami, które będą określać, czym ten pomocnik zajmuje się (patrz rysunek 13.5). Patrząc na nazwę pomocnika łatwiej zorientować się w konstrukcji programu.

Rysunek 13.5
Pomocnik oznaczony napisem

Bez względu na to, jaki sposób oznaczania pomocników wybierzemy, dobrze jest za oznaczeniem pomocnika wprowadzić komentarz, który opisuje zadanie pomocnika. (patrz rysunki 13.3 i 13.4).

Element **Nowy pomocnik** wspólnie z oznaczeniem pomocnika i komentarzem będziemy nazywać **nagłówkiem pomocnika**. Za nagłówkiem wprowadzamy kolejne polecenia, które pomocnik powinien wykonać, by zrealizować zadanie – innymi słowy tu wprowadzamy podprogram pomocnika. Te polecenia (podprogram) tworzy **treść pomocnika**.

Polecenia można wprowadzać tuż za nazwą pomocnika, ale dla lepszej czytelności lepiej zacząć dopiero w następnym wierszu. Łatwiej będzie w przyszłości zaznaczyć całego pomocnika jako blok gdy zrobimy wcięcie wszystkich poleceń o jedno pole (patrz rysunek 13.6).

Treść pomocnika (i cała definicja pomocnika) kończy się albo końcem programu, albo początkiem definicji następnego pomocnika. Część programu od początku programu do definicji pierwszego pomocnika będziemy nazywać **program główny**. We współczesnych językach programowania nawet program główny może być zdefiniowany jako pomocnik.

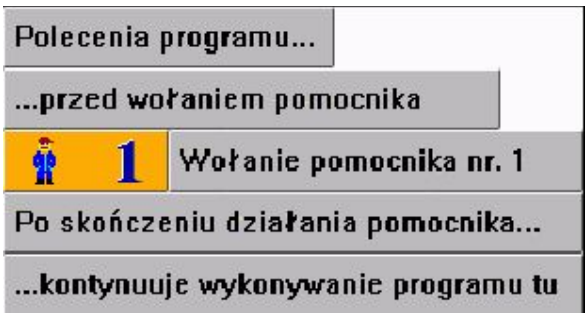
Rysunek 13.6
Wygląd typowego pomocnika

Dla lepszej czytelności warto wstawiać pomiędzy definicje kolejnych pomocników jeden lub dwa puste wiersze.

Ponieważ w naszych programach będzie wielu i to długich pomocników, będziemy oznaczać koniec dłuższych pomocników specjalnym wierszem zbudowanym z elementu **Komentarz wierszowy** i z elementu **Zawołaj pomocnika** i nagłówka pomocnika bez elementu **Nowy pomocnik** (tego nie pozwoli Baltie wprowadzić w tym miejscu) – jak na rysunku 13.6.

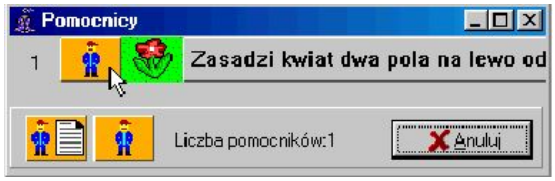
Pomocnika zawołamy wpisując w programie element **Zawołaj pomocnika** z oznaczeniem właściwego pomocnika w miejscu, gdzie pomocnik ma wykonać swoją pracę. Gdy podczas wykonywania programu Baltie napotka polecenie wywołania pomocnika, przekaże wszystkie swoje kompetencje temu pomocnikowi. Po zakończeniu pracy pomocnika Baltie kontynuuje wykonywanie poleceń, które znajdują się za poleceniem wywołania (na rysunku 13.7 pokazano, jak to działa).

Wywołanie pomocnika najlepiej wprowadzić do programu otwierając tablicę pomocników i wskazując w niej wiersz



Rysunek 13.7
Działanie programu używającego pomocnika

z nagłówkiem odpowiedniego pomocnika. Pojawia się tam natychmiast, gdy wprowadzimy element **Nowy pomocnik** do programu. Po wskazaniu na element **Nowy pomocnik**, którym zaczyna się nagłówek, element ten zmieni się na element **Zawołaj pomocnika** (patrz rysunek 13.8).



Rysunek 13.8
Tablica z nagłówkiem definiowanego pomocnika

Kliknięciem chwytamy element i jednocześnie zamykamy tablicę pomocników. Potem już wystarczy znaleźć miejsce w programie, w którym chcemy tego pomocnika wywołać i wprowadzić trzymany element na to miejsce. Na to miejsce Baltie wstawi nie tylko element **Zawołaj pomocnika**, lecz także oznaczenie odpowiedniego pomocnika. Wprowadzone zostanie całe polecenie wywołania pomocnika.

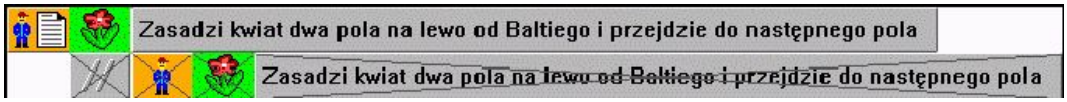
Pomocników można wywołać nie tylko z programu głównego. Mogą zostać zawołani też przez innych pomocników. Pomocnik może nawet wywołać samego siebie (mówimy wtedy o **rekurencji**).

13.3 Pomocnik sadzi kwiaty

Pokażemy teraz to, o czym mówiliśmy na jakimś prostym przykładzie. Dobrym miejscem na wprowadzenie pomocnika jest program do sadzenia kwiatów na wyspie w basenie, którym skończyliśmy poprzedni rozdział. Tam w dwóch miejscach użyliśmy dokładnie tego samego fragmentu programu.

Wprowadźmy więc pomocnika, który potrafi zasadzić kwiatek dwa pola przed Baltiem. Postępujemy według następnego przepisu:

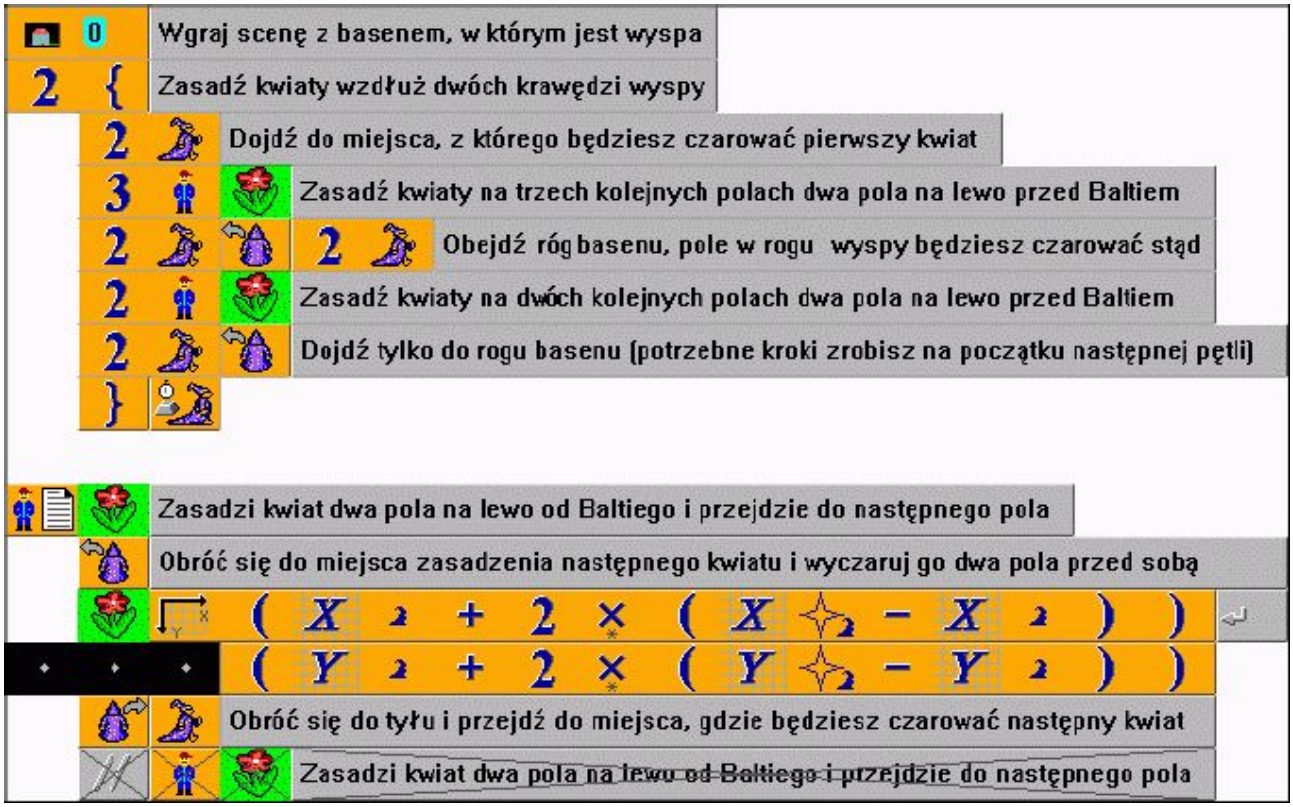
1. Otwieramy program *Kwiaty w basenie* z końca poprzedniego rozdziału. Na koniec programu wprowadzamy dwa puste wiersze, by nasz przyszły pomocnik został wyraźnie oddzielony od programu głównego.
2. Naciśnięciem przycisku **Pomocnicy** otwieramy tablicę pomocników i naciskamy przycisk **Nowy pomocnik.** Baltie umieści odpowiedni element na końcu programu.
3. Za właśnie wprowadzony element dodaj oznaczenie – w naszym programie można do oznaczenia użyć np. kwiat, który Baltie czaruje w klombie w basenie (bank 1, przedmiot 1139).
4. Za oznaczeniem pomocnika dodajemy komentarz, w którym opiszemy działanie definiowanego pomocnika. W naszym przypadku możemy tu napisać „Zasadzi kwiatek dwa pola na prawo obok Baltiego i przejdzie na następne pole".
5. Ten punkt oraz następny posłuży tylko do zwiększenia czytelności definicji pomocnika. Można to pominąć (choć nie zalecamy) kontynuować od razu w punkcie **7**. Naciśnięciem przycisku **Pomocnicy** ponownie otworzymy tablicę pomocników. Tym razem będzie już w niej nagłówek definiowanego pomocnika. Gdy wskażemy na pierwszy element nagłówka (**Nowy pomocnik**) zmieni się on w element **Zawołaj pomocnika** (patrz rysunek 13.8). Kliknięciem chwytamy go wspólnie z oznaczeniem pomocnika. To od razu zamknie tablicę. Chwycony element wprowadzamy do programu w wierszu za nagłówkiem pomocnika. Posłuży nam do wstawienia komentarza.
6. Przed właśnie wprowadzonym elementem dodajemy wcięcie (odstęp) i element **Komentarz wierszowy**. Tak wprowadziliśmy dodatkowy wiersz, który będzie optycznie zamykać całą definicję pomocnika i pozwoli w przyszłości łatwiej zorientować się w programie, nawet gdy będzie on bardzo rozbudowany. Oczekiwany efekt można zobaczyć na rysunku 13.9.



Rysunek 13.9
Pusty pomocnik – wygląd początkowy przyszłego dłuższego pomocnika

7. Za nagłówkiem pomocnika, ale przed wierszem końcowym stworzonym w punktach 5 i 6, wprowadzamy definicję działań pomocnika. W naszym przypadku możemy skopiować tu treść pętli, w której Baltie sadzi kwiaty.
8. Teraz, gdy już jest gotowa definicja pomocnika, możemy wprowadzać do programu wywołanie tego pomocnika. Jeśli chcemy w programie zawołać odpowiedniego pomocnika, by wykonał to, do czego został stworzony, wprowadzamy do programu jego wywołanie według kroków opisanych w punkcie **5**. Wygląd całego programu po

wszystkich potrzebnych zmianach można zobaczyć na rysunku 13.10.



Rysunek 13.10
Ostateczny wygląd programu sadzenia kwiatów na wyspie w basenie po wprowadzeniu pomocników

Teraz popatrzmy, jak działa nasz nowy program.

1. Najpierw otworzy scenę i uruchomi pętlę, która zostanie wykonana dwa razy.
2. W treści pętli zewnętrznej Baltie zrobi dwa kroki, by dojść do pozycji początkowej, z której zacznie sadzić kwiaty.
3. Trzykrotnie zawoła pomocnika kwiatowego, który zawsze zasadzi kwiat na odpowiednim miejscu i przesunie Baltiego o jedno pole dalej.
4. Ostatnie pole w wierszu zostanie ominięte i Baltie obejdzie róg.
5. Dwa razy zawoła pomocnika kwiatowego, czyli zasadzi kwiaty wzdłuż krótszej strony klombu.
6. Obejdzie róg basenu. Przechodząc przez pętlę po raz pierwszy, wróci do punktu 2 i ponownie wykona treść pętli.

13.4 Tworzymy program z wieloma pomocnikami

W poprzednim programie był jedyny pomocnik. Nasze programy będą jednak przeważnie używać wielu pomocników. Zbudujemy taki program i na jego przykładzie pokażemy, jak można pracować z pomocnikami i jak Baltie ułatwia nam to zadanie.

Z pierwszej części na pewno pamiętamy, jak uczyliśmy Baltiego budować wioskę w lesie. Spróbujmy teraz zmodyfikować ten program tak, by zamiast zestawu takiego samego rodzaju domków Baltie losowo budował różne rodzaje domków. Wioska może wyglądać np. tak jak na rysunku 13.11 (las na razie pominiemy. Na tym przykładzie pokażemy, jak powinniśmy tworzyć złożone programy.

Rysunek 13.11
Wioska z losowo budowanych domków

Umówmy się, że od tej chwili w naszych programach program główny będzie tworzony jedynym poleceniem –

wywołaniem pomocnika „głównego”. Tak zbliżymy się do zwyczajów współczesnych języków programowania, w których program główny jest definiowany jako „główny” pomocnik. Z góry przepraszamy, że w krótszych przykładach złamiemy tę regułę dla zaoszczędzenia miejsca. Przy dłuższych programach będziemy jej ściśle przestrzegać.

Dekompozycja problemu i projektowanie metodą zstępującą

Na pewno pamiętamy, że każde skomplikowane zadanie powinniśmy najpierw rozłożyć (**dekomponować**) na szereg zadań prostszych i potem rozwiązywać prostsze zadania. Jeśli niektóre z tych zadań będą wciąż za bardzo skomplikowane, dzielimy je ponownie. Metodę tę nazwaliśmy *programowaniem zstępującym* lub metodą *dziel i rządź*.

Rysunek 13.12
Koncepcja budowania losowej wioski

Teraz, gdy już mamy pomocników, dekompozycja będzie najczęściej polegać na podzieleniu zadań między poszczególnych pomocników. Nie wystarczy jednak tylko zdefiniować, co który pomocnik ma robić. Równie ważna jest dokładna **specyfikacja** czyli określenie, jak pomocnik będzie się komunikować z programem głównym. Powinniśmy zawsze wiedzieć, w jakim stanie pomocnik będzie przejmować Baltiego i jego dworek oraz w jakim stanie będzie je oddawać po wykonaniu zadania. Mówimy, że powinniśmy zdefiniować **interfejs** między pomocnikiem i programem go wywołującym.

Wszystko pokażemy na przykładzie. Na rysunku 13.12 można zobaczyć wstępną postać programu. Jak widać, program główny składa się z jednego polecenia, którym jest wywołanie pomocnika **Buduj wioski.** Ten pomocnik zajmuje się wszystkim.

Pomocnika przygotujemy tak, aby można było ciągle budować nowe i nowe wioski. Po naciśnięciu klawisza lub przycisku myszy pomocnik zbuduje w miejscu istniejącej nową wioskę.

Początkowe ustawienie na placu budowy (parceli), tak jak i końcowy powrót na pozycję wyjściową oprogramujemy raczej ręcznie i nie będziemy tworzyć specjalnego pomocnika. Podobnie poradzimy sobie z przejściem na początek następnego wiersza w pętli wewnętrznej. Zbudowanie domku można potraktować jako zadanie bardziej złożone, więc jego realizację przekażemy pomocnikowi.

Zanim zaczniemy zajmować się pomocnikiem, wyjaśnijmy, jaki będzie interfejs między programem wywołującym a wywoływanym pomocnikiem, czyli czego może oczekiwać pomocnik od programu go wywołującego. Ważne jest także, czego program może oczekiwać od pomocnika, który wykonał swoje zadanie.

Na przykład umówmy się, że pomocnik będzie ustawiać Baltiego przed domem, który ma zbudować. Po zakończeniu zadania pozostawi Baltiego za zbudowanym domem.

Po tych wszystkich przygotowaniach możemy zakończyć konstruowanie pomocnika **Buduj wioski.** Wszystkie działania są stosunkowo proste, więc nie powinno być z nimi problemów. Możliwą postać pomocnika **Buduj wioski** mamy na rysunku 13.13.

Pomocnik **Buduj wioski** jest już przygotowany, lecz nie można go jeszcze wypróbować i przetestować, bo nie mamy napisanego pomocnika **Zbuduj domek.** Żeby nie odkładać testu programu

Rysunek 13.13
Buduj wioski – postać końcowa

wywołującego zaprogramujemy tymczasowe rozwiązanie, które powinno wystarczyć. Najprostszym rozwiązaniem byłoby zrobić trzy kroki, lecz potem trudno sprawdzałoby się przejście do następnego wiersza.

Zbudujemy więc pomocnika według rysunku 13.14 tak, by po prostu oznaczył fragmentem ściany miejsce, gdzie będzie stać przyszły domek. Teraz już nic nie przeszkadza nam w uruchomieniu programu.

Rysunek 13.14
Zbuduj domek – tymczasowy

Gdy program wywołujący pomocnika zostanie sprawdzony i usuniemy ewentualne błędy, możemy rozpocząć tworzenie kolejnych wywoływanych pomocników. W naszym przypadku weźmiemy się za tworzenie pomocnika **Zbuduj domek.** Wiemy, że jego zadaniem jest zbudować domek, którego wygląd powinien być wybierany w sposób przypadkowy. Nawet przypadek ma jednak swoje ograniczenia. Na pewno nie będziemy budować domku z dachem umieszczonym pod fundamentami. Budując ściany możemy pozwolić pomocnikowi dowolnie zadecydować, czy na określonym miejscu zamontuje pełną ścianę, okno czy drzwi. Podobnie budując dach będzie mógł wybrać, czy zbuduje dach frontem do nas czy domek będzie odwrócony bokiem.

Rysunek 13.15
Buduj domek – koncepcja

Narysujmy teraz wstępną wersję pomocnika **Zbuduj domek**. W programie pokazanym na rysunku 13.15 zadecydowaliśmy, że w ścianie wybierzemy każdy element (panel) z osobna a dach wybierzemy jako całość. Jeśli zechcemy zrzucić zbudowanie pojedynczego panelu na pomocnika, zdefiniujmy ten projekt pomocnika **Panel** i jego interfejs: pomocnik tylko zbuduje przed Baltiem odpowiedni panel. Przed i po wykonaniu pracy pomocnika **Panel** Baltie będzie stał w tym samym miejscu.

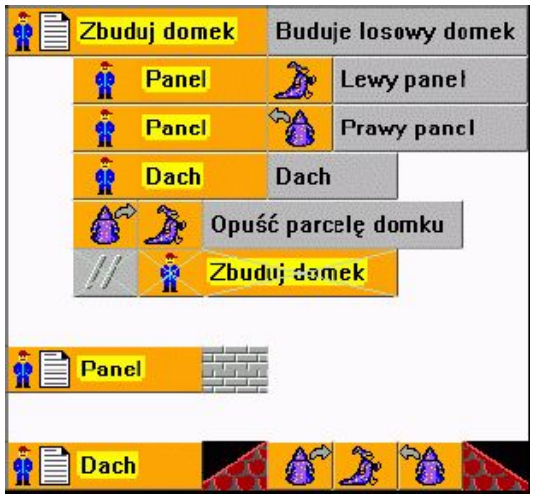
Jeśli zechcemy zostawić pomocnikowi także budowanie dachu, powinniśmy ustalić też jego interfejs. Powiedzmy, że pomocnik otrzyma Baltiego stojącego pod lewą połową dachu, obróconego do góry (na północ), zaś pozostawi go pod drugą połową dachu znów patrzącego na północ.

Tak zdefiniowane interfejs już wystarczająco dokładnie określa, jak Baltie powinien opuścić parcelę budowanego domku. Bo to zadanie jest stosunkowo proste i najlepiej nie zostawiać go wewnątrz żadnego pomocnika lecz zaprogramować osobno.

Teraz gdy mamy już pomocnika zaprojektowanego musimy sprawdzić jego działanie.

Zamiast pomocników dla ustawianie paneli zdefiniujmy ponownie tymczasowych zastępców pomocników. Możliwy wygląd pomocnika **Zbuduj domek** i wygląd zastępców budujących panele i dach można zobaczyć na rysunku 13.16.

Jeśli teraz uruchomimy program, Baltie zacznie budować domki. Wszystko działa, wystarczy tylko zastanowić się, jak zmieniać losowo wygląd domków.



Rysunek 13.16
Zbuduj domek – projekt

Losowo wywoływani pomocnicy

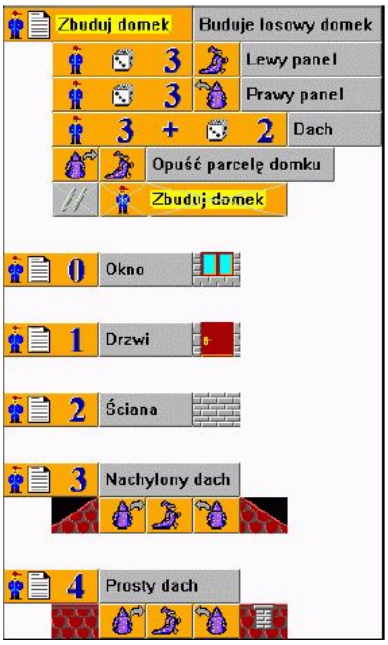
Obecna wersja pomocnika **Zbuduj domek** zakłada, że losowaniem zajmą się jego pomocnicy **Panel** i **Dach,** których wywołuje. Byłoby to możliwe, lecz jeszcze nie mamy wystarczającej wiedzy, by ten sposób zaprogramować.

Możemy jednak podejść do problemu z drugiej strony i skorzystać z faktu, że Baltie pozwala oznaczać pomocników numerami. Jeśli utworzymy osobnego pomocnika dla każdego rodzaju panelu i dla każdego rodzaju dachu, a potem wprowadzimy wewnątrz pomocnika **Zbuduj domek** losowanie numeru pomocnika, który zbuduje odpowiednią część domku, mamy zadanie rozwiązane.

Do zbudowania paneli utworzymy pomocników o numerach **0, 1** i **2**. Będą budować (wyczarować) zwykłą ścianę, ścianę z drzwiami lub ścianę z oknem – wystarczy losowo wybrać numer od zera do dwóch wywołać odpowiedniego pomocnika.

Podobnie możemy zrobić z dachem. Jeśli oznaczymy pomocników budujących dach numerami **3** i **4,** wystarczy losowo wybrać jeden z tych numerów. Jak wiemy, element **Liczba losowa** daje nam liczby od zera do liczby o jeden mniejszej, niż podana liczba. Nikt nam jednak nie broni dodać liczbę do otrzymanego numeru i wybierać pomocnika według uzyskanego wyniku.

Końcowa wersja naszego programu jest pokazana na rysunku 13.17. Możemy jeszcze rozszerzyć jej losowy charakter dodając następne typy paneli i dachów.



Rysunek 13.17
Zbuduj domek i jego pomocnicy – końcowy wygląd

13.5 Przydatne funkcje tablicy pomocników

Autorzy Baltiego wiedzieli, że doświadczeni programiści piszą długie programy z szeregiem najróżniejszych pomocników i dlatego wbudowali do tablicy pomocników kilka przydatnych funkcji, które ułatwią nam pracę.

Do tej pory z tablicy pomocników braliśmy pierwszy element **Nowy pomocnik** i dalej już jej nie używaliśmy (pierwsze elementy dla następnych pomocników mogliśmy potem kopiować z innego miejsca programu). Teraz pokażemy, do czego jeszcze można używać tablicy pomocników.

Tablica pomocników otwiera się w oknie, które można nie tylko przesuwać, lecz też powiększać i zmniejszać. Możemy więc jej wielkość ustalić tak, by widzieć wszystkich swoich pomocników. Każdy pomocnik odnotowany jest w jednym wierszu tablicy podzielonym na trzy części, z których każda pełni ważną funkcję.

Pierwszy element to liczba określająca pozycję pomocnika w programie. Po wskazaniu na tę liczbę myszą zostanie ona wyświetlona jako przycisk. Jeśli chwycimy go myszą, zmieni się wygląd wskaźnika myszy na rękę z linią poziomą powyżej. Linia ta określa miejsce, w które trzymana definicja pomocnika zostanie przemieszczona, gdy zwolnimy przycisk. Tak więc możemy łatwo zmieniać pozycje definicji poszczególnych pomocników w programie.

Działanie drugiej kolumny z elementami **Nowy pomocnik**, już opisywaliśmy. Element, na który wskażemy myszą, zmieni się w element **Zawołaj pomocnika**. Jeśli chwycimy go, tablica pomocników zostanie zamknięta i możemy wprowadzić wywołanie pomocnika do programu.

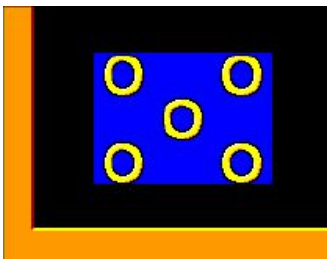
Ważną funkcję pełni też ostatnia część. Gdy na niej klikniemy, tablica pomocników zostanie zamknięta i w obszarze roboczym program przesunie się do definicji odpowiedniego pomocnika. Tak możemy łatwo i szybko poruszać się nawet po bardzo długich programach.

Rysunek 13.18
Tablica pomocników

Rysunek 13.19
Przycisk Wstecz

Po przesunięciu pojawi się w panelu narzędzi nowy symbol (po lewej stronie od przycisku otworzenia panelu elementów) – patrz rysunek 13.19. Kliknięciem na nim wrócimy na to poprzednie miejsce w programie.

13.6 Zadanie samodzielne

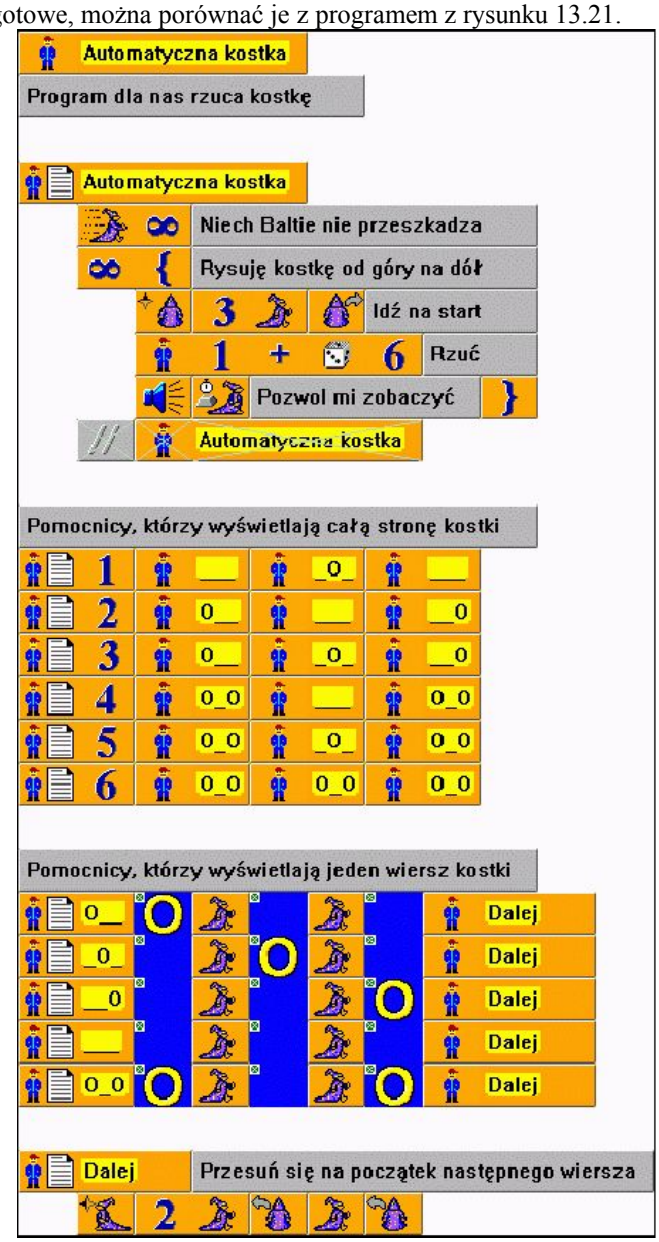


Rysunek 13.20
Proponowany wygląd kostki

Spróbujmy teraz samodzielnie rozwiązać jedno większe zadanie. Na końcu rozdziału o współrzędnych stworzyliśmy prosty program, którego można było użyć zamiast klasycznej kostki do gry. Spróbujmy teraz przygotować lepszą wersję programu do rzucania kostką. Przygotujmy program, który po każdym kliknięciu myszą lub naciśnięciu klawisza wygeneruje w lewym dolnym rogu dworku Baltiego rysunek kostki, podobnie jak to pokazano na rysunku 13.20.

Spróbujmy zbudować program samodzielnie. Pomoże nam poprzedni program rzucania kostką i program losowego czarowania wioski, który stworzyliśmy przed chwilą. Spróbujemy przestrzegać wszystkich zasad poprawnego projektowania złożonego programu, o których mówiliśmy i które pokazaliśmy podczas projektowania programu budowania wioski.

Gdy rozwiązanie będzie gotowe, można porównać je z programem z rysunku 13.21.



Rysunek 13.21
Kostka automatyczna – możliwy wygląd programu

Omówimy konstrukcje przedstawionego rozwiązania. Wszystkie programy tworzymy tak, by zawierały jedno jedyne polecenie: wywołanie głównego pomocnika, który w naszym przypadku nazywa się **Kostka automatyczna.** Ten pomocnik najpierw maksymalnie przyspieszy Baltiego i potem w kółko generuje poszczególne rzuty kostką. Za każdym razem najpierw przyprowadzi Baltiego do miejsca, gdzie ma wyczarować kostkę. Potem wygeneruje liczbę losową od 0 do 5, z której po dodaniu jedynki otrzyma numer od 1 do 6. Ten numer jest też numerem pomocnika, któremu rozkaże wyświetlić kostkę. Potem, gdy pomocnik wyświetli kostkę, Baltie piknie i poczeka, aż użytkownik zobaczy wynik i naciśnięciem klawisza lub przycisku myszy poprosi o następny rzut kostką.

Pomocnicy wyświetlający kostkę mają numery odpowiadające liczbie kropek na kostce. Nie wyświetlają ich jednak bezpośrednio. Pomocnik dzieli kostkę na poszczególne wiersze. Do wyświetlenia wiersza zawoła pomocnika, którego nazwa odpowiada postaci tworzonego wiersza kostki. Pomocnik butujący kostkę oczekuje, że każdy z tych pomocników wyświetli odpowiedni wiersz i przesunie Baltiego do pozycji, w której można wyświetlać następny wiersz.

Każdy z pomocników wyświetlenia wiersza zmusi Baltiego do wyczarowania przedmiotów, które zawierają kropkę lub pustą przestrzeń. W naszym przykładzie użyto do tego przedmiotów z banku 7: litery **O** (przedmiot 7090) i jednego z pustych przedmiotów (np. przedmiot 7142). Jak powiedzieliśmy w poprzednim akapicie, po wyświetleniu wiersza pomocnik powinien jeszcze przesunąć Baltiego na pozycję wyjściową do wyświetlania następnego wiersza. Tym jednak nie zajmują się pomocnicy samodzielnie lecz zawołają pomocnika **Dalej,** który zrobi wszystko, co trzeba.

13.7 Czego nauczyliśmy się

W tym rozdziale dowiedzieliśmy się, co to są pomocnicy i gdzie mogą nam się przydać. Nauczyliśmy się definiować pomocników i wywoływać ich później w programie. Opowiadaliśmy o niektórych zasadach wyglądu pomocników, których przestrzeganie znacznie poprawi czytelność naszych programów.

W następnej części powróciliśmy do sprawy podziału skomplikowanych problemów na szereg problemów prostszych i pokazaliśmy, jak można poprawić czytelność programu używając pomocników. Powiedzieliśmy też o ciekawych właściwościach tablicy pomocników, które pomogą nam w pracy. Na końcu samodzielnie przygotowaliśmy dość złożony i efektowny program.

14. Zaczynamy pamiętać – zmienne

Czego nauczymy się w tym rozdziale

W tym rozdziale pokażemy, jak programy mogą zapamiętać najróżniejsze dane, które wprowadzimy lub które wyliczymy w czasie działania programu. Wyjaśnimy, jaka jest przewaga stałych nad literałami. Zapoznamy się ze zmiennymi, których zawartość może zmieniać się w trakcie działania programu. Zapoznamy się też ze specjalnymi, prywatnymi zmiennymi pomocników i nauczymy się korzystać z ich przydatnych właściwości.

Jak dotąd nasze programy miały określone wszystkie wartości już na etapie tworzenia. Dokładnie wiedzieliśmy, ile kroków musi zrobić Baltie, by dojść do domku, które dwie liczby chcemy dodać, którego przedmiotu użyjemy do wyczarowania kwiatu.

Życie jest jednak ciągłą zmianą. Zmienia się pogoda, zmienia się nasz wiek, zmienia się wygląd ulicy, przez którą przechodzimy co dzień. Teraz poznamy środki, które pozwolą nam wprowadzić do naszych programów reakcje na niektóre zmiany.

14.1 Stałe

Do tej pory wszystkie potrzebne wartości wprowadzaliśmy do naszych programów używając literałów. Literały mają jednak pewną wadę: każdy z nich jest osobnym elementem. Jeśli chcemy któraś wartość zmienić, musimy szukać każdego wystąpienia literału w programie.

Wyobraźmy sobie, że przygotowujemy wielki program, który będzie regulować program dnia. Program ten będzie wiedział, że w dzień roboczy trzeba wstać o godzinie 7^{00} a w sobotę i w niedzielę o godzinie 8^{00}. Gdy zmienimy pracę trzeba będzie wstawać o 6-tej. Musimy zatem przejrzeć cały program i zmienić czas wstawania z 7^{00} na 6^{00}. Trzeba będzie przy tym uważać, by nie pomylić siódemki, która określa godzinę wstawania z siódemką, określającą niedzielę (7-my dzień w tygodniu).

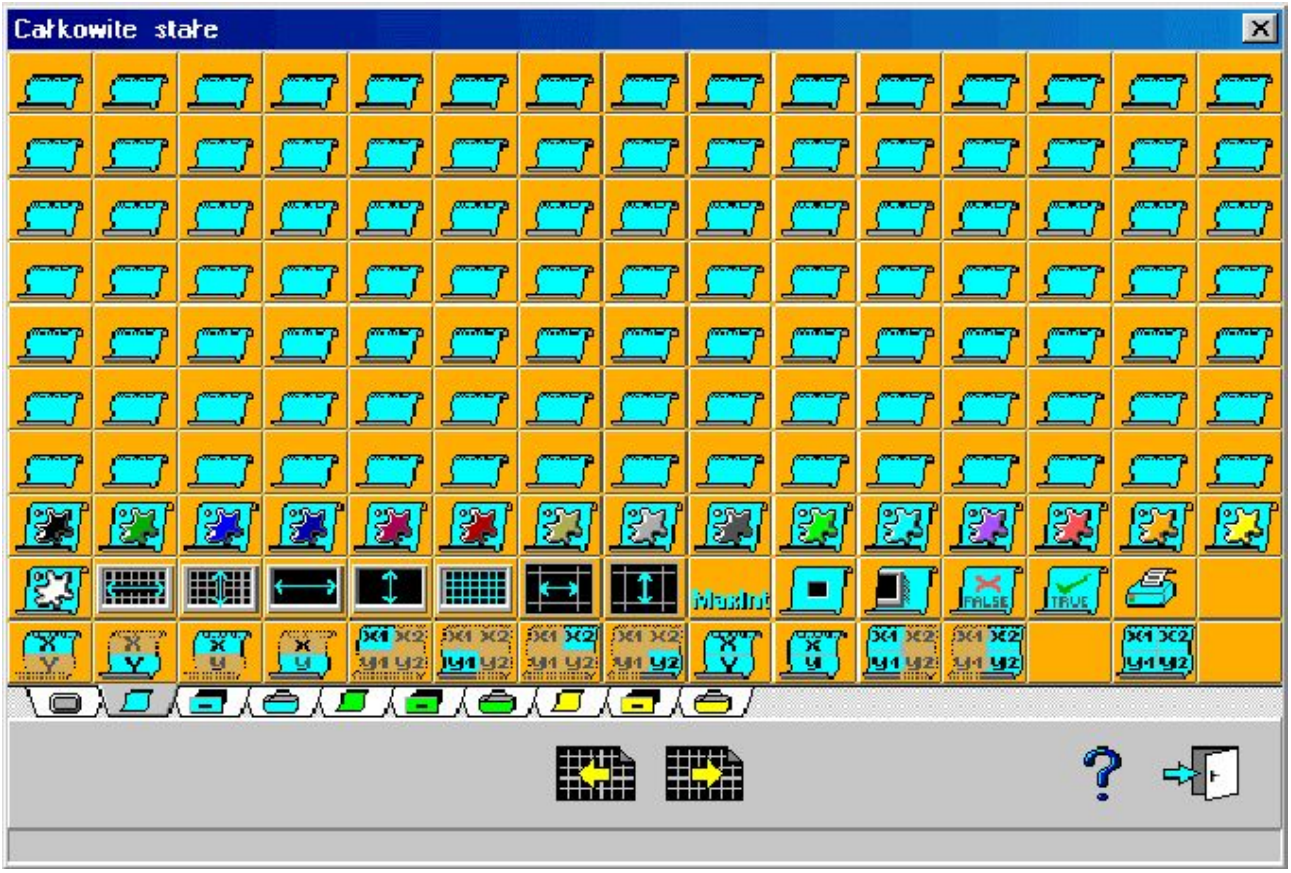
Podobnych problemów pojawia się wiele podczas pracy z literałami. Dlatego programiści używają literałów tylko w przypadkach, gdy z konstrukcji programu wynika, że w tym miejscu nie będzie nigdy innej liczby (innego napisu). We wszystkich innych sytuacjach preferują **stałe z nazwą** (literał jest traktowany jako stała bez nazwy). Ponieważ nie będziemy nigdzie używać terminu stała bez nazwy tylko terminu literał, w dalszym tekście stałe z nazwą określać będziemy po prostu jako **stałe**.

Rysunek 14.1
Przycisk Klawisze, stałe, zmienne

Baltie oczywiście ułatwia korzystanie ze stałych. Chowa je w specjalnych bankach, do których można dostać się naciśnięciem przycisku **Klawisze, stałe, zmienne**, który znajduje się w panelu narzędzi na prawo od przycisku **Pomocnicy** (patrz rysunek 14.1). Po jego naciśnięciu otworzy się okno, które można zobaczyć na rysunku 14.2.

To okno zawiera 10 banków, nazwa banku bieżącego jest widoczna w nagłówku okna. Będziemy zajmować się teraz tylko bankami stałych, tzn. bankami, w których na zakładce jest zgięta kartka papieru. Są tylko trzy takie banki: bank z błękitną kartką zawiera *stałe całkowite* (na rysunku 14.2 ten bank jest akurat widoczny), bank z zieloną kartką zawiera *stałe rzeczywiste* a bank z żółtą kartką zawiera *stałe napisowe*. Pamiętamy oczywiście skąd taki dobór kolorów.

W każdym banku jest 150 elementów. W przedostatnim lub w dwóch przedostatnich wierszach są *stałe predefiniowane*, w których znajdują się już przygotowane najczęściej używane wartości. W banku stałych całkowitych, który jest na rysunku 14.2, są predefiniowane stałe dla każdego koloru Baltiego, dla rozmiarów dworku Baltiego, przedmiotów i kilka innych. Ostatni wiersz zawiera stałe specjalne, których znaczenie wyjaśnimy później.



Rysunek 14.2
Banki klawiszy, stałych i zmiennych

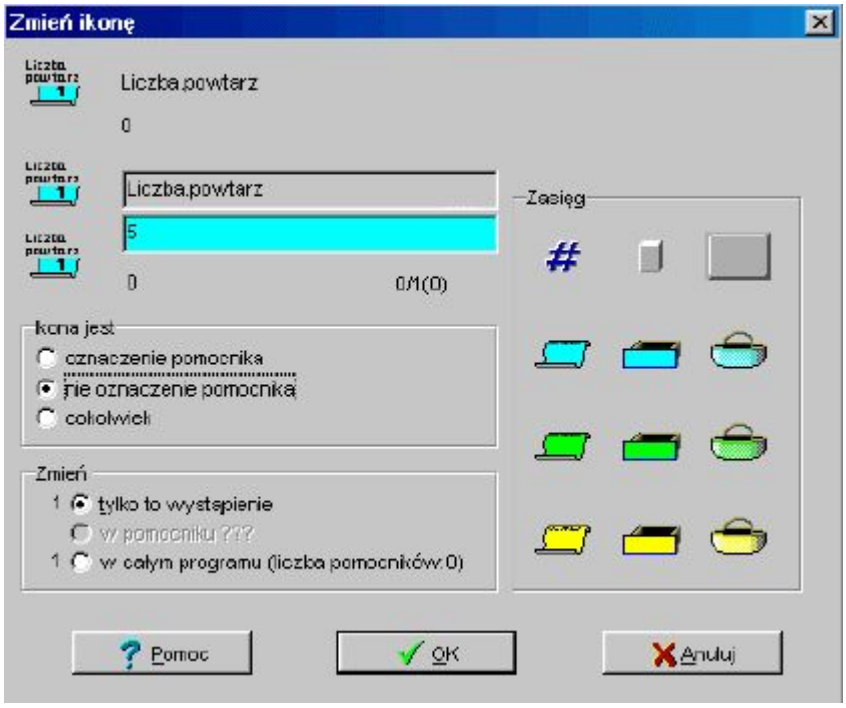
Jeśli nie będzie jasne znaczenie jakiejś stałej predefiniowanej, wystarczy kliknąć w oknie z bankami zmiennych i stałych na znaku zapytania (pomocy). Otworzy się pomoc z ustawionym tematem *Stałe,* na końcu którego znajdziemy listę wszystkich stałych predefiniowanych z ich znaczeniem i uwagami dotyczącymi ich użycia.

Wprowadzenie stałej do programu

Jeśli będziemy chcieli użyć w swoim programie stałej, otwieramy okno z bankami stałych. Jeśli nie jest ustawiony bank, który potrzebujemy, klikamy na zakładkę odpowiedniego banku. Potem już wystarczy kliknąć na elemencie z odpowiednią stałą i w znany sposób wprowadzić wybraną stałą do programu.

Jeśli wprowadzamy jedną z pustych stałych, po wprowadzeniu jej do programu otworzy się okno dialogu **Zmiana elementu** (patrz rysunek 14.3), w którym wprowadzimy nazwę stałej i jej wartość. Te atrybuty stałej nie zmienią się podczas działania programu (będą stałe).

Popatrzmy teraz, co można w tym oknie dialogowym wprowadzić i jak można to osiągnąć. Przeglądamy okno z góry na dół.



Rysunek 14.3
Okno dialogu Zmiana elementu

W lewym górnym rogu Baltie wyświetla aktualną postać elementu, tzn. wygląd, który element miał przed otwarciem okna dialogu. Przy pierwszym użyciu zamiast nazwy stałej będzie tu numer elementu w banku a w miejscu wartości nie będzie nic, bo jeszcze nie została ona wprowadzona. Gdy popatrzymy na rysunek 14.3, zobaczymy, że jest trochę inaczej – pokazano w nim, jak zmienić wartość stałej *Liczba powtarzań* z 0 na 5.

Pod początkowym wyglądem elementu jest umieszczony wygląd elementu, tzn. postać, jaką element będzie mieć po naciśnięciu przycisku **OK.** Na prawo są dwa pola wprowadzania:

- Do górnego pola wprowadzamy nazwę elementu, przy czym możemy po lewej ciągle śledzić, jak będzie wyglądać element po wprowadzeniu nazwy. Zauważmy jednak, że w nazwie nie można użyć liter diakrytycznych (ąśćń czy ł), są ignorowane przez Baltiego. Na rysunku po lewej można też sprawdzić, czy nazwa stałej jest poprawnie podzielona na wiersze. Jeśli nie odpowiada nam sposób podziału możemy dostosować go wprowadzając kilka odstępów.
- W dolnym polu wprowadzamy wartość stałej. Podczas podawania wartości Baltie kolorem tła pola informuje nas o rozpoznanym typie wartości. Pamiętajmy, że jeśli typ wartości nie będzie się zgadzać się (np. jeśli będziemy do stałej całkowitej wprowadzać tekst), Baltie nie pozwoli na zatwierdzenie blokując naciśnięcie przycisku**OK**.

W dowolnym momencie gdy chcemy zmienić nazwę i/lub wartość stałej wskazujemy stałą w programie lub otwieramy okno **Klawisze, stałe, zmienne** i w nim wskazujemy na zmienianą stałą. Naciśnięciem klawisza **F2** lub poleceniem **Edycja** z menu lokalnego otwieramy okno **Zmiana elementu** i wprowadzić nową nazwę i/lub wartość.

Za pomocą okna **Zmiana elementu** możemy też zastąpić element innym. Jeśli klikniemy prawym przyciskiem myszy na jedną z widocznych z pokazanych wersji elementu otworzy się okno z bankami, w których możemy wybrać element, który zastąpi w programie element zmieniany. Możliwość ta dotyczy też elementów z prawej części. Można to wykorzystać do zmiany typu elementu, bo okno z bankami zawsze otworzy się tak, że będzie w nim ustawiony bank z typem elementu, na który kliknięto.

Przy jakiejkolwiek zmianie musimy pomyśleć o zasięgu zmiany. Przełącznikiem w polu **Zmienić** ustawimy, czy zmieniamy tylko kliknięty obiekt, czy wszystkie takie elementy w pomocniku, czy w całym programie.

Trzeba liczyć się z tym, że niektórych zmian (np. zmiany nazwy lub wartości obiektu) nie wolno wykonać dla jednego elementu, lecz je trzeba wykonać dla całego programu. Zanim nie zostanie to właściwie ustawione, przycisk**OK** pozostanie nieczynny.

Z poprzedniego akapitu widać przyczynę często popełnianego błędu – niepoprawne wprowadzenie zasięgu zmian. Jeśli Baltie nie będzie chciał przyjąć zmiany, najprawdopodobniej, źle został ustawiony zasięg zmian.

14.2 Zmienne

Dotychczas pracowaliśmy tylko z obiektami, których wartość znaliśmy już podczas tworzenia programu lub najpóźniej przed jego uruchomieniem. Teraz opowiemy o obiektach, których wartość może się zamieniać podczas działania programu. Programiści nazywają te obiekty **zmienne.**

Baltie zawsze próbuje pokazywać użytkownikom ikony wyraźnie oznaczające cechę elementu. Dla zmiennych wykorzystano symbol szuflady. W szufladzie można pozostawić rzeczy, które później możemy z szuflady wyjąć. Tak samo jest ze zmiennymi. Służą właśnie do tego, byśmy w nich schowali dane, o których wiemy, że za chwile będą potrzebne. W odpowiednim momencie możemy skorzystać z wartości z szuflady.

Zmienna działa tak jak *Schowek* w systemie *Windows*: możemy do niej schować tylko jedną informację. Po pobraniu wartości ze zmiennej informacja pozostaje dalej w zmiennej tak, jakbyśmy tylko zajrzeli do szuflady, lecz nie wyjmowali z niej zawartości. Po wprowadzeniu nowej informacji stara zawartość zmiennej zostanie usunięta.

Jeśli chcemy wykorzystać wartość przechowywaną w zmiennej, wprowadzamy do programu element ze zmienną dokładnie tak, jak wprowadzamy liczbę. W programie widocznym na rysunku 14.4 Baltie zrobi tyle kroków jaka jest wartość zmiennej **Szerokość,** obróci się w lewo i wyświetli przed sobą tekst przechowywany w zmiennej tekstowej **Facet.**

Rysunek 14.4
Użycie zmiennych

Przypisanie i zmiana wartości

Jeśli będziemy chcieli przechować w zmiennej nową wartość, używamy strzałki przypisania, którą znamy z wyświetlania danych na ekranie. Na pewno znamy formułę $P = \pi r^2$, według której oblicza się powierzchnię okręgu o podanym promieniu. Jeśli oznaczymy szufladę zmiennej rzeczywistej **P** jako **Powierzchnia** i szufladę **R** jako **Promień**, to program z rysunku 14.5 obliczy powierzchnię okręgu o promieniu **Promień** i przechowa wynik w zmiennej **Powierzchnia** (element π znajduje się wśród stałych rzeczywistych).

Rysunek 14.5
Przydzielamy wartość zmiennej

Możemy skrócić program zawierający więcej tych samych przypisań. Baltie pozwala przydzielić jedną wartość kilku zmiennym na raz. Wystarczy że po lewej stronie strzałki przypisania wprowadzimy wszystkie zmienne, którym chcemy przypisać tą samą wartość. Program z rysunku 14.6 przydzieli jednym ruchem wartość 5 zmiennym **Szerokość**, **Wysokość** i **Długość.**

Rysunek 14.6
Przypisanie grupowe

Jedną z najczęstszych zmian wartości zmiennych jest ich zwiększenie, zmniejszenie lub inna modyfikacja wyprowadzona z poprzedniej wartości zmiennej. Ponieważ te operacje są dość częste Baltie ma dla nich przygotowane specjalne elementy **Zwiększ** i **Zmniejsz**. Jeśli w poleceniu za zmienną umieścimy jeden z tych elementów spowoduje to zwiększenie lub zmniejszenie wartości zmiennej o jeden. Jeśli dodatkowo za strzałką umieścimy liczbę wartość zmiennej, zostanie powiększona lub pomniejszona o tę liczbę. Inne możliwości zastosowania tych elementów można zobaczyć w pomocy. Program z rysunku 14.7 powiększy wartość zmiennej **Wysokość** o jeden, wartość zmiennej **Szerokość** zwiększy o 2 a wartość zmiennej **Długość** zmniejszy o 3.

Rysunek 14.7
Zmiana wartości

Obiekty innych typów

Baltie pozwala nam pracować nie tylko z liczbami i tekstami, lecz także z przedmiotami. Przedmioty nie mają osobnych banków zmiennych, lecz przechowujemy je jako liczby całkowitych. Kiedy chcemy przechować przedmiot w zmiennej, Baltie zamieni go na liczbę, którą zapamięta w zmiennej. Jeśli chcemy użyć przedmiotu w wyrażeniu, np. chcemy wyczarować przedmiot z numerem o 2 większym, musimy samodzielnie zamienić przedmiot na liczbę. Także odwrotnie, gdy chcemy wykorzystać ponownie numer przedmiotu jako przedmiot, musimy sami dokonać zamiany

(konwersji).

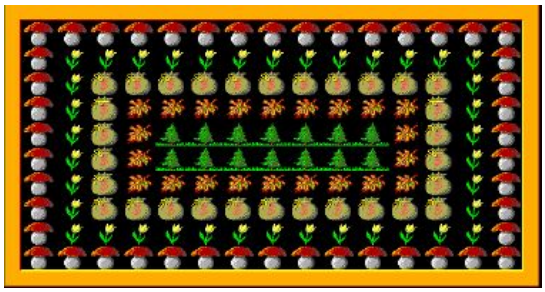
Do konwersji przedmiotu na liczbę (numer przedmiotu) służy element **Konwersja na numer** a do konwersji liczby na przedmiot element **Dowolny przedmiot,** które znajdziemy przy prawej krawędzi panelu elementów – patrz rysunek 14.8. Więcej o nich można przeczytać w pomocy. Używano ich podobnie jak elementu **Konwersja na ciąg**:

– jeśli w programie przed przedmiot wprowadzimy element **Konwersja na numer** otrzymamy liczbę całkowitą określającą numer tego przedmiotu,

– jeśli wprowadzimy przed liczbą element **Dowolny przedmiot** otrzymamy przedmiot, który zostanie wyczarowany.

Rysunek 14.8
Konwersja na numer i na przedmiot

Wszystkiego, czego nauczyliśmy się w tym rozdziale, użyjemy teraz w programie, który nauczy Baltiego wyczarować na swoim dworku prostokąty leżące wewnątrz siebie. Czarować będzie z przedmiotów, które są umieszczone w banku kolejno. Efekt może być taki jak na rysunku 14.9.



Rysunek 14.9
Prostokąty

Program realizujący to zadanie znajduje się na rysunku 14.10. Od razu wyjaśnimy użyte w nim konstrukcje. Dla lepszej czytelności opisu przed każdym wierszem umieszczono komentarz z numerem wiersza programu. Przejdźmy teraz kolejno przez wszystkie wiersze programu:

1. Do zmiennej **Szerokość** wprowadzimy liczbę kroków, które Baltie może zrobić poziomo. Wiemy, że jest ich o jeden mniej, niż pól poziomych. Obliczymy je korzystając ze stałej predefiniowanej.

2. Do zmiennej **Wysokość** wprowadzimy liczbę kroków, które Baltie może zrobić w pionie.

Rysunek 14.10
Program budujący prostokąty ze wspólnym środkiem

3. Do zmiennej **Przedmiot** wprowadzimy numer pierwszego wyczarowanego przedmiotu. Ten przedmiot Baltie wyczaruje w zewnętrznym prostokącie.
4. W przestrzeni Baltiego mieści się 5 zagnieżdżonych prostokątów, dlatego w pętli wyczarujemy 5 razy prostokąt z odpowiednich przedmiotów.
5. To już znamy: prostokąt tworzymy jako parę jego kątów.
6. **Szerokość**-razy przed sobą Baltie wyczaruje przedmiot, którego numer jest pamiętany w zmiennej **Przedmiot**, po czym przejdzie na następne pole. Pierwsze pole pozostanie puste, bo liczymy na to, że zostanie wypełnione podczas tworzenia ostatniego boku. Na koniec Baltie się obróci by wyczarować następny bok.
7. Jak poprzednio, lecz tworzymy bok pionowy, dlatego pętla zostanie powtórzona **Wysokość**-razy.
8. Koniec pętli, tworzącej prostokąt. Prostokąt jest gotowy i Baltie stoi na pozycji, z której zaczął go rysować.
9. Wewnętrzny prostokąt będzie miał każdy bok o 2 pola krótszy, powinniśmy więc o 2 zmniejszyć wartości zmiennych określających ile kroków zrobi Baltie podczas tworzenia boków prostokąta.
10. Poszczególne prostokąty mają zostać wyczarowane z przedmiotów, które są kolejno za sobą w banku. Dlatego prostokąt wewnętrzny będzie zbudowany z przedmiotów, których numer będzie o jeden większy od numeru przedmiotu z prostokąta zewnętrznego. Powiększymy więc zawartość zmiennej **Przedmiot** o jeden.
11. Przesuniemy Baltiego do początku prostokąta wewnętrznego, tzn. do jego lewego dolnego roku i obrócimy Baltiego na wschód.
12. Koniec pętli tworzącej prostokąty. Teraz wszystkie prostokąty powinny już być narysowane i Baltie powinien być gdzieś w środku swojego dworku.
13. Na koniec tylko mały drobiazg dla dodatkowego efektu: Baltie wyczaruje przed sobą drzwi, otworzy je i wejdzie do środka (robiliśmy już coś podobnego). Zamknie jeszcze drzwi za sobą.
14. Zakończymy wyczarowując zamiast drzwi przedmiot, który był na tym miejscu przed wyczarowaniem drzwi i oczywiście wtedy drzwi znikną. Powinniśmy przy tym pamiętać, że na końcu każdego obrotu pętli tworzącej nowe prostokąty powiększaliśmy numer czarowanego przedmiotu o jeden. Zatem na końcu wzmiennej **Przedmiot** jest liczba o jeden większa niż numer ostatnio użytego przedmiotu. W miejscu drzwi powinniśmy zatem wyczarować przedmiot, którego numer jest o jeden mniejszy niż wartość zmiennej **Przedmiot.** Na końcu programu już tylko dodamy „obowiązkowe" czekanie na klawisz.

14.3 Wprowadzenie przedmiotu podczas działania programu, banki i przedmioty

Najpierw powinniśmy dowiedzieć się, jak można podczas działania programu wprowadzić przedmiot, którego będziemy później w programie używać, jak zapamiętać taki przedmiot i jak go później użyć.

Rysunek 14.11
Otrzymanie elementu Przedmioty

Jeśli chcemy pozwolić użytkownikowi programu na wybranie przedmiotu podczas działania programu, wykorzystujemy element **Przedmioty**. Elementu tego nie znajdziemy się w panelu elementów. Otrzymamy go klikając prawym przyciskiem myszy w panelu narzędzi na przycisku **Przedmioty** i w otwartym menu lokalnym (patrz rysunek 14.11) wprowadzając polecenie **Chwyć.** Chwycony element umieszczamy w programie.

Gdy Baltie natrafi na ten element podczas wykonywania programu, otworzy okno z bankami przedmiotów i pozwoli użytkownikowi wybrać przedmiot podobnie jak przy tworzeniu sceny. Jeśli element **Przedmioty** występuje w programie bez dodatkowych elementów, Baltie od razu go wyczaruje. Widać to na na rysunku 14.12, gdzie pierwszy wiersz programu pozwoli użytkownikowi wybrać przedmiot, który zostanie natychmiast wyczarowany w polu o współrzędnych (3 , 4).

Rysunek 14.12
Wyczaruj wybrany przedmiot

Wybranego przedmiotu nie musimy natychmiast wyczarować. Można przechować jego numer tak, jak w programie z rysunku 14.13. Numer wybranego przedmiotu zostanie najpierw zapamiętany w zmiennej **Przedmiot,** potem Baltie może wyczarować ten przedmiot wspólnie z jego prawym i lewym sąsiadem wbanku (lewy sąsiad ma numer o jeden mniejszy a prawy o jeden większy).

Numer przedmiotu składa się z dwóch części:

- ostatnie trzy cyfry określają pozycję przedmiotu w banku (mogą mieć wartość od 1 do 150),
- cyfry przed nimi określają numer banku (mogą mieć wartość od 0 do 199).

Jeśli w zmiennej pamiętamy numer przedmiotu, możemy wyliczyć z tego numeru obie części. Numer banku otrzymamy jako wynik dzielenia całkowitego przez tysiąc, numer przedmiotu otrzymamy jako resztę z dzielenia przez tysiąc. Program z rysunku 14.14 wyjaśnia wszystko na przykładach.

Rysunek 14.13
Zapamiętaj przedmiot i wykorzystaj go później

Rysunek 14.14
Dla wybranych przedmiotów wyświetlaj ich pozycję w banku i numer banku

14.4 Kółko i krzyżyk na życzenie

Najwyższy czas spróbować znów samodzielnego programowania. Spróbujmy ulepszyć program gry w kółko i krzyżyk dodając możliwość wybrania przedmiotu, którym zostanie pokryty dworek oraz przedmiotów, których gracze będą używać do budowania swych linii.

Nie zapominajmy, że porządny program powinien użytkownika poinformować, czego od niego oczekuje. Dopiero po podaniu odpowiedniego komunikatu powinien odebrać informację od użytkownika. Biorąc pod uwagę nasze umiejętności najlepiej będzie wyświetlić na ekranie tekst, w którym podamy swoją prośbę. Potem poczekamy, aż użytkownik przeczyta ją i dopiero potem otworzymy okno z bankami przedmiotów, z których użytkownik wybierze przedmiot.

Gdy stworzymy swoją wersję, możemy porównać ją z programem z rysunku 14.15. Dla lepszego zrozumienia znowu omówimy cały przykładowy program.

Od razu po ustawieniu maksymalnej szybkości przydzielamy zmiennej tekstowej **Klawisz** tekst przyszłego komunikat. Zwróćmy uwagę na odstęp na końcu tekstu. Pozwoli nam to kilka razy powtarzać ten komunikat bez potrzeby wprowadzania do programu tego samego długiego literału. Ponadto gdybyśmy chcieli później zmienić tekst, trzeba by to zmienić w trzech miejscach. Teraz wystarczy zmienić go tylko w miejscu przypisania do zmiennej. Z punktu widzenia czystości programu lepiej byłoby wprowadzić ten tekst do stałej, bo w trakcie działania programu nie powinien się zmienić.

Komunikat będzie pokazywany przed Baltiem, dlatego najpierw trzeba przenieść go na miejsce, gdzie zostanie wyświetlona pierwszy komunikat. Komunikat wypiszemy w wierszu nr 1, żeby tekst nie przykleił się do górnej krawędzi dworku.

Teraz poprosimy użytkownika, by wybrał przedmiot na tło. Wyświetlimy przed Baltiem tekst ze słowem „TŁO”. Poczekamy, aż użytkownik naciśnie klawisz lub przycisk myszy i pokażemy mu okno z bankami przedmiotów, z którego będzie mógł wybrać przedmiot. Numer wybranego przedmiotu zapamiętamy z zmiennej **Waga** i od razu pokażemy numer za komunikatem. Dlatego podajemy w poleceniu wyświetlania tylko współrzędną **X**, bo chcemy by współrzędna **Y** przedmiotu była taka sama jak współrzędna **Y** Baltiego.

Po wywołaniu elementu **Przedmioty** pojawi się na ekranie okno z otwartym bankiem 0. Jeśli zechcemy otworzyć okno z innym bankiem, musimy za element **Przedmioty** wprowadzić numer tego banku. Ponieważ możemy oczekiwać, że użytkownik będzie wybierać wszystkie trzy przedmioty z tego samego banku, obliczymy numer banku, z którego został wybrany przedmiot dla tła i zapamiętamy go w zmiennej **Bank.**

Następna część programu widocznego na rysunku 14.15 jest nieco zagęszczona tak, żeby zmieścił się cały na stronie. Przy okazji zwróćmy uwagę na to, że Baltie może czasami połączyć elementy w polecenia inaczej, niż chcieliśmy. W takiej sytuacji najprostszym rozwiązaniem jest włożyć odstęp między dwa niepoprawnie interpretowane polecenia. Baltie zakończy pierwsze polecenie przed odstępem a drugie rozpocznie za nim.

Rysunek 14.15
Kółko i krzyżyk pozwalający na wybór przedmiotów

Jak można zobaczyć w zagęszczonej części programu, Baltie przejdzie o dwa wiersze poniżej, poprosi o przedmiot dla pierwszego gracza, zapamięta go i wyświetli. To samo zrobi z przedmiotem dla drugiego gracza. Na koniec ogłosi, że po następnym naciśnięciu klawisza rozpocznie się gra.

Następnej części już nie musimy wyjaśniać, bo jest praktycznie taka jak w naszym poprzednim programie „Kółko i krzyżyk". Różni się tylko tym, że zamiast znajomych przedmiotów wyczarujemy na odpowiednich pozycjach przedmioty, których numery są zapamiętane w odpowiednich zmiennych. Przy okazji uprościliśmy program wyświetlając w lewym górnym kącie dworku przedmiot gracza, który ma wykonać ruch.

14.5 Koszyki – prywatne dane pomocników

Już parę razy powiedzieliśmy w tej książce, że program powinien przede wszystkim być czytelny. Jest to bardzo ważne, bo najwięcej czasu podczas programowania zajmuje szukanie i usuwanie błędów. Wszystkie współczesne języki programowania próbują jak najlepiej wspomagać programistę w tym zakresie.

Jedną z metod poprawienia czytelności programu jest możliwość umieszczenia w pomocnikach zmiennych, o których nikt inny (tzn. żadna inna część programu) nie wie, więc nie może ich zmienić (np. przez nieuwagę).

Taką możliwość oferuje też Baltie. Jak pamiętamy oprócz banków stałych i banków zmiennych (z szufladami) mamy też banki z koszykami. Koszyki to są zmienne, które mogą być używane tylko przez pomocników. Przy tym pomocnik ma zagwarantowane, że koszyk jest jego wyłączną własnością. Nie musimy więc zastanawiać się, czy przypadkiem dany koszyk nie jest wykorzystywany przez innego pomocnika. Koszyk jest pod wyłączną kontrolą właściciela. Nawet kiedy pomocnik zawoła do pomocy kopię samego siebie, jego dubler otrzyma swoje własne koszyki. Możemy więc spokojnie przyjąć, że bez względu na to, co robi reszta programu, to, co pomocnik schowa w swoim koszyku, to znajdzie.

Sytuację z szufladami i koszykami możemy sobie wyobrazić jeszcze inaczej. Szuflady umieszczone są w szafie szufladowej, do której każdy może w dowolnej chwili włożyć lub wyjąć dane. Nie jest ważne kto wprowadził informację do szuflady. Jeden pomocnik może wprowadzić informację, drugi może ją sprawdzić i zmienić, trzeci może z niej skorzystać.

Natomiast koszyki każdy pomocnik nosi ze sobą, więc gdy dwaj pomocnicy mają koszyk o tej samej nazwie, to w rzeczywistości są to różne koszyki różnych pomocników. Koszyki jednego pomocnika są dla innych pomocników niewidzialne i przez to niedostępne.

Programiści używają dla naszych szuflad i koszyków mądrzejszych terminów: szuflady nazwaliby **zmiennymi globalnymi**, tzn. zmiennymi widocznymi i dostępnymi w całym programie. Koszyki nazwaliby **zmiennymi lokalnymi**, tzn. zmiennymi, które są widoczne tylko w określonym miejscu – wewnątrz pomocnika.

Podawanie parametrów

Wspaniale, że pomocnik może używać swoich własnych zmiennych. Jeszcze przyjemniejszy jest fakt, że przed rozpoczęciem pracy pomocnik pozwoli nam przydzielić koszykom wartości początkowe i jest w stanie po skończeniu swojej pracy wartości swoich koszyków przydzielić wskazanym przez nas zmiennym.

Koszyki, których wartości można ustawić lub na otrzymać po zakończeniu działania pomocnika nazywamy **parametrami**. Możemy podzielić parametry na:

- **parametry wejścia**, to te, którym przydzielimy wartość początkową, ale ich wartość po skończeniu pracy pomocnika nas nie interesuje,
- **parametry wyjścia,** to te, których wartość interesuje nas po zakończeniu pracy pomocnika, lecz nie określamy ich wartości początkowej,
- **parametry wejścia i wyjścia,** to te, którym przydzielamy wartość początkową i na końcu chcemy znać ich wartość końcową.

Jeśli zechcemy przydzielić koszykowi pomocnika wartość początkową i nie potrzebujemy wartości tego koszyka po zakończeniu pracy pomocnika, wprowadzamy do polecenia wywołania pomocnika za odznaczenie koszyka odpowiednie przypisanie.

Rysunek 14.16
Przypisanie wartości początkowych koszykom pomocnika

Na rysunku 14.16 widzimy fragment programu, w którym wywoływany jest pomocnik **Wyświetl** i podczas wołania ustawiane są trzy jego parametry (koszyki): parametrowi **Wiersz** przydzielamy

wartość **3**, do parametru **Kolumna** kopiujemy wartość koszyka z tą samą nazwą, ale należącego do pomocnika wywołującego, parametrowi **Tekst** przydzielamy tekst, który pomocnik powinien pokazać na ekranie.

Zwróćmy przede wszystkim uwagę na umieszczone w pierwszym wierszu programu przypisanie **Kolumna ← Kolumna,** które mówi, że pomocnik, który woła pomocnika **Wyświetl,** wstawi do jego koszyka **Kolumna** zawartość swojego koszyka **Kolumna.** Przypominamy tylko, że chodzi o dwa **różne koszyki**, chociaż mają tą samą nazwę.

Jeśli interesuje nas wartość końcowa koszyka wywołanego pomocnika, musimy przygotować zmienną, do której po skończeniu pracy pomocnika przydzielimy wartość z koszyka. Zmienna musi mieć typ zgodny z typem koszyka, którego wartość po skończeniu pracy pomocnika pobierze.

Swoje zainteresowanie końcową wartością koszyka określimy dodając do polecenia wywołania pomocnika element z odpowiednim koszykiem i bezpośrednio za nim element ze zmienną (wszystko jedno, czy będzie to szuflada, czy koszyk), w której ma zostać umieszczony wynik. Przykład takiego wywołania można zobaczyć na rysunku 14.17.

Zatrzymajmy się przy tym rysunku. W drugim wierszu wołamy pomocnika **Otrzymaj** zaznaczając, że na końcu pobierzemy od niego wartości z jego koszyków **Wiersz** i **Kolumna** do swoich koszyków z odpowiednimi nazwami. Taka forma zapisu może być czasami nieczytelna. Dlatego w piątym wierszu pokazujemy, że można pomiędzy poszczególne parametry wstawić przecinek dla lepszej czytelności programu.

Drugą rzeczą, na którą zwrócimy uwagę jest wywołanie pomocnika **Gdzie.** W pierwszym wywołaniu przydzielamy jego koszykom wartości początkowe – to jest zupełnie w porządku. Jednak w drugim wywołaniu (na końcu programu głównego) koszykom przydzielamy nie tylko wartości początkowe, ale całe koszyki. Na pierwszy rzut oka niczym to nie grozi, bo pomocnik **Gdzie** nie zmieni zawartości koszyków. Poza tym wywołanie pomocnika **Gdzie** jest ostatnim poleceniem programu głównego. Tak to wygląda na pierwszy rzut oka. Takie przydzielanie parametrów jest niepoprawne i **bardzo niebezpieczne**.

Rysunek 14.17
Pobranie końcowych wartości parametrów

Dlaczego? Bo musimy przecież zawsze uważać, że być może zmienimy program w przyszłości i błąd, który teraz nie spowoduje żadnych konsekwencji, bo program się skończy, może zacząć zmieniać działanie programu, gdy dopiszemy kolejne polecenia. W takim prościutkim programie to zapewne zauważymy i zmienimy niepoprawne wywołanie pomocnika, ale polecamy:

Pisząc proste programy uczmy się zachowywać tak, jakbyśmy pisali programy bardzo rozbudowane.

W przeciwnym razie nabędziemy złych przyzwyczajeń, których później będzie bardzo trudno się pozbyć.

Zalecane konwencje używania koszyków w pomocnikach

Na rysunku 14.18 mamy trochę dłuższy program, na którym pokażemy zalecane konwencje używania zmiennych lokalnych (koszyków) i parametrów w pomocnikach.

Przepraszamy za złamanie zasady, że program główny będzie zdefiniowany jako pomocnik, ale w tym przypadku program powiększyłby się o następne 5 wierszy. Żeby zmieścić się na jednej stronie konieczne byłoby zmniejszenie rysunku, a to pogorszyłoby czytelność.

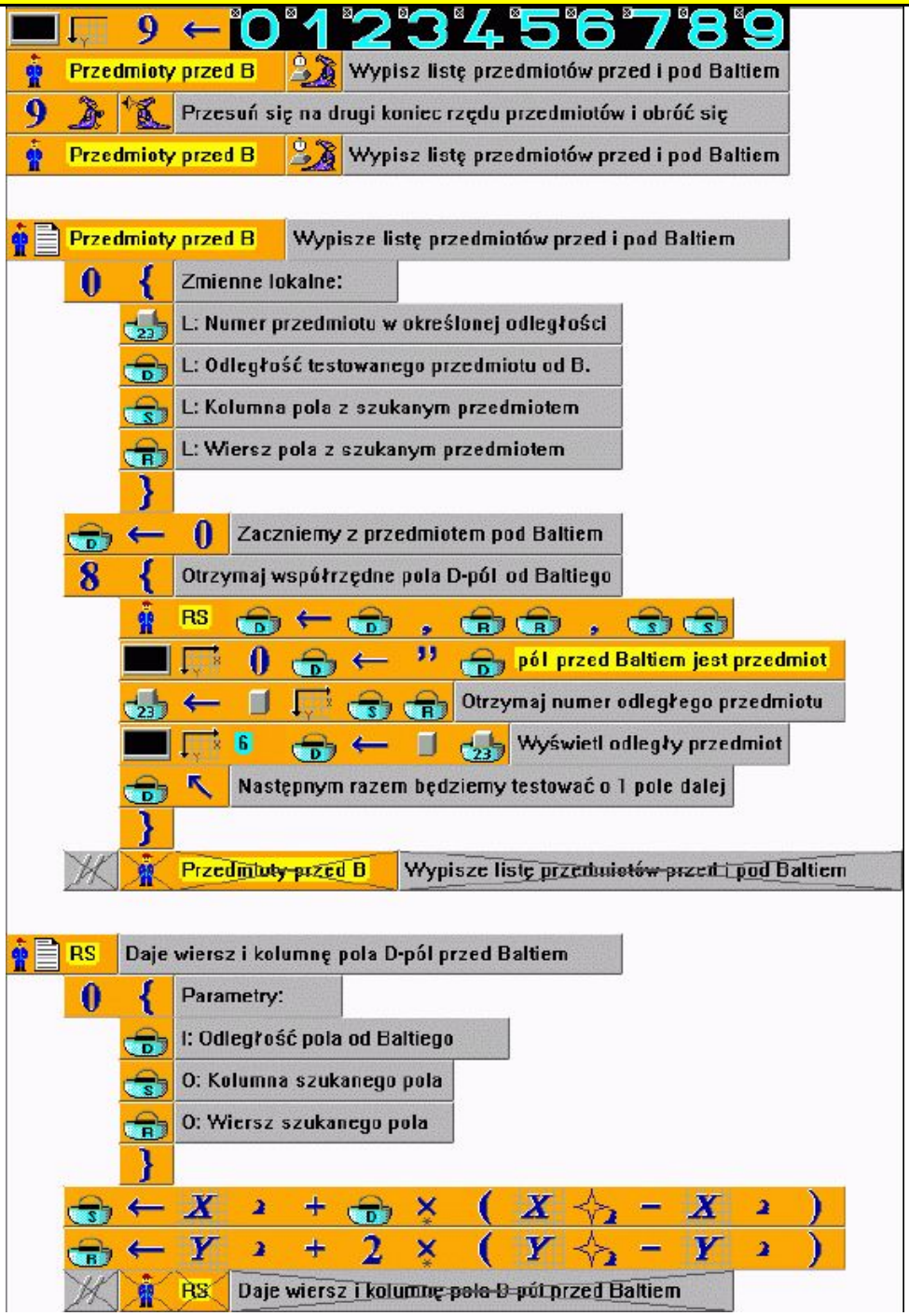
W programie użyto dwóch pomocników używających lokalnych zmiennych (koszyków). Dobrym zwyczajem jest wypisać na początku każdego pomocnika wszystkie koszyki, których pomocnik będzie używać. Warto też do każdego z nich dodać komentarz wyjaśniający przeznaczenie koszyka. Nie jest ważne, czy do tego użyjemy bloku, który nigdy nie zostanie wykonany (jak na rysunku), czy użyjemy komentarza wierszowego lub blokowego. Ważne, by przyzwyczaić się do pokazania i opisania na początku pomocnika wszystkich używanych koszyków. W następnej części

pokażemy polecenia pomocne w pokazaniu i opisaniu zmiennych i stałych globalnych.

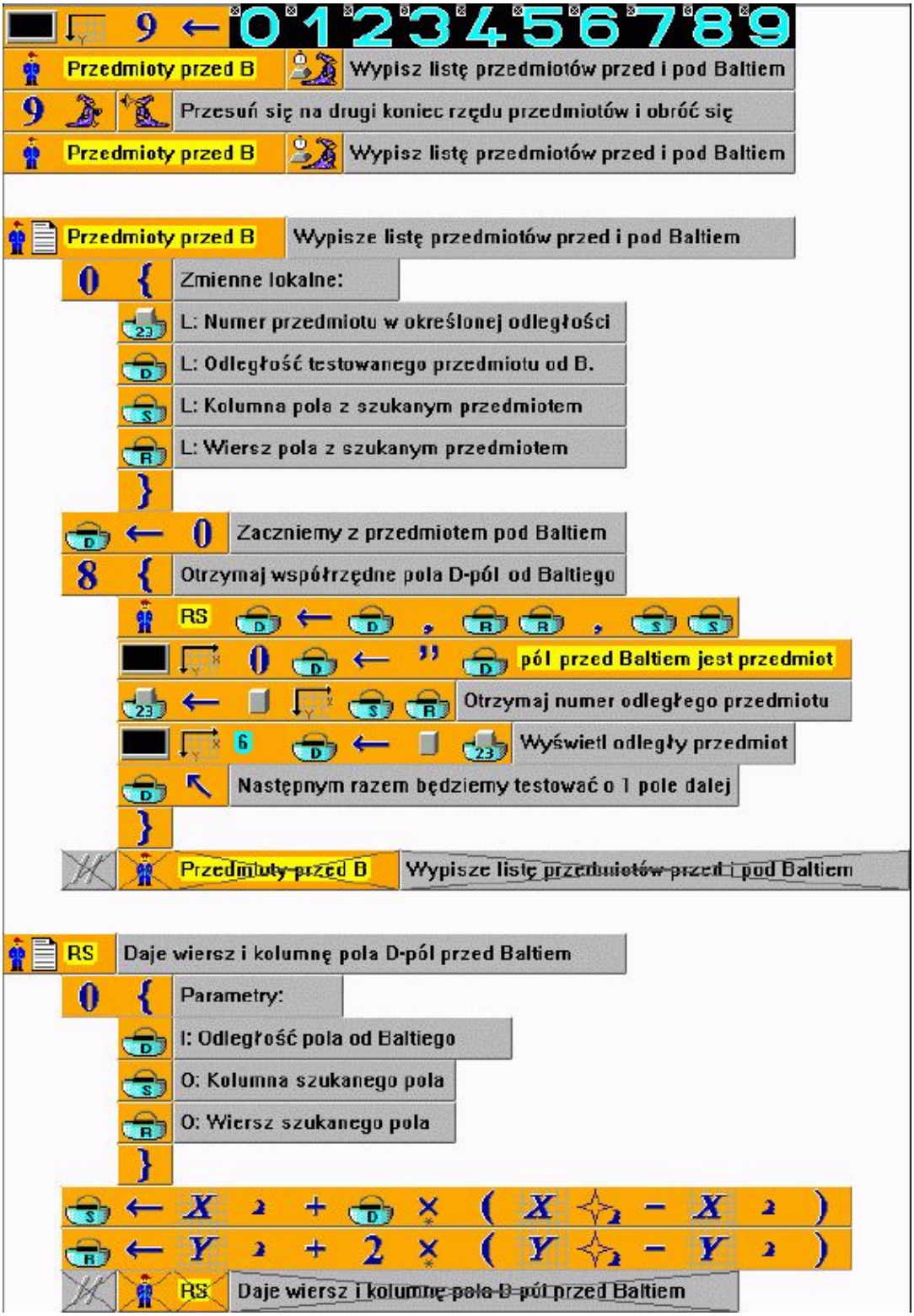
Nie ulegajmy wrażeniu, że jest to marnowanie miejsca. Gdy będziemy tworzyć długie programy okaże się, że czas spędzony nad pozornie zbędnymi komentarzami w programie wróci się jako czas zaoszczędzony przy szukaniu błędów i analizie zmienianych pomocników.

Na początku komentarza opisującego znaczenie zmiennej warto wprowadzić jakiś znak, po którym później poznamy, czy chodzi o zmienną lokalną, czy też o parametr, zaś w przypadku parametru to, czy jest to parametr wejścia, czy też wejścia i wyjścia.

Baltie, tak jak większość języków programowania, nie ma zmiennych pełniących tylko funkcję wyjścia, bo każdy koszyk, który na końcu skopiuje swoją zawartość do innej zmiennej, może na początku przeczytać swoją wartość początkową.



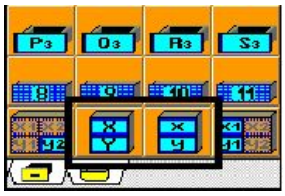
Ponieważ słowa *wejście* i *wyjście* zaczynają się od tej samej litery, w przykładowych programach używamy liter początkowych angielskich oznaczeń: parametry wejścia odznaczamy literą **I** (***I**nput*), parametry wyjścia **O** (***O**utput*) i dla koszyków, które nie są parametrami, używamy litery **L** (***L**ocal*). Oczywiście można wymyślić inną konwencję nazywania. Ważne, by konsekwentnie jej przestrzegać.



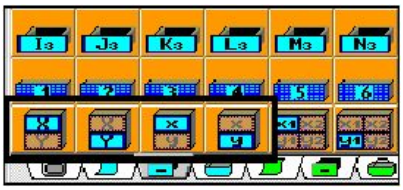
Rysunek 14.18
Program testujący i wyświetlający przedmioty daleko przed Baltiem

14.6 Jak najlepiej zapamiętać współrzędne

Jak już wiemy, współrzędne ekranu opisywane są przez parę danych. Pracując ze zmiennymi musieliśmy zawsze używać dwóch zmiennych, X i Y. Ponieważ często wykorzystuje się współrzędne w programach Baltie, autorzy Baltie przygotowali małe ułatwienie: wprowadzili **zmienne współrzędnych,** częściej nazywane **zmiennymi podwójnymi**. Zmienna podwójna to para trwale związanych zmiennych, którymi można posługiwać się tak jak jedną zmienną. Zmienne podwójne znajdują się zawsze w ostatnim wierszu banku zarówno w szufladach, jak i w koszykach. W każdym banku zmiennych są dwie zmienne podwójne nazwane **XY** oraz **xy** – patrz rysunek 14.19. Zmienna podwójna odznaczona wielkimi literami służy do zapamiętania współrzędnych pola, zmienna podwójna odznaczona małymi literami służy do przechowywania współrzędnych piksela.



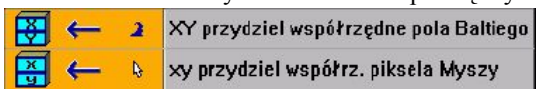
Rysunek 14.19
Zmienne podwójne



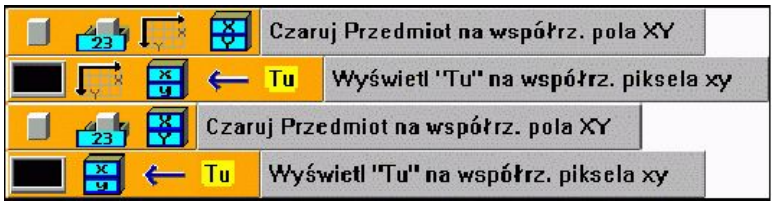
Rysunek 14.20
Składniki zmiennych podwójnych

Często zachodzi potrzeba skorzystania z pojedynczych składników zmiennej podwójnej, na przykład tylko ze współrzędnej X. Dla naszej wygody w ostatnim wierszu banku umieszczone są elementy zastępujące te składniki. W bankach zmiennych znajdziemy je po lewej stronie ostatniego wiersza – patrz rysunek 14.20.

Zmiennym podwójnym można przydzielać współrzędne jednym poleceniem (patrz rysunek 14.21). Można czarować lub wyświetlać na współrzędnych zapamiętanych w zmiennej podwójnej (patrz rysunek 14.22). Ponieważ Baltie zakłada, że zmienne podwójne zawierają współrzędne, nie musimy nawet używać elementów współrzędnych i skrócić program do postaci widocznej w dwóch ostatnich poleceniach na rysunku.



Rysunek 14.21
Przypisanie wartości współrzędnych do zmiennych podwójnych



Rysunek 14.22
Wyświetlanie i czarowanie na współrzędnych zapamiętanych w zmiennej podwójnej

Jak już powiedzieliśmy, pojedyncze składniki zmiennych podwójnych są dostępne za pomocą specjalnych zmiennych. Wartości można także zapamiętywać bezpośrednio w zmiennych podwójnych. Pierwsza wartość zostanie przydzielona składnikowi **x** lub **X** a druga składnikowi **y** lub **Y**. Na rysunku 14.23 znajdziemy kilka przykładów korzystania ze zmiennych współrzędnych.

W pierwszym wierszu na rysunku wartość 1 jest przydzielona składnikowi poziomemu współrzędnych pola (X). W drugim wierszu składnikowi pionowemu Y przydzielona jest dwukrotnie zwiększona wartość składnika poziomego X. Polecenie w trzecim wierszu powiększy wartość obu składników o jedynkę.

Rysunek 14.23
Przykłady operacji na zmiennych współrzędnych

Piąty wiersz pokazuje, jak niedoświadczony użytkownik mógłby próbować wyświetlić wartości współrzędnych pola. Jak powiedzieliśmy, jeżeli zmienna podwójna pojawi się na miejscu, gdzie można użyć współrzędnych, Baltie będzie traktował ją jako polecenie wprowadzenia współrzędnych. Piąty wiersz programu wyświetli zatem wartość zero (nie podano, co ma być wyświetlone) na współrzędnych ze zmiennej **XY**, tzn. na współrzędnych **(2 , 3)**.

Szósty wiersz pokazuje, jak może wyglądać polecenie wyświetlenia w czytelnej formie wartości współrzędnych ze zmiennej **XY**.

Uważajmy na to, że podczas przypisywania współrzędnych piksela do współrzędnych pola i odwrotnie następuje automatyczne przeliczenie współrzędnych piksela na współrzędne pola i odwrotnie. Jeśli np. mieliśmy współrzędne piksela **x = 5** i **y = 25**, to po przypisaniu **XY ← xy** w obu składnikach zmiennej **XY** będzie zero, bo ten piksel leży w polu (0,0). Podobnie, jeśli współrzędne pola wyniosą **X = 1** i **Y = 1,** po przypisaniu **xy ← XY** i automatycznej konwersji otrzymamy **x = 39** i **y = 29 .**

14.7 Czego nauczyliśmy się

W tym rozdziale poznaliśmy stałe i wyjaśniliśmy ich zalety w stosunku do literałów. Poznaliśmy zmienne i nauczyliśmy się używać ich w swoich programach. Wyjaśniliśmy różnicę między zmiennymi globalnymi i lokalnymi (szufladami i koszykami). Pokazaliśmy, jak można podawać parametry wywoływanym pomocnikom i jak można odbierać od nich uzyskane wartości. Na kilku przykładowych programach zapoznaliśmy się z kolejnymi zasadami, których należy przestrzegać podczas programowania.

15. Duży program i jego poprawianie

Czego nauczymy się w tym rozdziale

W tym rozdziale pokażemy na przykładzie jednego większego programu, jak postępować podczas tworzenia dużego programu. Wykorzystamy wszystkie wiadomości i nauczymy się nowych umiejętności, które ułatwią nam tworzenie programów. Będziemy pracować coraz lepiej i szybciej.

Wiemy już stosunkowo dużo o programowaniu. Najwyższy czas wykorzystać posiadaną wiedzę. Wróćmy do Baltiego, który sadził kwiaty na wyspie w basenie. Spróbujmy poprawić program tak, by móc regulować wielkość basenu i odległość wyspy od brzegu poprzez ustawienie wartości wybranych stałych. Ponadto nie pozwolimy Baltiemu obchodzić całego basenu lecz zmusimy go, by czarował kwiaty na całej wyspie podczas przejścia wzdłuż południowej strony wyspy. Program będzie jeszcze efektowniejszy, jeśli Baltie nie będzie czarował całych, wyrośniętych kwiatów. Zaczaruje tylko miejsce, w którym potem kwiat wyrośnie.

Ponieważ mamy już teraz program stosunkowo skomplikowany, nie będziemy nikogo zmuszać do samodzielnego konstruowania, choć byłoby to wskazane. Obejrzyjmy teraz wspólnie proponowane rozwiązanie. Przy okazji zwrócimy uwagę na wprowadzenie i przestrzeganie następnych zasad oraz pokażemy kilka przydatnych sztuczek, które ułatwią nam poprawianie programów.

15.1 Pomocnicza procedura wypisywania komunikatów kontrolnych

Zanim zaczniemy myśleć o konstrukcji naszego nowego programu, zacznijmy od jednego bardzo przydatnego dodatku w programie. Chodzi o specjalnego pomocnika, którego zadaniem będzie wyświetlić we wskazanym miejscu ekranu zadany tekst. Pomocnika można oznaczyć np. dzwonkiem i tak go nazwać. W naszym nowym programie optymalnym miejscem do wyświetlania komunikatów kontrolnych będzie górna krawędź dworku Baltiego. Tam będzie tekst najmniej przeszkadzać.

Jeśli popatrzymy na program z rysunku 15.1 zobaczymy, że **Dzwonek** najpierw wyświetli kilku odstępów – właśnie tyle, by zapisać wiersz aż do końca. Tak usunie poprzedni komunikat, więc nie będą się nam mieszać kolejne komunikaty. Przekonać się o potrzebie takiego rozwiązania można próbując wyświetlić w tym samym miejscu ekranu najpierw długi tekst a potem krótszy. Nasz pomocnik na oczyszczone spacjami miejsce wpisze tekst przechowywany w koszyku **Początek**, piknie i poczeka na naciśnięcie klawisza lub przycisku myszy.

Rysunek 15.1
Procedura pomocnicza do wypisywania komunikatów kontrolnych

Jeżeli nie jesteśmy pewni, co dzieje się w pewnej części programu, szukamy w niej punktu krytycznego i wstawiamy tam wywołanie pomocnika **Dzwonek.** Tekst, który podamy jako wartość początkową jego parametru, zawsze zaczyna się od tekstu, po którym poznamy, w którym miejscu programu się znajdujemy. Można też dołączyć do podawanego ciągu znaków wartości niektórych kluczowych zmiennych, co może ułatwić odszukanie błędu – patrz rysunek 15.5.

Za każdym razem, kiedy program wywoła **Dzwonek,** wypisze podany tekst na ekranie, piknie i poczeka, aż go przeczytamy i naciśnięciem klawisza „popchniemy" program. Pozwala to na obserwowanie kolejnych działań programu. Znalezienie błędu jest potem znacznie łatwiejsze.

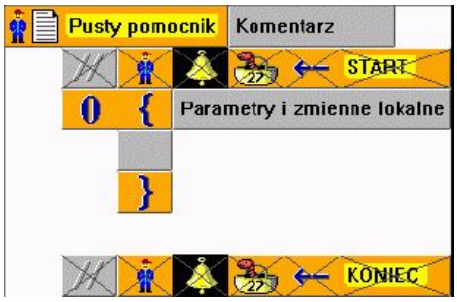
Przeważnie nie usuwamy komunikatów kontrolnych z programu po znalezieniu i usunięciu błędu. Wygodniej jest zrobić z tego polecenia komentarz. Często zdarza się, że po jakimś czasie szukamy innego błędu i wtedy korzystnie mieć gotowe polecenia dla punktów kontrolnych.

15.2 Pusty pomocnik

Wywołanie pomocnika **Dzwonek** można umieścić na początku i końcu każdego nowego pomocnika. Parametrowi **Początek** przydzielamy ciąg znaków z nazwą definiowanego pomocnika, przed które w wywołaniu początkowym wprowadzimy słowo START a przy wywołaniu końcowym słowo KONIEC (patrz rysunek 15.2). Dodamy też w razie potrzeby wypisywanie wartości parametrów. Wywołania pomocnika **Dzwonek** na końcu definiowanego pomocnika wygodnie jest potem użyć zamiast komentarza końcowego, o którym mówiliśmy w poprzednim rozdziale.

Po wykorzystaniu komunikatów kontrolnych maskujemy wywołanie **Dzwonka** za pomocą komentarza wierszowego. W razie wątpliwości, czy rzeczywiście pomocnik jest wywoływany wystarczy usunąć komentarz i od razu wszystko będzie jasne. **Dzwonek** zawsze zgłosi, że pomocnik przejmuje kontrolę w programie. Zasygnalizuje też koniec działania pomocnika.

Tworzenie tych „obowiązkowych" wierszy można uprościć przygotowując **pustego pomocnika** (patrz rysunek 15.2). Jest to pomocnik, który nic nie robi, ale jego forma odpowiada ustalonym konwencjom: za oznaczeniem następuje komentarz, treść pomocnika zaczyna się od komunikatu kontrolnego, pomocnik ma przygotowany blok do opisania parametrów i zmiennych lokalnych, cały pomocnik zakończony jest standardowym komunikatem kontrolnym. Nazwy pomocnika w wypisach kontrolnych jeszcze nie ma, wkopiujemy je dopiero wtedy, gdy nadamy nazwę nowemu pomocnikowi.



Rysunek 15.2
Pusty pomocnik

W ten sposób mamy zbudowany „szkielet" przyszłych pomocników. Zbędne fragment pomocnika łatwo jest usunąć w dowolnym momencie. W pomocnikach bez zmiennych lokalnych i parametrów usuwamy blok deklaracji, przy bardzo krótkich pomocnikach zbudowanych z jednego lub dwóch wierszy czasami nie używamy nawet wypisów kontrolnych.

Tworząc nowego pomocnika kopiujemy w jego miejsce pomocnika pustego, dodajemy parę tekstów i możemy skoncentrować się na treści pomocnika mając zachowane konwencje konstrukcji. Każdy pomocnik będzie zbudowany według tego samego schematu.

15.3 Śledzenie wartości zmiennych

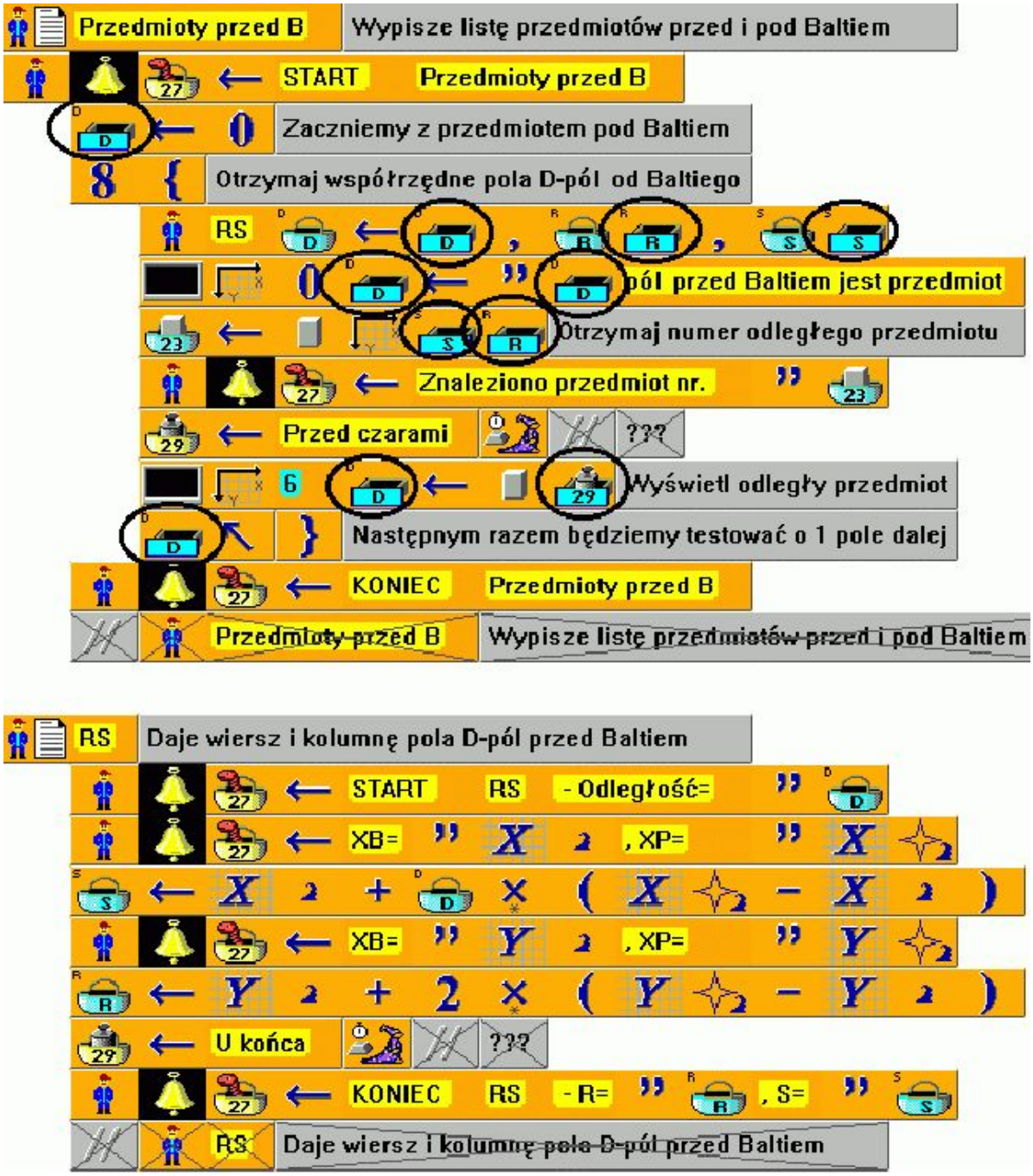
Pomocnik **Dzwonek** potrafi zatrzymać program w miejscu, gdzie potrzebujemy popatrzeć na tymczasowy wynik i może pooglądać wartości niektórych zmiennych. Nie potrzeba jednak wkładać wszystkich wartości do napisu wysyłanego do dzwonka. Baltie pozwala zobaczyć wartości zmiennych w dowolnym momencie podczas pracy programu. Wystarczy nacisnąć przycisk **Pokaż zmienne**, który znajduje w szarej krawędzi pod lewym dolnym rogiem dworku Baltiego – patrz rysunek 15.3.

Rysunek 15.3
Jak pokazać zmienne

Po naciśnięciu tego przycisku rozwinie się on w trzy przyciski, wyświetlone jako naciśnięte (patrz rysunek 15.4). Ponownym naciśnięciem dużego przycisku, który nacisnęliśmy na początku, ukryjemy wszystkie zmienne. Naciskając zamiast dużego przycisku jeden z mniejszych przycisków ukryjemy tylko albo szuflady, albo koszyki w zależności od tego, który przycisk nacisnęliśmy.

Wypróbujmy teraz to wszystko w naszym ostatnim programie testującym przedmioty przed Baltiem. Program pokazaliśmy na rysunku 14.18. Wprowadzimy kilka drobnych zmian. Różnicę między pokazywaniem zawartości szuflad i koszyków zobaczymy po zastąpieniu koszyków szufladami w pomocniku **Przedmioty przed B**. Poza tym dodamy do niego kilka komunikatów kontrolnych. Komunikaty kontrolne dodamy też do pomocnika **RS.** Wprowadzimy ponadto do programu mały błąd, z powodu którego nie wyświetlają się przedmioty. To uzasadni użycie punktów kontrolnych.

Rysunek 15.4
Jak ukryć zmienne



Rysunek 15.5
Poprawianie programu, pokazującego przedmioty przed Baltiem

Zmieniony program można zobaczyć na rysunku 15.5. Dla uproszczenia pracy nad zmianami odznaczono wszystkie zmiany zmiennych lokalnych na globalne w pomocniku **Przedmioty przed B**. Łatwo odszukać i uzupełnić komunikaty kontrolne.

Zauważmy, że używamy dwóch rodzajów punktów kontrolnych:

– wywołanie pomocnika **Dzwonek**, któremu podajemy ciąg znaków do wyświetlania pod górną krawędzią dworku,

– wstrzymanie biegu programu (wiersze kończące się komentarzem z trzema znakami zapytania).

Każde z tych rozwiązań ma wady i zalety. Przeglądnijmy je po kolei. Korzyścią z wywołania pomocnika **Dzwonek** jest to, że możemy wyświetlić nie tylko wartość wybranych zmiennych ale i wartości niektórych wyrażeń. Na przykład w pomocniku **RS** z rysunku 15.5 wypisujemy współrzędne Baltiego i współrzędne pola przed Baltiem. Wadą takiego rozwiązania jest wypisywanie komunikatów przesłaniających część dworku Baltiego. Nie można też zobaczyć koszyków pomocnika, który zawołał pomocnika **Dzwonek**, bo **Dzwonek** o tych koszykach nie wie, więc nie wyświetli ich zawartość w szarej przestrzeni pod dworkiem – znajdziemy tam tylko koszyki **Dzwonka**.

Korzyścią ze wstrzymania programu jest możliwość obejrzenia wartości wszystkich koszyków pomocnika, w którym program został wstrzymany. Koszyki widzimy na dole na szarej przestrzeni okna roboczego. Wadą takiego rozwiązania jest to, że nie możemy sprawdzić wartości niektórych wyrażeń. Nie możemy np. sprawdzić, czy w formule nie ma błędu. Dodatkowo ze zmiennych napisowych zobaczymy tylko kilka pierwszych znaków – reszta nie mieści się na ograniczonej przestrzeni.

Przeglądając program z rysunku 15.5 może nasuwać się pytanie, dlaczego przed zatrzymaniem przydzielono zmiennej napisowej **Waga** (żółty koszyk z odważnikiem) tekst, który nigdy nie jest wykorzystywany i dlaczego wprowadzono za polecenie **Czekaj** element **Komentarz wierszowy,** który niczego nie przesłania. Odpowiedź jest prosta.

Tekst z koszyka pozwoli nam później rozpoznać, w którym miejscu programu zatrzymaliśmy się. Przy każdym zatrzymaniu w szarej przestrzeni znajdziemy żółtą zmienną **Waga** i według jej zawartości sprawdzimy, czy program zatrzymał się w tym miejscu, gdzie powinien. Potem sprawdzimy wartości zmienne, które nas interesują.

Element **Komentarz wierszowy** na końcu wiersza umieszczono, by w momencie, że punkt kontrolny już nie będzie potrzebny nie szukać go w panelu narzędzi lub w programie. Po prostu wiemy, że jest on na końcu wiersza i przesuniemy go stąd na początek wiersza. Takie maskowanie jest lepsze niż usuwanie elementu, bo zamaskowane polecenia mogą się jeszcze kiedyś przydać. Usuniemy je dopiero wtedy, gdy będziemy pewni, że program nie ma błędów i będziemy komuś ten program przekazywać. Oczywiście nikt nie chce pokazać, ile miał błędów w programie i jak ich szukał.

Śledzenie programu

Teraz uruchomimy zmodyfikowany program. Pamiętajmy o włączeniu pokazywanie wartości zmiennych. Pod dworkiem Baltiego pojawi się lista wartości zmiennych, podobna do listy z rysunku 15.6.

Rysunek 15.6
Wartości zmiennych

Nie będzie to ta sama lista, bo do programu dodano jeszcze kilka zmiennych. Nie mieszczą się one w ograniczonym obszarze wyświetlania, więc pod przyciskami, którymi włącza i wyłącza się pokazywanie zmiennych, pojawiły się strzałki, które można zobaczyć na rysunku. Te strzałki sygnalizują, że lista zmiennych jest za długa, nie mieści się w wydzielonej przestrzeni. Kliknięciem na strzałce na prawo lub na lewo możemy pokazać następną lub poprzednią część listy.

Na naszym rysunku program jest zatrzymany w pomocniku **RS** w miejscu, gdzie zmiennej **Waga** przydzielamy wartość **U końca.** Zwróćmy uwagę na to, że nazwy zmiennych **D**, **R**, **S** występują tu dwukrotnie. Przy dłuższej obserwacji można zauważyć, że w pierwszym wystąpieniu nazwa każdej zmiennej jest wyświetlona normalnie a w drugim wystąpieniu jest wyświetlona kursywą. Pierwsza grupa nazw oznacza zmienne globalne (szuflady), druga zmienne lokalne (koszyki).

Konstrukcja naszego programu może być myląca. Koszyki pomocnika **RS** nazywają się tak samo jak szuflady, które, jak pamiętamy, są widoczne w całym programie. Poza tym szuflady **R** i **S** pobierają wartości końcowe z koszyków z tymi samymi nazwami (koszyki **R** i **S** są parametrami wyjścia pomocnika **RS**) i mają ciągle wartość zgodną z tymi koszykami. Może to sprawiać wrażenie, że chodzi o te same zmienne. Ale to nie jest prawda.

Spróbujmy teraz przejść przez cały program i znaleźć w nim błędy. Swoje wnioski możemy porównać z informacjami zawartymi w następnym akapicie.

Błędy znalezione? Program działa? Miejmy nadzieję, że tak. Jeżeli nie, dodatkowa podpowiedź: sprawdźmy w pomocniku **Przedmioty przed B** czy wszędzie użyto poprawnych zmiennych.

W programie są trzy błędy:

- gdy dostajemy numer odległego przedmiotu, nie wkładamy otrzymanej wartości do szuflady **Przedmiot** (tak byłoby poprawnie), lecz do koszyka **Przedmiot** (a to niepoprawnie),
- od razu w następnym wierszu, gdzie wypisujemy numer znalezionego przedmiotu, użyto ponownie koszyka zamiast szuflady,
- żeby jeszcze pogorszyć sytuację, w wierszu, w którym wyświetlany jest odległy przedmiot, użyto zamiast szuflady **Przedmiot** szuflady **Waga.**

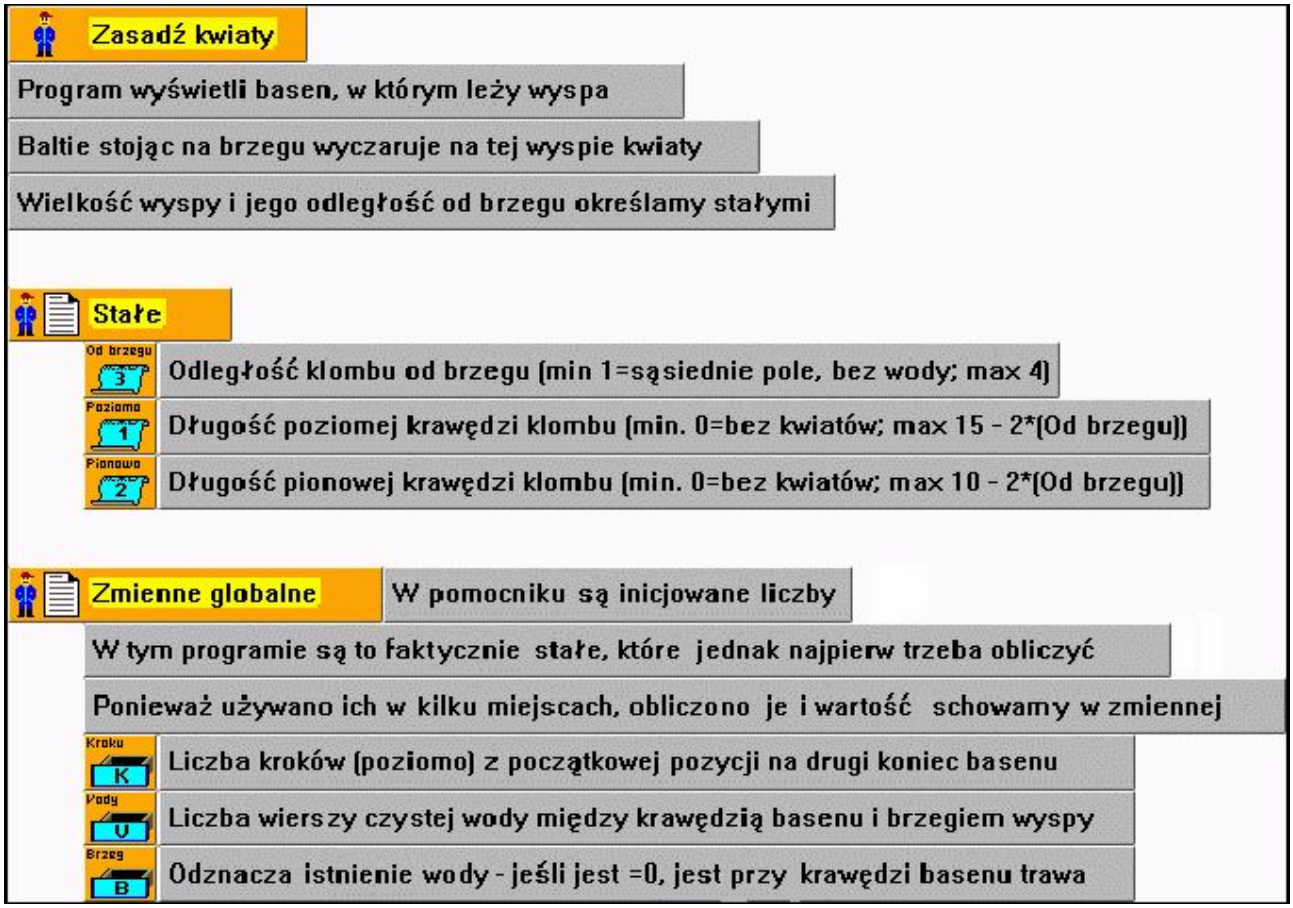
Ciągłe wyświetlanie wartości zmiennych bardzo spowalnia program. Prośbą o wyświetlenie wartości zmiennych dodajemy Baltiemu mnóstwo roboty. Jeśli pokazywanie jest włączone Baltie powinien oprócz wykonywania programu ciągle śledzić wartości wszystkich zmiennych i po wykonaniu każdego polecenia wszystkie je wyświetlać. Nie zostawiajmy więc włączonego pokazywania wartości zmiennych bez potrzeby.

15.4 Zmienne globalne i stałe

Najwyższy czas zacząć programować. Jeszcze chwila cierpliwości: spróbujemy wyjaśnić korzyści przestrzegania następnej zasady, która pomoże zorientować się w naszych programach nawet po kilku latach.

Warto przyzwyczaić się do wprowadzania za wywołaniem programu głównego fikcyjnych pomocników, w których wypiszemy wszystkie użyte stałe i zmienne globalne (szuflady) podobnie, jak to robiliśmy w przypadku koszyków pomocników. Tych pomocników nazywamy fikcyjnymi bo rzeczywiście nie są żadnymi pomocnikami. Nigdy ich nie wywołamy. Użyjemy ich w momencie, gdy zapomnimy, co oznacza jakaś stała lub zmienna. W takiej chwili za pomocą tablicy pomocników szybko przesuniemy się w miejsce z opisem odpowiedniej stałej lub zmiennej a potem znów równie szybko wrócimy w poprzednie miejsce programu.

Możemy wybrać inny sposób zapisu ale na pewno warto wprowadzać na początku programu pewną formę opisania poszczególnych obiektów globalnych. Przykładową wersję dla programu z wyspą w basenie mamy na rysunku 15.7.



Rysunek 15.7
Opisanie działania programu, stałych i zmiennych globalnych

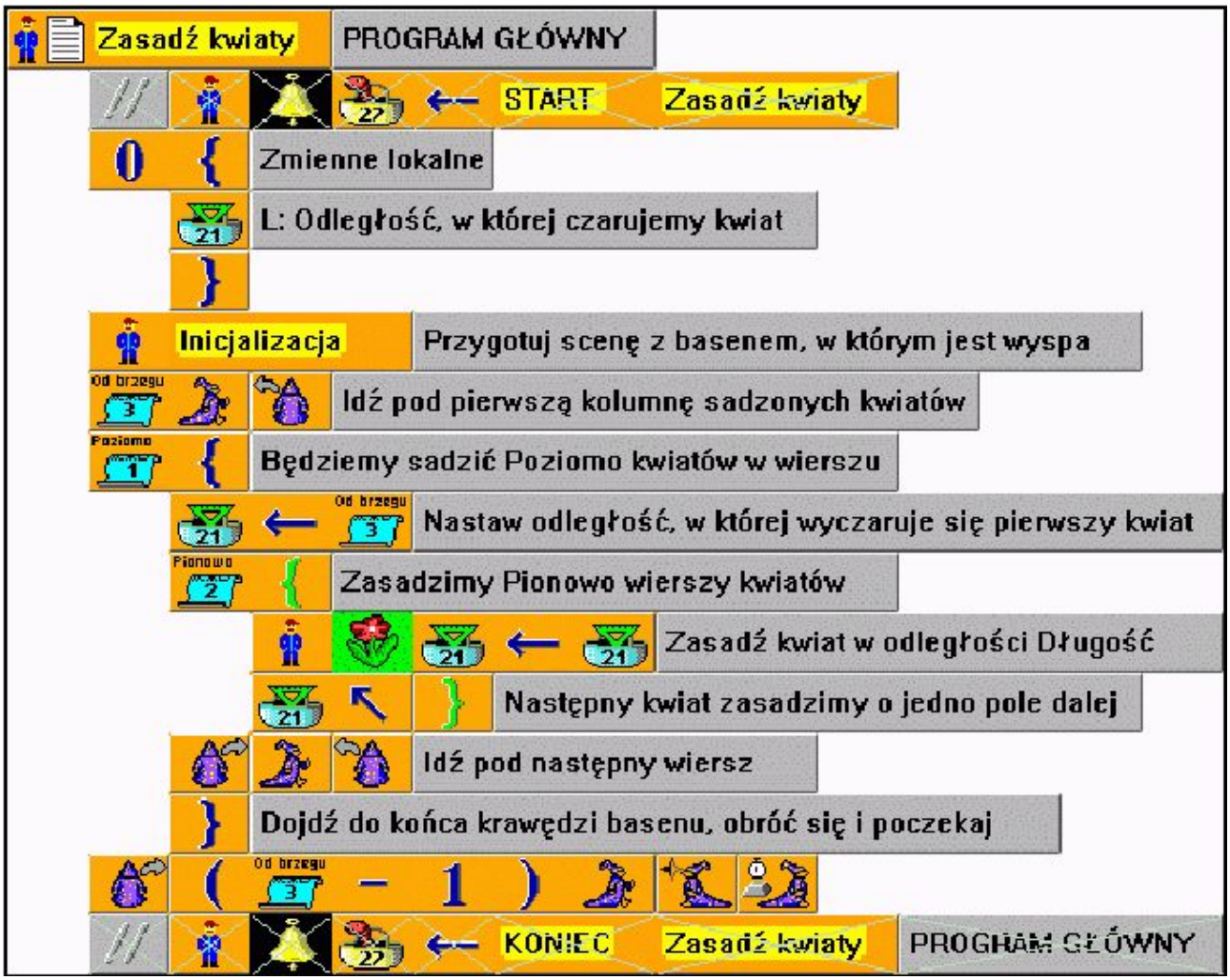
15.5 Analizujemy i programujemy zadanie

Nareszcie zajmiemy się samym programem. Zapoznaliśmy się z zawartością rozdziałów o dekompozycji i zasadach poprawnego projektowania programu. Nie będziemy już zatem pokazywać projektów poszczególnych pomocników, tylko nieco o nich opowiemy i zostawimy do samodzielnego oprogramowania. Swoje rozwiązanie można potem porównać z ostatecznym rozwiązaniem, które spróbujemy szczegółowo skomentować. Komentarze te nie będą zbyt szczegółowe. Chcemy teraz pokazać sposób postępowania przy rozwiązywaniu trudniejszych zadań, gdyż to pozwoli na nabranie praktyki przydatnej przy rozwiązywaniu przyszłych zadań.

Gdyby opisywany program wydawał się zbyt trudny lub opis okazał się nudny, spokojnie można pominąć resztę rozdziału. Byłoby jednak dobrze wrócić do niego w przyszłości i porównać opisany tu sposób projektowania ze swoim.

Jak mogliśmy zobaczyć na rysunku 15.7, nasz program główny nazywa się **Zasadź kwiaty.** Jego zadaniem jest wyświetlić basen z wyspą, przejść z Baltiem wzdłuż jego południowego brzegu i po drodze zaczarować niektóre miejsca na wyspie tak, by na nich potem wyrosły kwiaty.

Zachęcamy teraz do samodzielnego zaprojektowania głównego pomocnika. Rozwiązanie można porównać z wersją pokazaną na rysunku 15.8.



Rysunek 15.8
Główny pomocnik w programie sadzenia kwiatów na wyspie

Zanim zaczniemy z przygotowaniem operacji sadzenia kwiatów musimy wyczarować basen z wyspą. Tym zajmie się pomocnik nazwany **Inicjalizacja.** Powiedzmy sobie wyraźnie, że zbudowanie basenu to nie jest łatwe zadanie. W zależności od odległości wyspy od krawędzi basenu może to być:

– basen całkiem bez wody (odległość 1),

– może mieć tuż przy krawędzi basenu brzeg wyspy (odległość 2),

– może być między krawędzią basenu a brzegiem wyspy miejsce na jeden do dwóch przedmiotów tworzących czystą toń wodną.

Zastanowimy się nad tym zanim wszystko zaprogramujemy. Przypuśćmy na razie, że basen jest już zbudowany. Dalej jest już wszystko stosunkowo łatwe. Baltie najpierw podejdzie do pierwszej kolumny, gdzie można zasadzić kwiaty i po kolei zasadzi kwiaty w odpowiedniej kolumnie (w rzędzie). Po posadzeniu rzędu kwiatów przejdzie do następnej kolumny i tak dalej, dopóki nie posadzi kwiatów we wszystkich **Poziomo** kolumnach i w ten sposób wypełni całą wyspę.

Bezpośrednio do zasadzenia kwiatu zaprosi pomocnika **Kwiat,** którego jedynym parametrem jest odległość sadzonego kwiatu od Baltiego. Pomocnik będzie po kolei sadzić kwiaty w coraz większych odległościach zanim nie posadzi wszystkich kwiatów w swoich **Pionowo** wierszach.

Do wypróbowania pomocnika **Zasadź kwiaty** potrzebujemy działających pomocników **Inicjalizacja** i **Kwiat**. Jak powiedzieliśmy jedyną możliwością jest przygotowanie delegatów. Delegata pomocnika **Inicjalizacja** polecamy zaprogramować tak, że zamiast skomplikowanego budowania basenu otworzy scenę, w której sami wcześniej

przygotujemy basen według wprowadzonych parametrów. Delegat pomocnika **Kwiat** mógłby przed Baltiem wyświetlić liczbę określającą odległość, w której przyszły pomocnik **Kwiat** będzie sadzić kwiaty. Druga możliwość to użycie odpowiednio dostosowanego pomocnika stworzonego podczas konstruowania poprzedniej wersji tego programu.

Po sprawdzeniu działania pomocnika **Zasadź kwiaty** powinniśmy przejść do jednego z pomocników, których przed chwilą zastąpiliśmy delegatami. Ponieważ delegat pomocnika **Inicjalizacja** bardzo dobrze wykonuje swoje zadanie a pomocnik **Kwiat** wydaje się prostszy, zaczniemy od sadzenia kwiatów.

Zadaniem pomocnika **Kwiat** jest zaczarować miejsce w określonej odległości od Baltiego tak, by wyrósł na nim kwiat. To jest zadanie podobne do już rozwiązanego: musimy na poszczególnych miejscach wyświetlać kolejno przedmioty powiększającego się kwiatu tak, by widzowie odnieśli wrażenie, że kwiat rośnie.

Z zadania wynika, że w odpowiednim miejscu powinniśmy wyczarować kolejno kilka przedmiotów. Jedną z możliwości rozwiązania zadania jest przygotowanie pomocnika (nazwijmy go **Czaruj w odległości**), który potrafi wyczarować określony przedmiot w określonej odległości od Baltiego. Polecimy temu pomocnikowi, by czarował poszczególne fazy rośnięcia kwiatu na określonej pozycji.

Delegata pomocnika **Czaruj w odległości** dla nie trzeba tworzyć, bo będzie on tylko małą modyfikacją pomocnika, którego zbudowaliśmy już wcześniej. Możemy go więc napisać od razu.

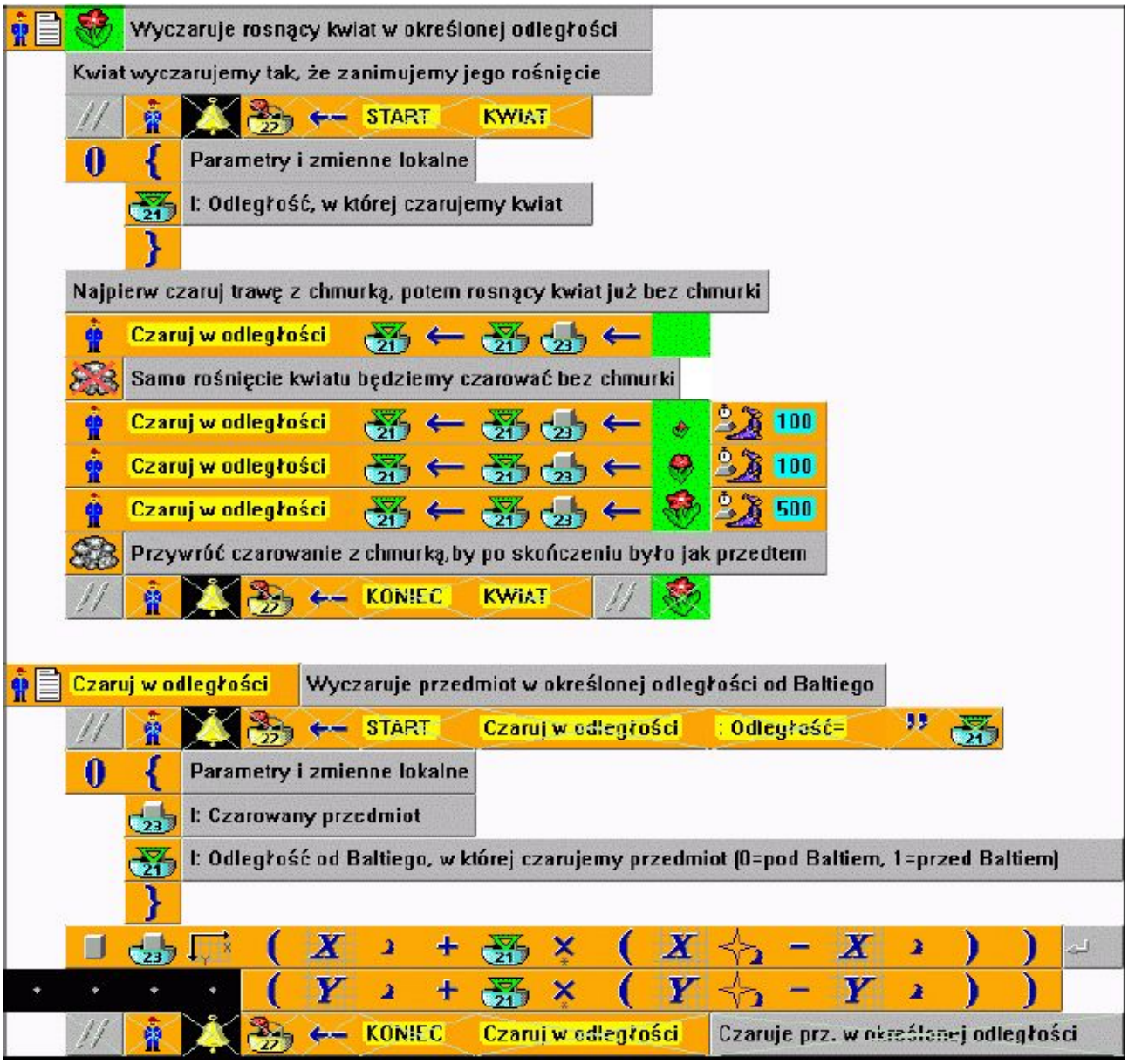
Podczas sprawdzania magicznego rośnięcia kwiatów na pewno zauważymy, że wybuchy chmurek pomiędzy poszczególnymi fazami pojawiania się kwiatu przeszkadzają w obserwacji i byłoby lepiej, by chmura „czaru" pojawiła się tylko podczas pierwszego zaczarowania trawy, Kwiat powinien potem rosnąć bez chmurek. Końcowy wygląd działającej pary pomocników można zobaczyć na rysunku 15.9

Teraz czeka nas omijany dotąd pomocnik **Inicjalizacja.** Jego zadaniem jest przygotować basen z wyspą. Jak już powiedziano (w związku z różnymi możliwościami wyglądu basenu) nie jest to zadanie proste.

Spróbujemy budować basen w kolejnych wierszach i to od dolnej krawędzi. Najpierw przygotujemy dolną krawędź, potem może kolejny wiersz z przedmiotami „woda", potem wnętrze wyspy i znów brzeg, wodę i krawędź basenu. Dla przypomnienia jeden z możliwych wyglądów basenu można zobaczyć na rysunku 12.17.

Analizując wygląd basenu z wyspą można zauważyć, że potrzebujemy paru innych informacji. Po pierwsze musimy wiedzieć, ile wierszy czystej wody będzie (jeśli w ogóle będą) między krawędzią basenu i brzegiem wyspy. Po drugie, czy wyspa będzie miała w ogóle jakiś brzeg. Gdyby odległość od krawędzi basenu do wyspy wynosiła jeden, kwiaty byłyby sadzone przy samej krawędzi basenu. Nie chodziło zatem o żadną wyspę lecz o obmurowany klomb. Żeby z kolei po zbudowaniu kolejnego wiersza basenu móc wrócić do punktu wyjścia przydałoby się wiedzieć, ile kroków prowadzi z jednej krawędzi basenu na drugą.

Zdecydowaliśmy się zatem na wprowadzenie następnego pomocnika, **Liczby**, którego jedynym zadaniem będzie otrzymać potrzebne informacje. Przechowa on liczbę wierszy wody w zmiennej **Wody,** liczbę kroków z jednej krawędzi basenu na drugą w zmiennej **Kroków** i informację o tym, czy będzie wyspa mieć brzeg, w zmiennej **Brzeg.**



Rysunek 15.9
Definicja pomocników Kwiat i Czaruj w odległości

Zadecydowaliśmy się poza tym przygotować pomocnika **Krawędź**, który zbuduje krawędź basenu i pomocnika **Woda,** który przygotuje odpowiednią liczbę wierszy wody. Dzięki temu pomocnik **inicjalizacja** będzie czytelniejszy.

Gdy popatrzymy na wyspę, zauważymy, że zarówno dolny i górny brzeg, jak i środek wyspy buduje się bardzo podobnie: zawsze trzeba narysować krawędź basenu, kilku pól wody, przedmiot brzegu, szereg przedmiotów środkowych, drugi przedmiot brzegu, znów może być czysta woda i na koniec krawędź basenu.

Warto więc stworzyć pomocnika, któremu tylko podamy, jakie przedmioty są krawędzią, jaki jest środkowy i kilka razy po kolei zawołamy tego pomocnika w programie.

Końcowy wygląd pomocnika **Inicjalizacja** można zobaczyć na rysunku 15.10.

Rysunek 15.10
Definicja pomocnika Inicjalizacja

Zanim utworzyć innych pomocników najpierw pomocnik **Liczby** powinien obliczyć niektóre ważne wartości. Zajmijmy się najpierw nim. Najłatwiej obliczyć liczbę wierszy czystej wody między krawędzią basenu i brzegiem wyspy. Liczba ta będzie o dwa mniejsza od odległości pierwszego kwiatu. Ponieważ wiemy, że najmniejsza odległość kwiatu od krawędzi basenu wynosi jeden, może zdarzyć się, że obliczona liczba wierszy czystej wody będzie ujemna. To jednak nie musi nam przeszkadzać. Jeśli powiemy, że coś ma się wykonać „minus jeden raz”, to tak samo, jakbyśmy powiedzieli, że ma się to wykonać zero razy.

Równie łatwo można otrzymać liczbę kroków do drugiej krawędzi basenu. Można ustalić ją na podstawie wielkości basenu. Wielkość basenu powinna być równa liczbie zasadzonych już kolumn kwiatów powiększonej o dwa razy liczbę elementów od krawędzi basenu do pierwszego kwiatu. Liczba kroków będzie o dwa mniejsza. Jeden odejmiemy dlatego, że Baltie już stoi na pierwszym polu, drugi odejmiemy dlatego, że nie prowadzimy Baltiego wstecz z drugiego końca basenu, ale z miejsca, z którego będzie czarować przedmiot krawędzi basenu, a to pole jest o jeden krok bliżej do początkowej.

Najtrudniejszym zadaniem jest wymyślenie formuły obliczającej, czy w basenie rzeczywiście będzie wyspa i czy będziemy zatem rysować brzeg. Potrzebujemy, by wartość tej zmiennej wynosiła zero w przypadku, gdy odległość od krawędzi basenu do pierwszego kwiatu wynosi jeden (brzeg będzie rysowany zero razy) oraz by wynosiła jeden w przypadku, gdy ta odległość jest większa.

W następnym rozdziale dowiemy się, jak ten problem rozwiązać bez matematyki. Jednak w programowaniu pojawiają się takie zadania stosunkowo często i chcemy pokazać sposób ich rozwiązywania, który czasami jest lepszy od metody wyboru, o której będzie mowa w następnym rozdziale.

Rysunek 15.11
Definicja pomocnika Liczby

Jak sprawić, by dla liczby **1** wartość formuły wynosiła zero a dla następnych trzech jeden? Można skorzystać z odcinania części dziesiętnej i wymyślić formułę, której wynik jest dla liczby **1** mniejszy od jeden a dla reszty wartości wynik jest gdzieś między jedynką a dwójką. Jak można przeczytać w programie na rysunku 15.11, zdecydowaliśmy się na formułę **(Od_brzegu + 1) / 3**, gdzie trójka w mianowniku jest liczbą rzeczywistą (zieloną!), więc wynikiem jest liczba rzeczywista:

(1 + 1) / 3 = 0,666

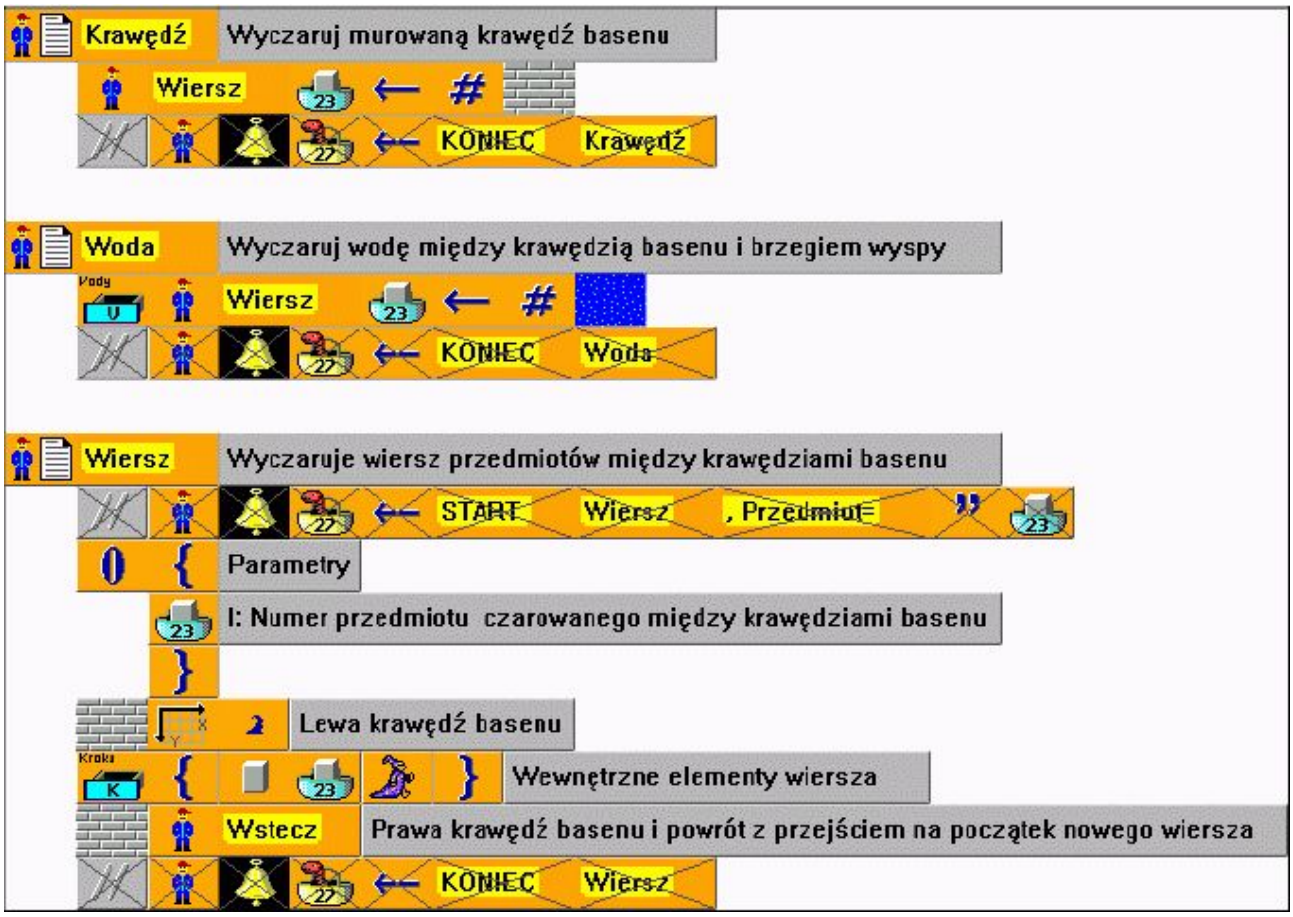
(2 + 1) / 3 = 1,000

(3 + 1) / 3 = 1,333

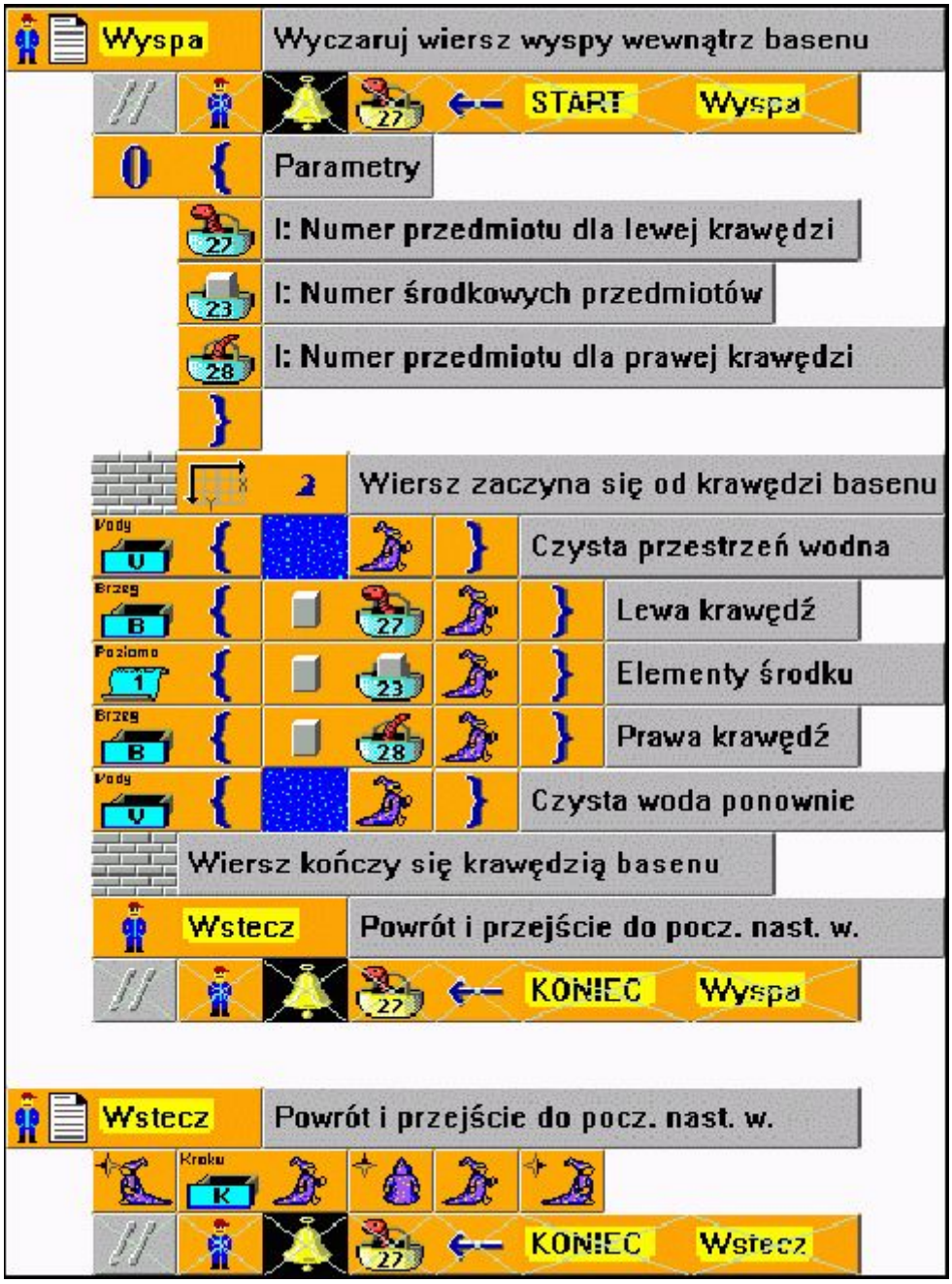
(4 + 1) / 3 = 1,666

Gdy odetniemy od tych wyników część dziesiętną (funkcja **int**), otrzymamy odpowiedni wynik.

Budowy następnych pomocników nie będziemy już wyjaśniać. Są już na tyle prosto zbudowani i szczegółowo skomentowani, że łatwo pojąć ich działanie bez dodatkowego komentarza. Definicje pomocników można zobaczyć na rysunkach 15.12 i 15.13.



Rysunek 15.12
Reszta pomocników – część pierwsza: Krawędź, Woda, Wiersz



Rysunek 15.13
Reszta pomocników – część druga: Wyspa, Wstecz

Zadanie samodzielne

Dostosujmy teraz wyżej opisany program tak, byśmy nie musieli wprowadzać parametrów basenu za pomocą stałych, lecz by te wartości program losowo sam wybierał na początku.

15.6 Pusty program

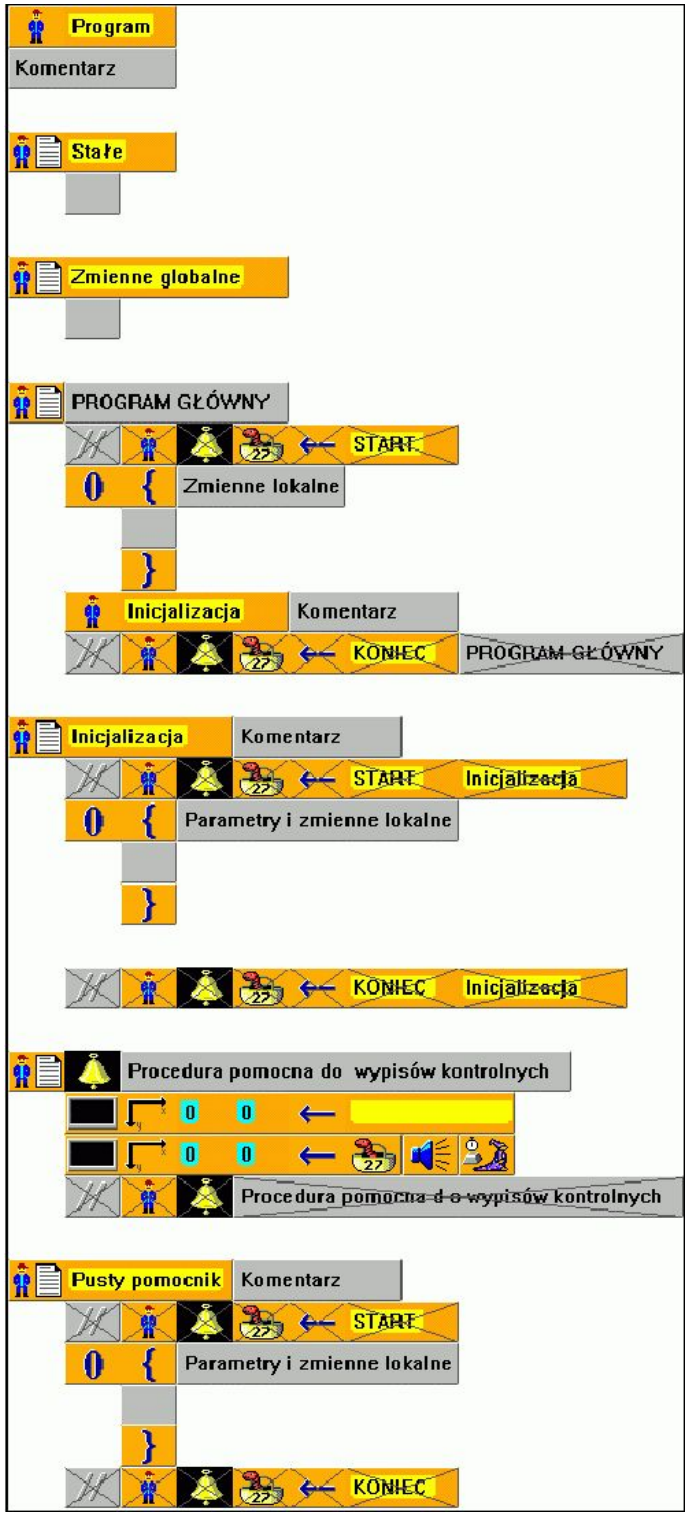
Przed opisaniem poprzedniego programu opowiadaliśmy o różnych zasadach i o przydatnych pomocnikach. Ponieważ jednak nie wiemy, czy będziemy takich pomocników potrzebować w każdym programie i niewygodnie jest kopiować pomocników za każdym razem do nowego programu, przygotujemy pusty program, którego będziemy używać

jako szablonu dla każdego nowego programu. Teraz konstruując nowy program będziemy po prostu otwierać program pusty i natychmiast zapisywać go pod nową nazwą. W ten sposób od razu będziemy mieć gotowe wszystkie potrzebne części (pomocnicy) i nie musimy zajmować się kopiowaniem z innych programów.

Jak można sprawdzić na rysunku 15.14, w programie jest przygotowane wywołanie pomocnika programu głównego, fałszywi pomocnicy opisujący deklaracje stałych i zmiennych globalnych, pusty program główny oraz pomocnik inicjalizacji, który znajduje się już praktycznie we wszystkich naszych programach. Na końcu czekają przygotowani pomocnicy **Dzwonek** do wypisywania komunikatów kontrolnych i **Pusty pomocnik** jako szablon do tworzenia nowych pomocników.

Wszyscy pomocnicy przestrzegają opisanych wcześniej konwencji i mają przygotowane miejsce na kolejne informacje, np. deklaracje parametrów czy zmiennych lokalnych.

Łatwo zauważyć, ile pracy może nam zaoszczędzić taki pusty program. Warto przygotować tego typu wzorcowy, pusty program, który będzie pasować do naszych własnych konwencji zapisu programu.



Rysunek 15.14
Pusty program zbudowany zgodnie z zasadami

15.7 Czego nauczyliśmy się

W tym rozdziale na przykładzie trochę większego programu pokazaliśmy, jak postępować podczas tworzenia takiego programu. Konstruując pomocnika **Dzwonek** pokazaliśmy, jak można przygotować pomocnika ułatwiającego nam analizowanie i śledzenie naszego programu. Przygotowaliśmy też pustą definicję pomocnika, co pozwoliło nam łatwiej i wygodniej przestrzegać ustalonych zasad zapisywania programu. Okazało się, że pusty pomocnik przyspieszył budowanie programu. Potem skoncentrowaliśmy się na analizie programu rozwiązującego postawione zadanie. Pokazaliśmy że pozornie prosty problem trzeba czasami rozwiązywać w bardziej złożony sposób niż to się na wstępie wydawało. Na koniec pokazaliśmy pusty program, którego można używać jako szablonu do tworzenia nowych programów.

16. Uczymy Baltiego jak się zatrzymać

Czego nauczymy się w tym rozdziale

W tym rozdziale będziemy opowiadać o pętlach. Jak dotąd używaliśmy w programach tylko pętli z określoną liczbą powtórzeń oraz pętli nieskończonych. Teraz powiemy o pętlach, których wykonywanie zależy od spełnienia pewnego warunku.

Od samego początku w tej książce uczymy się kolejnych metod tworzenia programów, które potrafiłyby jak najlepiej realizować postawione zadania. Wyobraźmy sobie teraz że chcemy, by Baltie znalazł wyjście z labiryntu. Do rozwiązania takiego problemu nie wystarczą nam posiadane umiejętności. Musimy nauczyć Baltiego podejmować decyzje na podstawie informacji, które otrzyma w ostatniej chwili.

16.1 Pętla warunkowa

Rozpocznijmy od prostego zadania: Baltie zbuduje obmurowany ogródek i potem chodzi w nim w kółko pilnując, by nikt nie zabrał niczego z ogródka. Powinniśmy przygotować program tak żeby Baltie był w stanie chodzić nawet po zupełnie nieznanym mu terenie. Powinno to być możliwe nawet w przypadku, gdy na początku programu nie wiemy, jak jest wielki jest pilnowany teren.

Gdybyśmy mieli taki program dla chodzenia po ogródku opisać słowami, pewnie powiedzielibyśmy Baltiemu tak:

1. sprawdź czy jest przed tobą murek
2. jeśli jest murek, idź o jeden krok dalej, jeśli muru nie ma, zrób „w lewo zwrot”
3. przejdź do punktu 1.

Dokładnie tak działa program z rysunku 16.1. Tym razem najpierw pokażemy, jak program działa a potem nauczymy się jak go zbudować.

Pierwszych trzech wierszy nie będziemy wyjaśniać. Mają one zbudować losowo wielki ogródek i umieścić w nim Baltiego. Powinno to być proste do zrozumienia i zaprogramowania.

Rysunek 16.1
Chodź w losowo wielkim ogródku

Pętla z warunkiem początkowym

W drugiej części programu interesuje nas treść pętli nieskończonej. Zaczyna się elementem, na którym pokazany jest owal ze strzałką (pomińmy na razie nawias otwierający). Ten element określa początek **pętli z warunkiem początkowym** (programiści nazywają ją przeważnie **pętlą while** [czytaj *wajl*]), tzn. pętli, w której przed każdym wykonaniem treści pętli najpierw jest sprawdzany tzw. **warunek początkowy**. Opis warunku wstawiamy za odpowiedni element pętli. Element z owalem wraz z warunkiem początkowym tworzą **nagłówek pętli**.

Za nagłówkiem znajduje się **treść pętli**, czyli seria poleceń wykonywanych podczas każdego obrotu pętli. Tak samo jak w pętli z określoną liczbą powtórzeń treść pętli trzeba zamknąć w nawiasy blokowe (złożone) jeżeli zawiera więcej niż jedno polecenie. Nic się jednak nie stanie, gdy w nawiasy ujmiemy jedno polecenie, jak to widać na rysunku 16.1. Powszechnie stosowaną formę pętli z warunkiem początkowym można zobaczyć na rysunku 16.2.

Rysunek 16.2
Pętla z warunkiem początkowym

Podczas wykonywania tej pętli Baltie najpierw sprawdzi warunek początkowy. Jeśli warunek jest spełniony (inaczej: jego wartością jest *Prawda*), Baltie wykona treść pętli i ponownie sprawdzi warunek początkowy. Gdy warunek nie będzie spełniony (będzie miał wartość *Fałsz*), przeskoczy treść pętli i będzie kontynuować program od pierwszego polecenia za pętlą.

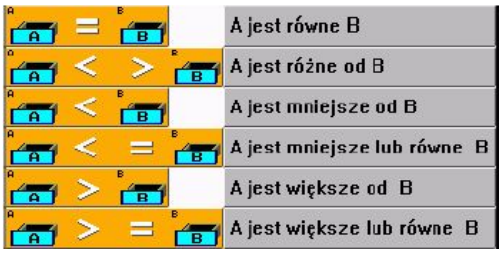
Na pewno jest jasne, że przy pętli z warunkiem początkowym może się zdarzyć, że treść pętli nie będzie wykonana ani razu. Tak będzie, gdy od razu na początku warunek początkowy okaże się niespełniony.

Warunki

W naszym programie użyliśmy najprostszego warunku, jaki zna Baltie: jeśli w warunku podamy przedmiot Baltie przypuszcza, że jest to przedmiot, który ma znajdować się przed nim. Jeśli ten przedmiot znajdzie Baltie przed sobą, traktuje warunek jako spełniony.

Podobnie prostym przypadkiem warunku jest liczba. Jeśli Baltie znajdzie w miejscu warunku liczbę, potraktuje warunek jako spełniony w przypadku gdy ta liczba jest niezerowa. Oczywiście zamiast liczby może to być wyrażenie, które zostanie przez Baltiego najpierw obliczone i rozpatrywany będzie wynik wyrażenia.

Przeważnie jednak w warunkach pętli porównujemy dwie wartości. Sposób zapisu operatorów porównania nieco różni się od zapisu, do którego jesteśmy przyzwyczajeni z matematyki. Ponieważ jeden rysunek wyjaśni więcej niż kilka akapitów tekstu, przygotowaliśmy poszczególne formy zapisu wyrażeń porównania z różnymi operatorami na rysunku 16.3.



Rysunek 16.3
Porównanie dwóch wartości

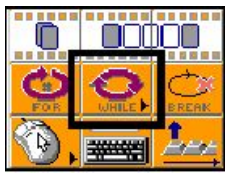
W warunku możemy porównywać dwie wartości lub dwa napisy. Jeśli chcemy porównywać inne rodzaje obiektów (np. przedmioty), powinniśmy najpierw zamienić je na liczby lub na ciągi znaków i porównywać takie obiekty. Tak samo trzeba pamiętać o konwersji gdy porównujemy napis i liczbę.

Po tych informacjach warunek początkowy pętli z programu na rysunku 16.1 będzie już zupełnie zrozumiały. Dopóki jest przed Baltiem ściana muru, warunek jest spełniony i Baltie pójdzie do przodu. Kiedy przed Baltiem nie będzie ściany, warunek nie będzie spełniony i program będzie kontynuować działanie od następnego polecenia za pętlą czyli od polecenia **W lewo zwrot**. Jest to ostatnie polecenie w treści pętli nieskończonej, po nim wracamy znów na początek pętli i ponownie uruchamiamy pętlę z warunkiem początkowym.

Wprowadzanie pętli do programu

Teraz już wiemy, jak wygląda pętla z warunkiem początkowym. Pora więc nauczyć się ją programować. Kolejność działania jest następująca:

1. Wprowadzamy w odpowiednie miejsce programu element **Dopóki**, który znajduje się w panelu narzędzi przy jego lewej krawędzi (patrz rysunek 16.4).
2. Obok wskaźnika myszy rozwinie się paleta dwóch elementów (patrz rysunek 16.5) – wybieramy z niej górny element z fioletową górną część owalu i napisem WHILE.
3. Za wprowadzonym elementem wpisujemy warunek początkowy.
4. Za warunkiem początkowym wprowadzamy treść pętli.



Rysunek 16.4
Element Dopóki

Rysunek 16.5
Paleta pętli Dopóki

Najczęściej używane języki programowania nazywają konstrukcje programowe słowami angielskimi, stąd programiści mówiąc o swoich programach używają nazw angielskich. Na szczęście wszystkie te języki mają podobne podstawowe konstrukcje.

Ponieważ już niedługo z pewnością będziemy korzystać z zaawansowanej literatury programistycznej ułatwimy przejście do fachowej terminologii. Przestaniemy od tej chwili używać polskich odpowiedników nazw kluczowych elementów, które określają poszczególne konstrukcje programowe. Będziemy używać nazw angielskich, które są zresztą pokazane na dole elementów Baltiego. Tak samo będziemy nazywać całe konstrukcje programowe.

Od tej chwili element **Dopóki** nazywać będziemy **While** a pętlę z warunkiem początkowym także będziemy nazywać **pętlą while**.

Rysunek 16.6
Klomb ze skradzionym tulipanem

Spróbujmy samodzielnie zaprogramować następny program. Baltie, który stoi w swojej pozycji wyjściowej, dowiedział się, że gdzieś z klombu tulipanów, który jest w wierszu nad nim, zniknął jeden kwiat (patrz rysunek 16.6). Jego zadaniem jest znaleźć miejsce po kwiatku, posadzić tam nowy tulipan i wrócić do swojej pozycji wyjściowej. Powinniśmy przy tym pamiętać, że dzisiejsi złodzieje są dość bezczelni i wykorzystując nieuwagę Baltiego, który właśnie patrzy na wschód, mogą ukraść tulipan tuż nad nim.

Rysunek 16.7
Zasadź skradziony tulipan

Podpowiedzmy, że wśród stałych całkowitych znajduje się stała **Krawędź**, która „wie", jaki numer przedmiotu ma ściana krawędzi dworku Baltiego. Z tej stałej możemy skorzystać podczas powrotu do pozycji wyjściowej. Przykładowe rozwiązanie pokazano na rysunku 16.7.

16.2 Pętla z warunkiem końcowym

„Siostrzaną" konstrukcją do pętli z warunkiem początkowym jest **pętla z warunkiem końcowym** (patrz rysunek 16.8). Od pętli z warunkiem początkowym różni się tylko tym, że warunek, który decyduje o następnym przejściu przez treść pętli, sprawdzany jest dopiero po wykonaniu treści pętli. Z tego widać, że treść pętli z warunkiem końcowym wykona się zawsze przynajmniej raz. Jest to przydatne zwłaszcza w przypadkach, gdy treść pętli dopiero przygotuje wartość, która będzie sprawdzana w warunku.

Rysunek 16.8
Pętla z warunkiem końcowym

Pętla z warunkiem końcowym jest przykładem pętli różnie nazywanej przez autorów języków programowania. W niektórych językach nazwano ją **pętlą do-while,** w innych **pętlą repeat-until.** My (wspólnie z Baltiem) wykorzystamy pierwszą z tych nazw.

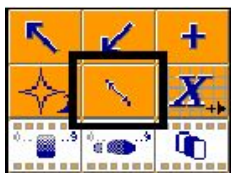
Typowym przykładem sytuacji, w których używamy pętli z warunkiem końcowym, jest sprawdzenie wprowadzonych danych, czyli *wejścia*. Wyobraźmy sobie, że użytkownik ma wybrać przedmiot, z którym będzie się później pracować. Koniecznie jednak chcemy, żeby to był przedmiot z banku numer zero. Po każdym wprowadzeniu przedmiotu powinniśmy więc sprawdzić, czy użytkownik wybrał rzeczywiście przedmiot z banku zero i jeśli skorzystał z innego banku musimy poprosić go o ponowne wybranie przedmiotu. Część programu, która realizuje to zadanie możemy zobaczyć na rysunku 16.9.

Rysunek 16.9
Sprawdzenie wejścia

16.3 Zadania samodzielne

Odległość dwóch pikseli na ekranie

Do przygotowania następnego program musimy wyjaśnić metodę otrzymywania odległości dwóch pikseli. Pisząc gry często potrzebujemy obliczyć jak daleko od siebie są jakieś dwa piksele lub dwa pola. Formuła służąca obliczeniu odległości nie należy do najprostszych i w dodatku używa operacji, których jeszcze nie znamy. Autorzy Baltiego ułatwili nam to zadanie i do elementów Baltiego dodali element **Odległość** pozwalający dość łatwo rozwiązać ten problem. Element **Odległość** znajduje się w środku pola elementów (patrz rysunek 16.10).



Rysunek 16.10
Element Odległość

Za elementem **Odległość** wprowadzamy albo elementy obiektów, których odległość chcemy

otrzymać (np. Baltie, wskaźnik myszy itd.), albo wprost współrzędne. Przykładowa pętla nieskończona z rysunku 16.11 ciągle wyświetla odległość wskaźnika myszy od lewego górnego narożnika dworku Baltiego.

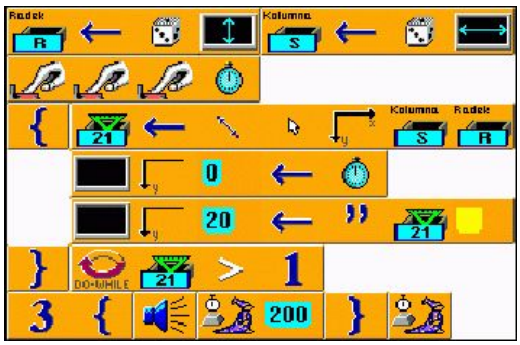
Rysunek 16.11
Ciągle wyświetlaj odległość wskaźnika myszy od lewego górnego narożnika dworku

Szukamy czarnych dziur

Sprawdzimy programując prostą grę, ile zrozumieliśmy z poprzednich rozważań. Wiele słyszy się o czarnych dziurach jako końcowej fazie istnienia wielkich gwiazd. Mówi się nawet, że czarna dziura może być takim szczególnym miejscem, przez które można dostać się do innej części wszechświata lub nawet do innego wszechświata.

Trudno odszukać czarne dziury we Wszechświecie bo są niewidzialne. Tą właściwość czarnych dziur wykorzystamy w programie, w którym będziemy mogli zagrać w grę „Szukanie czarnej dziury".

Na początku program wybierze losowo jeden piksel na dworku Baltiego i zrobi z niego czarną dziurę. Zadaniem gracza będzie odnalezienie czarną dziurę i wskazanie na nią myszą. Dla ułatwienia na ekranie będzie pokazywana aktualna odległość wskaźnika kursora myszy od dziury. Będzie także pokazany na ekranie czas szukania, który po znalezieniu dziury zostanie zatrzymany. Pozwoli to graczom robić wyścigi, kto szybciej znajdzie czarną dziurę. Po trafieniu „w dziurę" komputer „zadzwoni" ogłaszając zwycięstwo.



Rysunek 16.12
Gra Szukanie czarnej dziury

Swoje rozwiązanie możemy porównać z programem z rysunku 16.12. Grę można dalej ulepszać. Można np. przygotować sceny różnych wszechświatów i dodać nowy efekt: znalezieniu czarnej dziury kosmonauta przesunie się do innego wszechświata, w którym zostanie otworzona inna scena itd.

Wykafelkuj dworek

Teraz spróbujmy rozwiązać trochę trudniejszy problem. Wyobraźmy sobie, że Baltie powinien wykafelkować swój dworek. Otrzyma przy tym zadanie dodatkowe, by wykafelkował dworek w spirale od krawędzi do środku. Takie zadanie z pewnością spokojnie rozwiążemy.

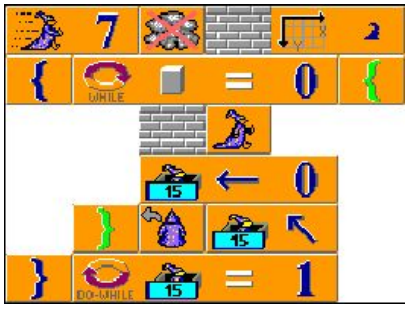
Jednym z możliwych sposobów rozwiązania zadania jest sprawdzanie przez Baltiego przedmiotów przed sobą. Dopóki nic nie znajdzie, czyli dopóki numer przedmiotu przed Baltiem to zero, niech czaruje przed sobą przedmiot, którym ma wykafelkować dworek. W chwili, gdy trafi na przedmiot o numerze różnym od zera będziemy wiedzieć, że doszedł do ściany lub do poprzednio położonego przedmiotu. Zrobimy wtedy „w lewo zwrot" położymy dalsze kafelki.

Przypuszczalnie pisząc program trafimy na moment, gdy Baltie wykafelkuje cały dworek i potem kręci się na miejscu szukając, gdzie przed nim jest puste pole. Rozwiązanie tego problem wymaga sprytu.

Baltie powinien rozpoznać, że obrócił się dwa razy i patrzy w kierunku, z którego przyszedł. Zapobiegnie to kręceniu się w miejscu na końcu programu, gdyż w tym momencie wszystko jest już wykafelkowane i nie ma pustego pola. Baltie może skończyć pracę.

Jak Baltie ma rozpoznać taką sytuację? Może na przykład liczyć swoje obroty a ich liczbę może odkładać do zmiennej. Powinien jednak liczyć tylko obroty na miejscu, zatem gdy ruszy do przodu musi wyzerować zmienną, w której liczy swoje obroty. Wtedy przy następnym obrocie będzie liczyć od początku.

Teraz już wiemy wszystko co potrzebne do zrozumienia konstrukcji programu z rysunku 16.13, który pokazuje jedno z możliwych rozwiązań. Wypróbujmy go.

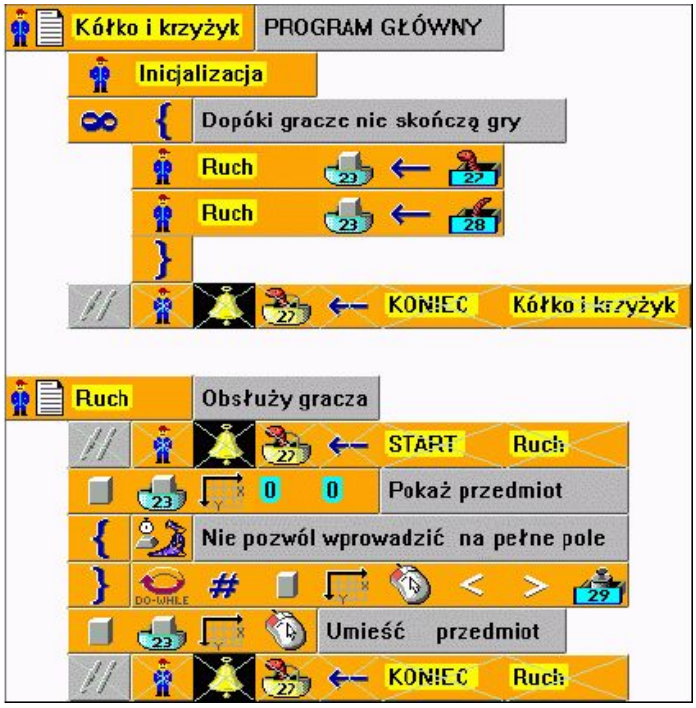


Rysunek 16.13
Wykafelkuj dworek w spirale

Bezpieczniejsze kółko i krzyżyk

Nasza aktualna wersja gry w kółko i krzyżyk nie sprawdzała, czy gracze poprawnie umieszczają swoje przedmioty na planszy. Pozwalała im umieścić przedmiot na polu, na którym leżał już inny przedmiot. Gracze mogli nawet umieścić przedmiot poza planszą. Spróbujmy wymyślić, jak należy zmienić pętlę gry, by gracze mogli umieszczać swoje przedmioty tylko na polach planszy do gry. Dodatkowo na polu nie może być jeszcze żadnego przedmiotu.

Na rysunku 16.14 możemy zobaczyć część rozwiązania, które jest rozwinięciem programu pokazanego na rysunku 14.15. Zmieniono w nim tylko pętlę gry wykorzystując możliwość poruszania przedmiotami za pomocą pomocnika, któremu podamy wygląd ustawianego kamienia. Pomocnik **Ruch** wyświetli potem przedmiot gracza, który ma ruch i poczeka, aż gracz kliknie na pustym polu. Czekanie powtarza, dopóki numer przedmiotu, na którym gracz kliknął lewym przyciskiem nie jest zgodny z numerem przedmiotu, który gracze na początku gry wskazali jako przedmiot do wypełnienia pustej przestrzeni.

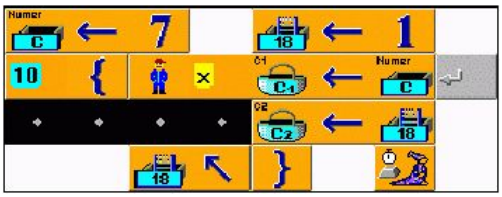


Rysunek 16.14
Kółko i krzyżyk z kontrolą dozwolonych ruchów – pętla gry i pomocnik Ruch

16.4 Pętla z parametrem

W programach bardzo często pojawiają się pętle, w których coś liczymy. Wyobraźmy sobie, że potrzebujemy napisać np. program, który wyświetli na ekranie tabliczkę mnożenia dla pewnej liczby. Taki program (jak i każdy inny) można napisać w wiele sposobów. Ponieważ we wszystkich będziemy chcieli wyświetlić cały przykład na ekranie, przygotujemy najpierw pomocnika (nazwijmy go **X** od znaku mnożenia), który zrobi co potrzeba. Możemy zobaczyć go na rysunku 16.15.

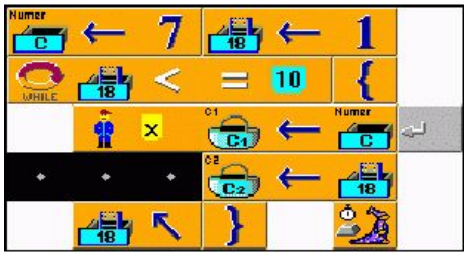
Rysunek 16.15
Pomocnik wyświetlenia przykładu mnożenia



Rysunek 16.16
Rozwiązanie z użyciem pętli z określoną liczbą powtórzeń

Kiedy jeszcze nie znaliśmy pętli z warunkiem użylibyśmy tutaj pętli z określoną liczbą powtórzeń. Nasz program wyświetlenia tabliczki mnożenia przez siedem wyglądałby pewnie nieco podobnie jak program z rysunku 16.16.

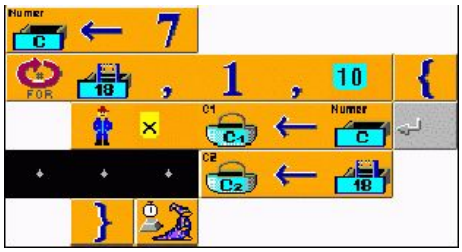
Niektóre języki programowania nie posiadają pętli z określoną liczbą powtórzeń. Wtedy liczbę powtórzeń musi programista sam obliczać i sam też powinien zaprogramować decyzję o momencie zakończenia pętli. Program, rozwiązujący nasz problem za pomocą pętli z warunkiem na początku, wyglądałby jak ten z rysunku 16.17.



Rysunek 16.17
Rozwiązanie wykorzystujące pętlę z warunkiem

Istnieje wiele podobnych zadań jak to z tabliczką mnożenia. Autorzy języków programowania starali się więc wprowadzić konstrukcję, który ułatwiałaby oprogramowanie zadań tego rodzaju. Dlatego wprowadzili

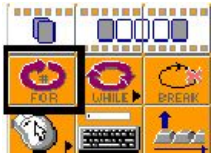
tzw. **pętlę z parametrem**, którą nazwano **pętlą for**. Ten rodzaj pętli ma też Baltie w swoim arsenale. Jedno z możliwych rozwiązań zadania wykorzystujące pętlę z parametrem można zobaczyć na rysunku 16.18.



Rysunek 16.18
Rozwiązanie z użyciem pętli z parametrem

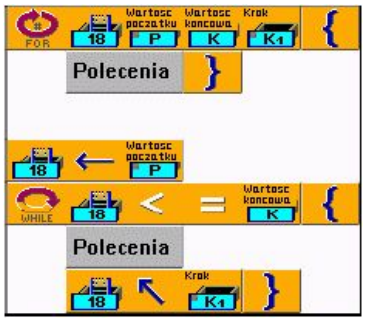
Popatrzmy teraz jak taka pętla wygląda i na co nam pozwala. Jak widać nagłówek tej pętli jest trochę bardziej skomplikowany.

1. Nagłówek zaczyna się od elementu **For (Dla)**, który znajduje się obok lewej krawędzi panelu elementów na lewo od elementu **While (Dopóki)** – patrz rysunek 16.19.



Rysunek 16.19
Element For (Dla)

2. Za elementem **For (Dla)** musi być **parametr pętli**, tzn. zmienna, w której pamiętana będzie liczba przejść przez pętlę i której wartość jest sprawdzana dla zdecydowania, czy już skończyć pętlę.
3. Za parametrem pętli może następować (nie musi, lecz prawie zawsze tu występuje) **wartość początkowa pętli**, czyli jest wartość przydzielana na początku parametrowi pętli. Jeśli tej wartości brak, Baltie użyje zera.
4. Za wartością początkową może znajdować się **wartość końcowa pętli**, czyli ostatnia wartość parametru pętli. Słowa „ostatnia" nie można brać dosłownie, wyjaśnimy to za chwilę. W razie nie podania wartości końcowej Baltie użyje zera.
5. Za wartością końcową może występować **krok**, to jest wartość, która po każdym wykonaniu pętli zostanie dodana do parametru pętli. Jeśli brak wartości kroku Baltie jako krok przydzieli jedynkę. Ponieważ większość pętli używa właśnie kroku jeden, określenie kroku stosunkowo rzadko wystąpi w nagłówku pętli.
6. Dalej już jest tylko treść pętli. To już znamy więc nie musimy tego opisywać. Działanie pętli z parametrem pokazano na rysunku 16.20. W górnej części rysunku mamy pętlę **For** (pętla z parametrem) z określonymi wszystkimi parametrami. W dolnej części jest pętla z warunkiem początkowym, która dokładnie opisuje działanie górnej pętli z parametrem.



Rysunek 16.20
Działanie pętli FOR dla kroku > 0

Jak widać program najpierw przydzieli parametrowi pętli jego wartość początkową następnie sprawdzi, czy wartość jest mniejsza lub równa wartości końcowej. Jeśli tak, wykona polecenia z treści pętli i zwiększy parametr pętli o wartość kroku. Potem znów sprawdza, czy ma wykonać pętlę jeszcze raz.

Z tego opisu wynikają następujące konsekwencje:

- jeśli wartość początkowa parametru jest większa od wartości końcowej, treść pętli nie zostanie wykonana ani razu,
- po zakończeniu działania pętli parametr pętli będzie miał wartość, która spowodowała odmowę kolejnego wykonania treści pętli,
- jeśli w treści pętli zmienimy wartość parametru, wartość końcową lub wartość kroku, test w nagłówku pętli będzie działać z tymi zmienionymi wartościami.

Nie wolno zmieniać w treści pętli wartości parametru, wartości końcowej ani kroku. Jest to poważny błąd programistyczny. Jeśli rozwiązujemy problem za pomocą zmiany niektórych z tych wartości, lepiej użyć pętli z warunkiem.

Jak można zauważyć czytając powyższe uwagi, sposób działania pętli **for** jest ważny nie tylko dla wartości dodatniej kroku pętli, ale i dla kroku ujemnego. Jeśli natomiast krok pętli byłby zerowy, Baltie po prostu wykona treść pętli jeszcze raz. Uważajmy na to!

Teraz spróbujmy jeszcze raz uruchomić program z rysunku 16.20 z dodanym zatrzymaniem w treści pętli. Włączmy pokazywanie wartości zmiennych podczas działania programu. Pozwoli to na pewno wszystko lepiej zrozumieć.

Program samodzielny

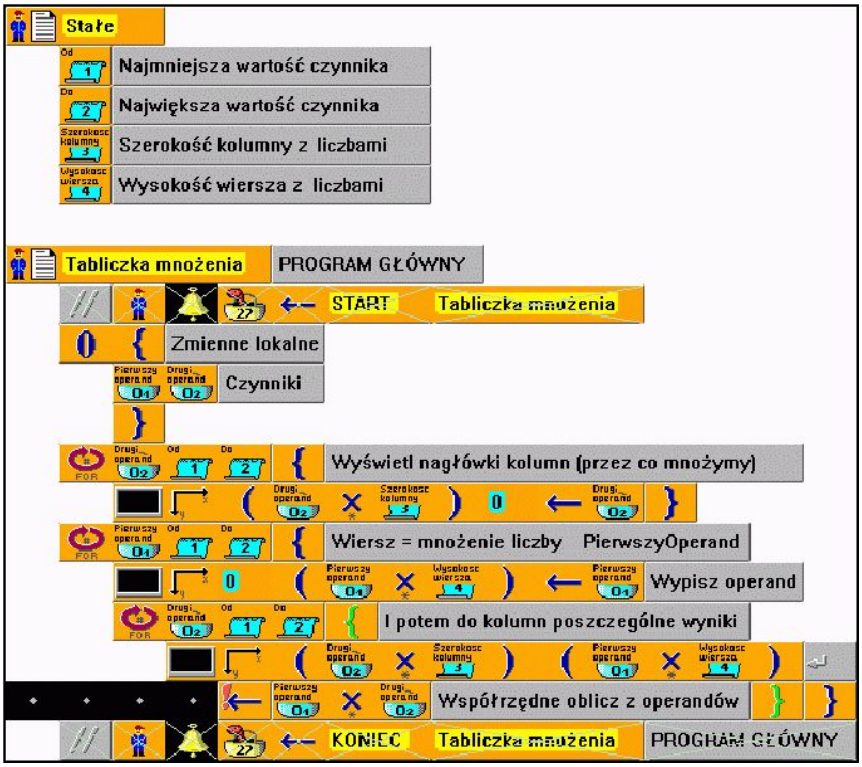
Spróbujmy przygotować program, który na dworku Baltiego wyświetli tabliczkę mnożenia. Jej wycinek można

zobaczyć na rysunku 16.21. Gdy będzie gotowy, porównajmy go z programem z rysunku 16.22.

Spróbujmy teraz wymyślić, jak zmienić program, by możliwe było określanie za pomocą stałych, od której do której liczby ma przechodzić pierwszy operand i od której do której drugi. Program mógłby wyświetlać także np. wielką tabliczkę mnożenia i inne tabliczki. Rozwiązania tego ulepszonego zadania nie pokażemy w tej książce – można je znaleźć na stronach WWW.

	1	2	3	4
1	1	2	3	4
2	2	4	6	8
3	3	6	9	12
4	4	8	12	16

Rysunek 16.21
Mała tabliczka mnożenia – fragment



Rysunek 16.22
Kluczowa część programu wyświetlenia tabliczki mnożenia na ekranie

16.5 Czego nauczyliśmy się

W tym rozdziale nauczyliśmy się programować pętle, których działanie zależy od spełnienia jakiegoś warunku. Przygotowaliśmy kilku programów, w których po kolei użyliśmy:

- pętli z warunkiem początkowym (pętli **while**),
- pętli z warunkiem końcowym (pętli **do-while**)
- pętli z parametrem (pętli **for**).

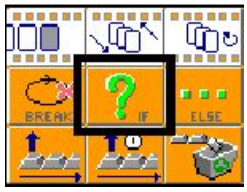
17. Uczymy Baltiego myśleć

Czego nauczymy się w tym rozdziale

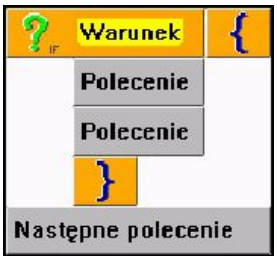
W tym rozdziale nauczymy się wyjaśniać Baltiemu, jak powinien zachowywać się w najróżniejszych sytuacjach. Najpierw pokażemy jak zapisać, że Baltie ma wykonać pewną akcję tylko w przypadku, gdy jest spełniony określony warunek. Potem wyjaśnimy, jak sprawić, by Baltie decydował w zależności od podanego warunku, którą z dwóch operacji wykonać. Na końcu nauczymy się programować bardziej złożone przypadki wyboru i dowiemy się, jak można poprawić szybkość i czytelność programu w sytuacji, gdy akcja jest wybierana w zależności od wyniku obliczonego wyrażenia.

17.1 Warunek prosty

Jak dotąd każda decyzja Baltiego dotyczyła tylko kontynuacji wykonywania pętli ewentualnie możliwości opuszczenia pętli i przejścia do wykonywania reszty programu. Teraz nauczymy się rozwiązywać zadania, gdy program powinien decydować w nieco inny sposób.



Rysunek 17.1
Element If (Jeśli)



Rysunek 17.2
Polecenie warunkowe

Bardzo często zdarza się w programowaniu, że program powinien wykonać jakąś akcję tylko wtedy, gdy jest spełniony jakiś warunek. Programiści używają wtedy specjalnej konstrukcji, która nazywa się **polecenie warunkowe proste** (słowo *proste* bywa często pomijane, używać też będziemy określenia **polecenie if)**. Taką konstrukcją dysponuje także Baltie. Jak wygląda polecenie warunkowe możemy zobaczyć na rysunku 17.2.

Kiedy podczas wykonywania programu Baltie natrafi na polecenie warunkowe, sprawdza, czy warunek jest spełniony. Jeśli tak, wykona polecenie umieszczone za warunkiem (na rysunku 17.2 za warunkiem umieszczono polecenie złożone). Jeżeli warunek nie jest spełniony, to polecenie zostanie ominięte i Baltie kontynuuje program od następnego polecenia.

Spróbujmy teraz za pomocą polecenia warunkowego rozwiązać następny problem: cały dworek Baltiego jest pokryty oknami (przedmiot nr 8076). Niektóre z nich są jednak rozbite. Zadaniem Baltiego jest przejść przez cały dworek i oszklić rozbite okna. Popatrzmy na rysunek 17.3 zawierający jedno z możliwych rozwiązań. Wyjaśnijmy go. Na początku potrzeba pokryć przestrzeń oknami (pierwsze dwie pętle **for**) i potem między nimi wyczarować losowo kilka rozbitych okien (trzecia pętla **for**). Tę część można wykonać z nieskończoną szybkością.

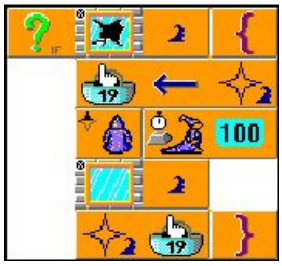
Rysunek 17.3
Baltie szklarz

Teraz na scenę wkracza Baltie szklarz. Powinien przejść przez 10 wierszy zawierających po 15 okien w każdym wierszu. Przy każdym oknie sprawdzi, czy nie jest przypadkiem rozbite. Oszkli rozbite okno. Dla uproszczenia nie czarujemy przed Baltiem, ale na jego pozycji. Całe okna nie interesują Baltiego. Kiedy sprawdzi ostatnie okno, dojdzie do ściany. Skorzystamy z tego, że można zrobić dodatkowy krok do ściany. Tam przesunie się o wiersz wyżej i obróci się na prawo. Jeśli zobaczy, że jest przed nim krawędź dworka, zrozumie, że pomylił się, obróci się w drugą stronę i kontynuuje sprawdzanie w następnym wierszu okien.

W tym przykładzie treść poleceń warunkowych były zawsze tworzone tylko przez jedno polecenie. Spróbujmy teraz poprzednie zadanie zmienić tak, by Baltie podczas szklenia okna stał czołem do niego i żeby to szklenie trwało jakiś czas.

Trochę skomplikuje nam to orientację Baltiego. Powinien teraz zapamiętać kierunek marszu, żeby po oszkleniu kontynuować podróż w tę samą stronę. Umożliwi nam to polecenie **Kierunek Baltiego,** którego używaliśmy do

otrzymania współrzędnych pola przed Baltiem. Teraz wykorzystamy go inaczej. Skorzystamy z tego, że samotny element **Kierunek Baltiego** podaje numer kierunku, w którym Baltie jest obrócony. Jeśli pojawi się w programie ten element z liczbą, Baltie interpretuje liczbę jako numer kierunku i odpowiednio się obróci. Spróbujmy dostosować polecenie warunkowe tak, żeby działało według zmienionego zadania. Uzyskane rozwiązanie możemy porównać z rozwiązaniem z rysunku 17.4.

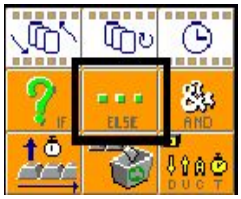


Rysunek 17.4
Szkli twarzą do okna

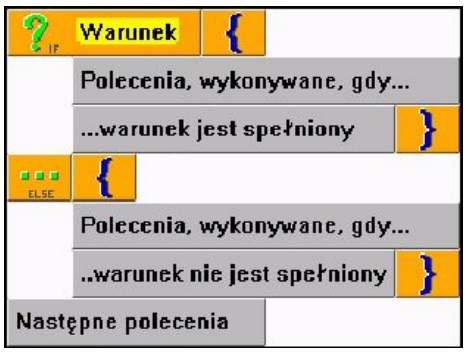
17.2 Wybieramy z dwóch możliwości

Stosunkowo często pojawiają się w programach sytuacje, gdy powinniśmy wybrać jedną z dwóch możliwości, rzadziej musimy wybrać, czy coś wykonać, czy nie. Do tego nie wystarczy proste polecenie warunkowe.

Programiści rozwiązali to zadanie dodając do prostego polecenia warunkowego drugą gałąź, która zawiera polecenie wykonywane w przypadku niespełnienia sprawdzanego warunku – patrz rysunek 17.5. Tę gałąź nazywamy **gałęzią else** i w programach Baltiego zaczyna się od elementu **Inaczej** (według naszej umowy będziemy nazywać go elementem **else**). W panelu elementów znajduje się obok elementu **if** – patrz rysunek 17.6. Całą konstrukcję nazywamy **kompletnym poleceniem warunkowym** (programiści nazywają go **poleceniem if-then-else** lub tylko **if-else**).



Rysunek 17.6
Element Else (Inaczej)



Rysunek 17.5
Kompletne polecenie warunkowe

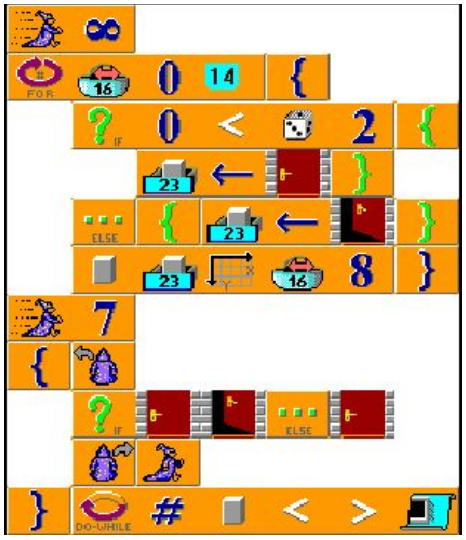
Rysunek 17.7
Szereg drzwi

Wypróbujmy go w prościutkim zadaniu. Wyobraźmy sobie, że Baltie stoi w pozycji startowej i nad nim jest szereg drzwi, z których niektóre są otwarte a inne zamknięte (patrz rysunek 17.7). Baltie lubi płatać figle, więc idzie i otwiera wszystkie zamknięte drzwi a zamyka drzwi otwarte.

Zadanie jest na tyle proste, że łatwo je rozwiązać samodzielnie. Rozwiązanie można porównać z tym z rysunku 17.8, w którym tym razem mamy polecenie ustawienia szybkości.

Jak już zapewne zauważyliśmy w programie na rysunku 17.8 kompletne polecenie warunkowe zostało użyte dwa razy:

pierwszy raz podczas generowania szeregu drzwi i drugi razem podczas ich otwierania i zamykania. W drugim poleceniu pokazano, że można zapisać całe polecenie warunkowe w jedynym wierszu o ile warunek i polecenia obu gałęzi są wystarczająco krótkie.



Rysunek 17.8
Baltie i drzwi

17.3 Bardziej skomplikowana decyzja, wybór z kilku możliwości

W wielu sytuacjach nie wystarczy nam pojedyncze kompletne polecenie warunkowe, lecz musimy użyć kilku takich poleceń. Stosunkowo często zdarza się, że w przypadku nie spełnienia warunku sprawdzamy następny warunek a potem następny itd. aż do kompletnego rozwiązania problemu.

Popatrzmy na szkielet programu z rysunku 17.9. Nie jest on jeszcze pełnym programem, warunki i polecenia są w nim zastąpione elementami tekstowymi. Wyjaśnimy go i pokażemy, jak mógłby według tego programu Baltie (lub ktoś inny) decydować, co będzie robić.

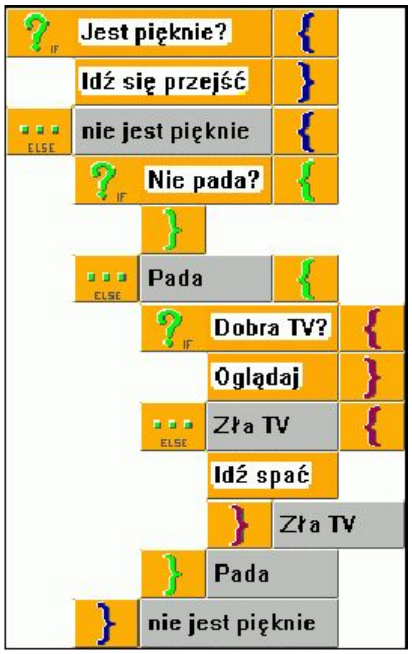
Najpierw sprawdziłby, czy na dworze jest piękna pogoda. Jeśli tak – idzie się przejść. Jeśli nie, sprawdzi, czy nie pada deszcz. Jeśli nie pada – pójdzie na ryby. Jeśli pada, sprawdzi, czy jest coś ciekawego w telewizji. Jeśli tak, będzie oglądać telewizję. Jeśli nie – idzie spać.

Zapis w postaci widocznej na rysunku 17.9 nie jest optymalny dla takiego rodzaju wyborów z poprawnym wcięciem poszczególnych bloków. W praktyce raczej stosuje się układ graficzny programu w postaci użytej na rysunku 17.10, który czyni program znacznie czytelniejszym.

Rysunek 17.10
Wybór z większej liczby możliwości

Przy takim układzie programu wszystkie elementy **else** mają równe wcięcia. Skorzystano z tego, że w gałęzi **else** jest tylko jedno polecenie (zagnieżdżone polecenie **if**), więc nie trzeba używać nawiasów blokowych, co upraszcza zapis.

Zagnieżdżonych poleceń warunkowych często używa się do wybrania akcji w zależności od wyniku jakiegoś polecenia. W takim przypadku nie trzeba jednak sprawdzać wszystkich możliwych wyników po kolei. Szybciej jest przygotować na początku tablicę wartości i przeskoczyć według wyniku do odpowiedniej gałęzi. Programiści korzystają z wprowadzonych do języków programowania (Baltie jest jednym z nich) konstrukcji programowych nazywanych przeważnie **poleceniem case** (tak będziemy je nazywać) lub **poleceniem select**. Schemat tego polecenia można zobaczyć na rysunku 17.11.



Rysunek 17.9
Zagnieżdżone polecenia warunkowe

Spróbujmy zaprogramować prosty przykład. Wyobraźmy sobie, że na dworku Baltiego jest łąka, na której rosną kwiaty. Kwiat najpierw wyrośnie jako mały pączek, (przedmiot 1109), podrośnie (przedmioty 1124 i 1139), zakwitnie (przedmiot 1141) i uschnie (przedmioty 1126 a 1111), a na jego miejscu znów pojawi się trawa (przedmiot 1122). Naszym zadaniem jest zaprogramować losowe umieszczenie kwiatów w poszczególnych fazach rozwoju (włącznie z trawą) i potem pozwolić tym kwiatom rosnąć i usychać.

Przygotowując program zauważymy, że jeśli będziemy po kolei zmieniać fazy rozwoju kwiatów, po łące będą przebiegać fale. Nie pomoże nawet ustawienie nieskończonej szybkości. Dlatego wprowadzimy losowy wybór kwiatów, które wyrosną lub uschną. Efektem jest migotanie na łące, co jest znacznie ciekawsze.

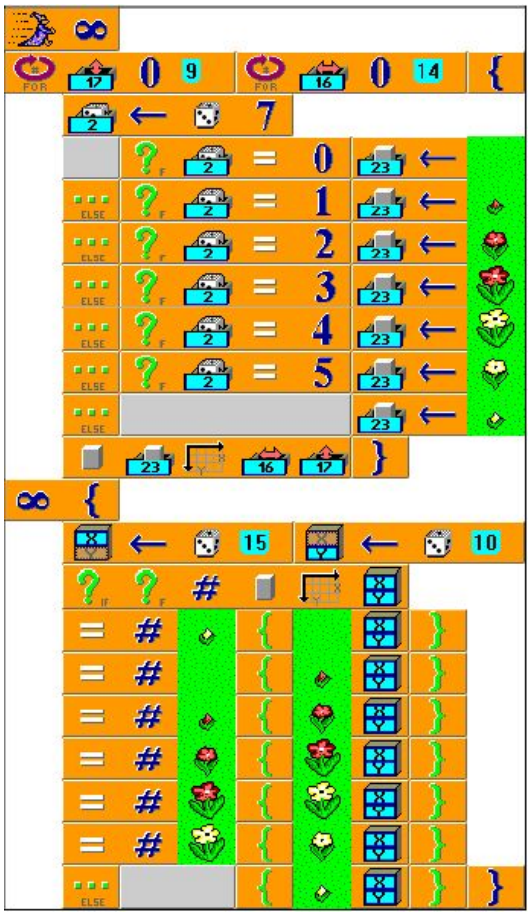
Program, który realizuje to zadanie można zobaczyć na rysunku 17.12. Każdą część zaprogramowano inaczej, aby pokazać obie konstrukcje, o których mówiliśmy w tym rozdziale.

Rysunek 17.11
Polecenie Case

W pierwszej części użyliśmy zagnieżdżonego polecenia warunkowego. Współrzędne kwiatów przechowywaliśmy w dwóch pojedynczych zmiennych. W treści poszczególnych poleceń warunkowych określamy tylko czarowany przedmiot, którego numer przechowujemy w zmiennej **Przedmiot** na końcu pętli. Potem wyczarowujemy ten przedmiot na odpowiednich współrzędnych.

W drugiej części użyto polecenia **case**. Współrzędne zapamiętane są w pojedynczych składnikach zmiennej podwójnej i odpowiedni przedmiot wyczarowujemy na określonych współrzędnych, gdy tylko dowiemy się, który przedmiot ma zostać wyczarowany.

Nawiasy blokowe (złożone) ograniczające pojedyncze sekcje polecenia **case** nie są obowiązkowe, bo treść zawiera pojedyncze polecenie: wyczarowanie określonego przedmiotu na odpowiednich współrzędnych. Wprowadziliśmy nawiasy do programu tylko dla poprawienia czytelności (przynajmniej mamy nadzieję, że program jest bardziej czytelny).



Rysunek 17.12
Kwitnąca i przekwitająca łąka

17.4 Czego nauczyliśmy się

W tym rozdziale nauczyliśmy się używać najróżniejszych rodzajów poleceń warunkowych i wprowadzać za ich pomocą do programu reakcje na najróżniejsze sytuacje. Najpierw pokazaliśmy proste polecenie warunkowe (polecenie **if**), którego używamy wtedy, gdy Baltie powinien wykonać jakąś akcję tylko, gdy jest spełniony określony warunek. Potem wyjaśniliśmy kompletne polecenie warunkowe (polecenie**if-else**), którego użyjemy wtedy, gdy Baltie po sprawdzeniu warunku decyduje, które z dwóch możliwych poleceń wybrać. Nauczyliśmy się programować bardziej złożone decyzje i dowiedzieliśmy się, jak można poprawić czytelność niektórych rodzajów decyzji. Na końcu pokazaliśmy polecenie**case** i zaprogramowaliśmy zadanie, w którym Baltie musiał ciągle o czymś decydować.

18. Określamy wygląd wyświetlanego tekstu

Czego nauczymy się w tym rozdziale

W tym rozdziale pokażemy, jak można zmienić wygląd wyświetlanych na ekranie liczby i tekstów. Pokażemy, jak określić zestaw znaków, wielkość liter, styl, kolor i niektóre inne właściwości.

18.1 Ustawienie parametrów czcionki

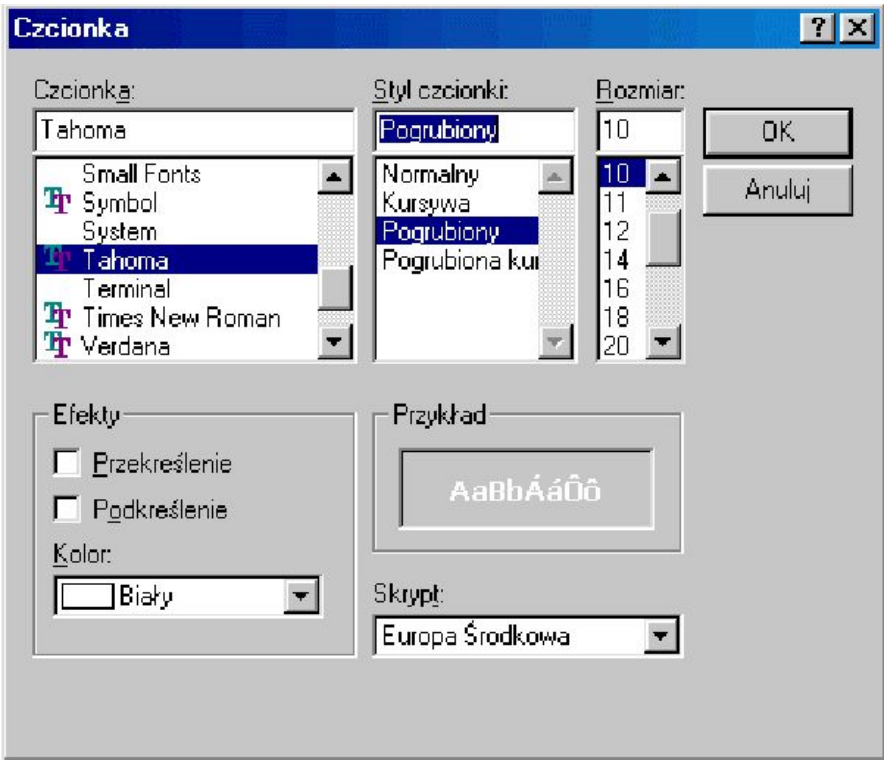
Gdy po raz pierwszy pokażemy komuś wyświetlanie tekstów i liczb na ekranie swojego komputera na pewno poskarży się na zbyt małe litery, których używa Baltie. Pokażemy teraz, jak można zmienić wygląd *tekstu wyjścia*. Jako *tekst wyjścia* będziemy traktować także liczby – cyfry też są przecież znakami.

Wygląd (fachowo mówi się *format*) tekstu na ekranie ustawiamy za pomocą elementu **Czcionka**, który znajduje się w panelu elementów w środku drugiego wiersza od dołu (patrz rysunek 18.1). Po wprowadzeniu tego elementu do programu otworzy się okno dialogu czcionki (patrz rysunek 18.2), w którym można ustawić wszystkie potrzebne parametry.

Rysunek 18.1
Element Czcionka

Ponieważ czasami chcemy coś szczególnie elegancko wypisać, popatrzmy po kolei na możliwości, które oferuje nam to okno.

Przy uważnym spojrzeniu na element **Czcionka** zobaczymy, że wokół trzech znaków ma ramkę okna z nagłówkiem. Podobnie wyglądają też dwaj sąsiedzi. Tak Baltie stara się podpowiedzieć, że po wprowadzeniu tego elementu do programu otworzy się okno dialogu, w którym określimy parametry, które zostaną uwzględnione przez program.



Rysunek 18.2
Okno dialogu Czcionka

Czcionka – zestaw znaków

Do pola z listą **Czcionka** możemy wprowadzić nazwę czcionki (zestawu znaków), którego chcemy użyć. Nazwę nie musimy wprowadzać ręcznie, wystarczy poszukać odpowiedniej nazwy w rozwiniętej liście i kliknąć. Komputer automatycznie przeniesie nazwę do pola wprowadzania.

Zwróćmy uwagę, że na lewo, obok nazw czcionek są jakieś ikony. Określają sposób konstrukcji czcionki. Z czcionkami, które tu nie mają ikony, komputer działa najszybciej, lecz możemy ich użyć tylko w kilku określonych wielkościach. Mają też i inne wady. Większą dowolność w ustawianiu wielkości i innych parametrów dają czcionki oznaczone ikoną z dwoma wielkimi literami **T** (czcionki typu *TrueType*) lub ikoną z wielkim **O** (czcionki typu *OpenType*).

Przekazując swoje programy przyjaciołom lub udostępniając je w Internecie wybierajmy tylko takie czcionki, co do których jesteśmy pewni, że wszyscy użytkownicy mają je zainstalowane. W przeciwnym razie może się zdarzyć, że użytkownik zamiast znaków oglądać będzie na ekranie prostokąciki lub różne dziwne znaki.

Wybierając czcionkę możemy liczyć na to, że na wszystkich komputerach z systemem operacyjnym *Windows* (a to właśnie dla nich tworzymy nasze programy) zawsze znajdziemy trzy czcionki:

- **Times New Roman** jako reprezentant czcionek szeryfowych (użyto jej do napisania tej książki),
- **Arial** jako reprezentant czcionek bezszeryfowych (czcionki Arial użyto w tej książce do nagłówków),
- **Courier New** jako reprezentant czcionek nieproporcjonalnych, tzn. czcionek, w których wszystkie litery mają taką samą szerokość. W czcionkach proporcjonalnych **m** zajmuje więcej miejsca niż literą **i**, w czcionkach Courier zajmują tyle samo miejsca.

Może się zdarzyć, że komputer oferuje więcej wersji jednej czcionki. Używamy wówczas tej, której nazwa kończy się znakami **PL** lub **CE**. Zawierają one polskie znaki diakrytyczne: ą, ć, ę itp. W nowszych wersjach *Windows* nie ma już czcionek z nazwą kończącą się **PL**, bo zmieniono koncepcję lokalizacji *Windows*. Czcionki „znają" wszystkie znaki włącznie z arabskimi i koreańskimi. To, z którego regionu językowego będzie czcionka, powinno ustawić się w liście **Skrypt,** lecz ponieważ opinia firmy *Microsoft* o tym, jak wspierać alfabety narodowe, zmienia się z każdą wersją systemu, Baltie nie reaguje na to ustawienie.

Styl czcionki

Większość czcionek może występować w kilku stylach. Czym różni się **czcionka tłusta** od normalnej, wiadomo. Wiadomo też, czym różni się czcionka pisana *kursywą* – jest to taka trochę zgięta czcionka. Jak jednak można zobaczyć w tym tekście, *kursywa* nie zawsze znaczy tylko pochylenie tekstu. Może zawierać też inne małe zmiany wyglądu czcionki. Kursywa czcionki Times jest bardziej "okrągła" niż zwykła czcionka.

Wielkość czcionki

Wielkość czcionki jest określana w tzn. **punktach.** Jeden punkt to około jednej trzeciej milimetra (dokładnie 1/72 cala, tzn. 25,4 / 72 = 3,53 mm). W oknie dialogu **Czcionka** w liście **Wielkość** system oferuje wielkości od 8 do 72 punktów. Można jednak wprowadzić wielkość, której na liście brak. Uważajmy na następne ograniczenia:

- wielkość liter powinna być liczbą całkowitą,
- większość czcionek na ekranie jest czytelna od wielkości 7 punktów,
- sześciopunktowa czcionka jest czytelna tylko przy odrobinie fantazji.

W przestrzeni Baltiego mieści się czcionka nie większa niż 160 punktów. Jeśli ograniczymy się do liter bez diakrytyki, zmieści się na dworku czcionka do wielkości około 200 punktów.

Podczas eksperymentów może zdarzyć się, że nastawimy za dużą wielkość czcionki. Wtedy przy próbie wypisania tekstu na ekranie program może odmówić wykonania polecenia.

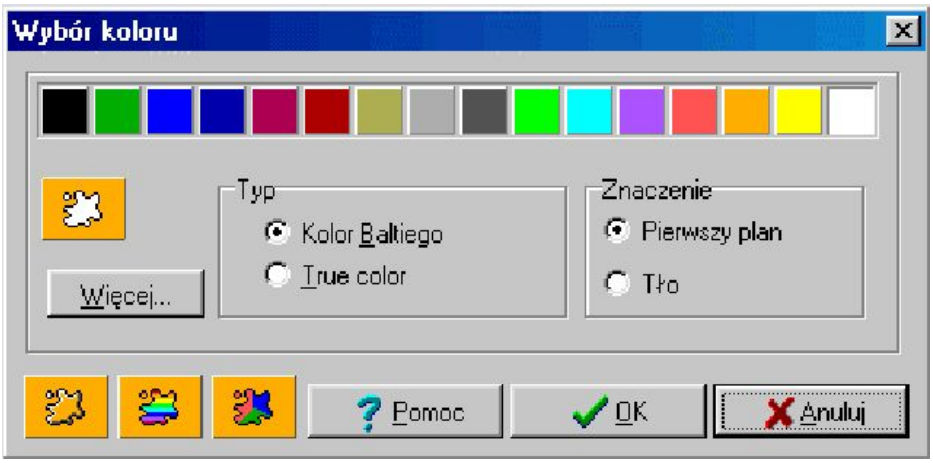
Kolor czcionki i kolor tła

Gdy klikniemy na strzałce obok pola **Kolor** rozwinie się lista, z której można wybrać jeden z 16 oferowanych kolorów. Kolor czcionki trzeba wybierać tak, by na tle, na którym wyświetlamy teksty, czcionka była dobrze widoczna

i nie gubiła się w tle. Możemy zmienić domyślny, czarny kolor tła. Wystarczy wprowadzić za element ustawienia parametrów wyjścia element **Kolor**. Znajduje się on w panelu elementów po lewej obok elementu **Czcionka** – patrz rysunek 18.3. Jak można odgadnąć z wyglądu elementu po jego wprowadzeniu do programu otworzy się okno dialogu (patrz notatka na końcu rozdziału 17), w którym ustawiamy odpowiedni kolor.

Rysunek 18.3
Element Kolor

W oknie dialogu **Kolor** (rysunek 18.4) należy najpierw ustawić przełącznik **Znaczenie** zależnie od tego, czy określamy kolor pierwszego planu czy kolor tła. Zgodnie z tym zostanie zmieniony także wygląd elementu pokazanego nad przyciskiem **Następny.** Potem klikamy na ten z 16 oferowanych kolorów Baltiego, który wybieramy i możemy zamknąć okno. Oferta kolorów jest pokazana pod nagłówkiem okna. Jeśli nie odpowiadają nam kolory Baltiego możemy wybrać dowolny z kolorów, który jest w stanie zaoferować system operacyjny. Jak to zrobić opisano w pomocy, którą można otworzyć np. przesunięciem elementu **Kolor** na znak zapytania pomocy w panelu narzędzi.



Rysunek 18.4
Okno dialogu Kolor

18.2 Eksperymentujemy z wyświetlaniem na ekranie

Po wprowadzeniu wszystkich parametrów okno dialogu **Czcionka** zamknie się i do programu zostanie wprowadzone polecenie składające się z serii kilku elementów. Pierwszym z nich będzie element **Czcionka**. Za nim kolejno nazwa czcionki i jego wielkość w punktach. Jeśli wprowadzimy czcienkę pogrubioną, kursywę, podkreśloną lub przekreśloną to za wielkością czcionki będzie wpisany ciąg tekstowy ze znakami, określającymi wprowadzone parametry (pogrubienie = **B** [*Bold*], kursywa = **I** [*Italics*], podkreślona = **U** [*Underlined*], przekreślona = **S** [*Strikeout*]). Jeśli zmienimy w zadaniu kolor czcionki, na końcu element pojawi się element określający kolor czcionki. Kolor tła określamy wprowadzając odpowiedni elementu koloru tła.

Pamiętajmy, że polecenie ustawienia formatu wyświetlania ustawia format dla wszystkich tekstów z wyjątkiem tych, które mają swój własny format. Format ten można określić wprowadzając na koniec polecenia wypisania element **Czcionka** z odpowiednimi parametrami.

Łatwo zgadnąć, że do późniejszych zmian niektórych parametrów wyświetlania nie musimy ponownie otwierać okno dialogu **Czcionka**, lecz wystarczy ręcznie dostosować wartości odpowiednich literałów. Tylko w przypadku zestawu znaków polecamy jednak wybór czcionki z listy w oknie dialogu **Czcionka,** bo tylko wtedy możemy być pewni, że nie zrobiliśmy błędu w nazwie czcionki i wybraliśmy czcionkę istniejącą w komputerze.

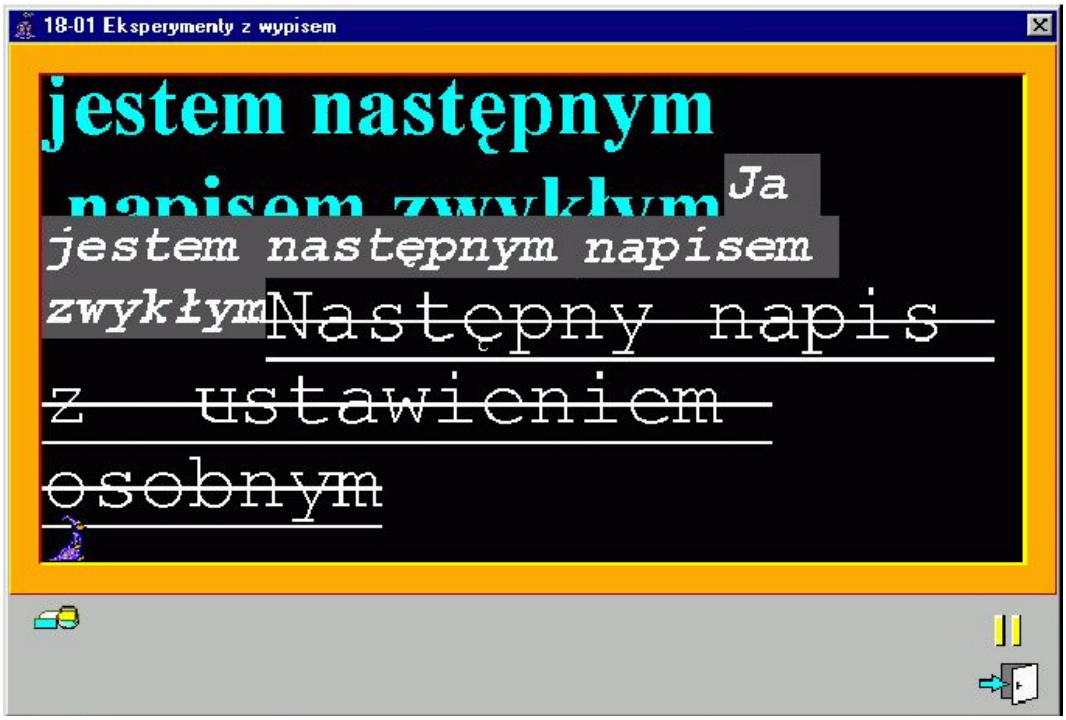
Rysunek 18.5
Program do eksperymentów z ustawieniem wyświetlania na ekranie

Teraz wszystko wypróbujemy. Wpiszmy program z rysunku 18.5 i popatrzmy, jaki wpływ mają poszczególne elementy polecenia na wygląd wyświetlanego tekstu.

1. Polecenie w pierwszym wierszu tylko ustawia format (wygląd) standardowego wyświetlania. Ustawi czcionkę**Times New Roman CE** o wielkości **40** punktów, **pogrubioną** (litera **B**), **błękitną**. Jak można sprawdzić podczas czekania na naciśnięcie klawisza, wykonanie tego polecenia nie zostawi od razu efektów na ekranie.
2. Drugi wiersz wyświetli ciąg **Ja jestem wypisem zwykłym** dokładnie tak, jak zostało to ustawione przez pierwsze polecenie i poczeka na naciśnięcie klawisza. Dalej już nie będziemy wspominać o czekaniu na klawisz, które kończy każdy wiersz.
3. Trzeci wiersz wyświetli tekst **Osobne nastawienie** według osobistego formatowania, tzn. **żółtą** czcionką **Arial** o wielkości **16** punktów.
4. Czwarty wiersz wyświetli tekst **Ja jestem następnym wypisem zwykłym**. Nie ma swojego osobistego ustawienia, więc będzie dla niego obowiązywać ustawienie standardowe wprowadzone w pierwszym wierszu.
5. Piąty wiersz ustawia nowy wygląd standardowego wyświetlania. Od tej chwili tekst będzie wyświetlany **białą** czcionką **Courier New** o wielkości **24** punktów, **pogrubioną kursywą** (litery **BI**). Wyświetlanie będzie się odbywało na ciemnoszarym tle.
6. Szósty wiersz wyświetli tekst **Ja jestem następnym wypisem zwykłym** w formacie ustawionym w poprzednim wierszu.
7. Siódmy wiersz wyświetli tekst **Następny wypis z nastawieniem osobnym.** Ponieważ brak ustawienia nowej czcionki, tekst zostanie wyświetlony czcionką według standardowego ustawienia, tzn. czcionką **Courier New**. Ustawiono wielkości czcionki (**36** punktów) i dodatkowych parametrów **US** – będzie więc wyświetlony czcionką podkreśloną i przekreśloną. Nie jest ustawiona czcionka pogrubiona ani kursywa. Brak też koloru więc będzie użyty kolor biały z ostatniego ustawienia standardowego.
8. Ostatnie, ósme polecenie wyświetli tekst **Ja jestem znów tekstem zwykłym** w formacie standardowym, który został ustawiony w piątym wierszu. Końcowy wyglądu dworku Baltiego można zobaczyć na rysunku 18.6.

Może dziwić że jeden napis nakłada się na drugi. Wyjaśnienie jest proste. Podczas kolejnego wyświetlania tekstu Baltiego już nie pamięta, gdzie wyświetlał ostatni napis. Przy przejściu do nowego wiersza przesunie się w dół zgodnie z wielkością obecnie używanej czcionki. Jeśli bieżąca czcionka jest mniejsza od tej, której użyto ostatnio, następny wiersz będzie nakładać się na poprzedni wiersz tak, jak widać w piątym i szóstym wyświetlonym wierszu na rysunku.

Zachęcamy do wypróbowania innych możliwości. Nie zapominajmy przy tym o możliwości wyświetlania przezroczyście i z animacją, o których mówiliśmy już w rozdziale *Trzy sposoby wyświetlania*.



Rysunek 18.6
Wygląd ekranu po wyjściu z programu z rysunku 18.5

18.3 Kursor wyjścia

Wyświetlając na ekranie najróżniejsze teksty mogliśmy dotąd wybierać jedną z dwóch możliwości:

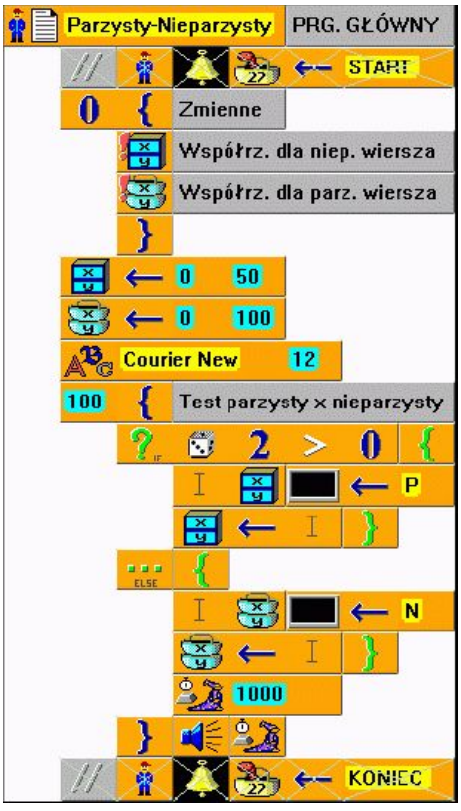
- albo kolejne napisy będą wyświetlane kolejno po sobie, lecz nie będziemy mogli określić, gdzie to ma być,
- albo będziemy mogli określić pozycję wyświetlania tekst, lecz wtedy pozycję, na której chcemy wyświetlić następny tekst, musimy obliczyć sami.

Rysunek 18.7
Element Kursor wyjścia

Teraz pokażemy jak określić współrzędne wyświetlania tekstu wyjścia i przy tym nie zastanawiać się nad tym, czy następny tekst zostanie wyświetlony na kolejnej pozycji na ekranie. Pomoże nam element **Kursor wyjścia**, który znajduje się na lewo na dole pod elementem **Literał** (patrz rysunek 18.7). Kursor ten określa pozycję wyświetlania na ekranie tekstu, dla którego nie określimy współrzędnych.

Współrzędne kursora tekstowego można pobrać i ustawić w zwykły sposób. Możemy je zapamiętać, możemy nawet wyświetlić kilka napisów w różnych miejscach ekranu i za każdym razem tylko ustawiać pozycję kursora na ostatnie współrzędne ważne dla danego napisu.

Przygotowaliśmy mały program żeby wszystko wypróbować. Pozwoli nam przy okazji sprawdzić wewnętrzny generator liczb losowych Baltiego (patrz rysunek 18.8). Zmusimy go do generowania losowo liczb 0 lub 1 i będziemy obserwować, jak często pojawia się każda liczba w serii prób. Program testuje częstość pojawiania się poszczególnych liczb wyświetlając w jednym wierszu wielką literę **P** przy wylosowaniu liczby zero a w drugim wierszu dużą literę **N** dla wylosowanej liczby jeden. Możemy w sposób



Rysunek 18.8
Test Parzysty – Nieparzysty

ciągły obserwować jak poszczególne wiersze ”ścigają się” o to, który będzie pierwszy przy prawej krawędzi.

Program został nazwany *Parzysty – Nieparzysty*. Polecamy samodzielne przekonstruowanie go do postaci wypisującej **P** przy wylosowaniu liczby parzystej a **N** przy wylosowaniu nieparzystej, np. z przedziału [0, 1000].

Program został zapisany jako pomocnik żebyśmy mogli później skorzystać z możliwości prostego zapamiętania współrzędnych w zmiennych podwójnych i prostego ustawienia współrzędnych. Takie rozwiązanie pozwala na zapamiętanie w szufladzie podwójnej współrzędnych kursora jednego wiersza a współrzędnych kursora drugiego wiersza w koszyku podwójnym.

Na początku przydzielimy zmiennym podwójnym współrzędne początku wiersza i ustawimy czcionkę. Potem „rzucamy kostką” (generujemy liczbę losową) i jeśli wynikiem jest numer 1 wyświetlamy następny znak w wierszu dla parzystych, jeśli wynikiem jest zero wyświetlamy znak w wierszu dla nieparzystych. Po wyświetleniu zmuszamy komputer, by chwilkę poczekał, abyśmy mogli zobaczyć wynik (i żeby było ciekawiej).

Dla uproszczenia program napisano tak, że generowanie liczb losowych (i cały program) zatrzyma się po 100 wykonaniach. Jednak nie byłoby dużym problemem zmienić program tak, by generator zatrzymał się dopiero w chwili, gdy dowolny wiersz dotrze do prawej krawędzi dworku Baltiego.

Tak zmodyfikowanego programu można używać jako prymitywnej wersji wyścigów. Każdy wiersz przedstawiałby jednego zawodnika i można podczas „wyścigów” przyjmować zakłady, który wiersz „dobiegnie” do celu wcześniej. Równie proste byłoby zapewne dostosowanie programu dla większej liczby startujących, jednak utracilibyśmy pewnie całą elegancję użycia zmiennych podwójnych.

18.4 Czego nauczyliśmy się

W tym rozdziale nauczyliśmy się określać wygląd tekstów wyświetlanych na ekranie. Potrafisz już ustawić zestaw znaków, wielkość i kolor czcionki, jego styl i kolor tła. Wiemy jak ustawić parametr dla wszystkich następnych napisów i jak sprawić, by jeden konkretny napis używał swoich osobistych parametrów. Na końcu rozdziału pokazaliśmy, jak można użyć kursora wyjścia do lepszego ustawienia napisów na ekranie.

19. Wprowadzanie danych z klawiatury

Czego nauczymy się w tym rozdziale

W tym rozdziale pokażemy, jak program może otrzymać od użytkownika dane wprowadzane za pomocą klawiatury. Najpierw opowiemy o tym, jak spowodować, by wszystkimi operacjami podczas wprowadzania danych przez użytkownika zajął się system, który na końcu odda nam cały wprowadzony ciąg znaków. Potem nauczymy się pobierać dane jeden znak po drugim oraz jak wykorzystać czas czekania na wprowadzenie następnego znaku.

Jak dotąd wszystkie nasze programy musiały z góry znać wszystkie dane. Gdy potrzebowaliśmy ustawić jakąś wartość, ustawialiśmy ją za pomocą stałych. Tak było np. w przykładzie z sadzeniem kwiatów na wyspie w środku basenu. Jedynym wyjątkiem od tej reguły była możliwość wybrania przedmiotu, którą wypróbowaliśmy w programie pozwalającym na grę w kółko i krzyżyk. Teraz pokażemy, jak podczas pracy programu wprowadzić do komputera teksty i liczby w najprostszy sposób, z klawiatury.

Pokażemy dwa podstawowe sposoby pracy na danych wejściowych. Najpierw wyjaśnimy prostszy, podczas którego komunikacją z użytkownikiem zajmuje się system oddający nam na końcu wprowadzoną wartość tekstową lub numeryczną. Potem opowiemy o bardziej wyrafinowanym sposobie, gdy otrzymujemy od użytkownika jeden znak po drugim. Ta metoda pozwoli nam użyć efektownego formatowania, poza tym możemy zrobić mnóstwo rzeczy między poszczególnymi naciśnięciami klawiszy.

19.1 Wprowadzenie całego ciągu danych z wejścia

Zaczniemy od sposobu, przy którym nie musimy zajmować się sposobem pobierania naciskanych przez użytkownika klawiszy. Po prostu rozkażemy Baltiemu, by otrzymał od użytkownika dane wejścia i oddał je nam do dyspozycji. Użyjemy do tego elementu **Czytaj numer lub ciąg z klawiatury,** który znajduje się obok lewej krawędzi panelu elementów pod elementem **Dokąd** – patrz rysunek 19.1 (dalej skrócimy nazwę tego elementu do **Czytaj z klawiatury**).

Rysunek 19.1
Element Czytaj numer lub ciąg znaków z klawiatury

Kiedy podczas wykonywania programu Baltie dojdzie do tego polecenia, pokaże w swoim dworku małe prostokątne okienko (dalej nazwiemy go **polem wstępnym**), w którym oczekuje od użytkownika wprowadzenia danych. Użytkownik wpisuje dane w tym okienku.

Jeśli jednak pole wprowadzania danych pojawi się bez jakiejkolwiek informacji, o co Balti prosi, użytkownik może się nie zorientować, że program czegoś od niego chce. Program powinien więc najpierw poprosić użytkownika o wprowadzenie danych i dopiero potem zaoferować pole, do którego użytkownik wprowadzi odpowiednie dane.

Jeśli nie ustalimy inaczej Baltie umieści pole wprowadzania danych w lewym górnym rogu dworku, tzn. na współrzędnych (0 , 0). Jednak czasem nie jest to optymalne miejsce. Na szczęście element **Czytaj z klawiatury** pozwala nam wprowadzić za sobą w nawiasach współrzędne położenia pole wprowadzania danych. Powinniśmy tylko pamiętać, że wszystkie parametry wejścia powinny być zamknięte w nawiasach zaraz za elementem **Czytaj z klawiatury.**

Na rysunku 19.2 mamy prosty program pokazujący, jak komunikacja z użytkownikiem wygląda w rzeczywistości. Program spyta użytkownika o interesujący go numer banku i po wprowadzeniu liczby pokaże na ekranie zawartość odpowiedniego banku.

Rysunek 19.2
Pokaż określony bank

Często chcemy użytkownikowi przygotować domyślną wartość, którą mógłby po prostu zatwierdzić lub tylko trochę zmodyfikować. Cała wartość byłaby wprowadzona tylko w przypadku, gdy chciałby wprowadzić zupełnie inną wartość, niż była przygotowana. Osiągamy to dodając ciąg znaków z wartością domyślną do nawiasów za elementem **Czytaj z klawiatury**. Jeśli chcemy wprowadzić liczbę powinniśmy najpierw dokonać jej konwersji na ciąg znaków.

Wprowadzając wartość domyślną określamy też wielkość pola wejścia – będzie ono tak duże, by zmieściło się w nim tyle znaków, ile zawiera ciąg z wartością domyślną. Jednak może nam to przeszkadzać,

gdy zechcemy umożliwić użytkownikowi wprowadzenie tekstu dłuższego od wartości domyślnej. Możemy dodać do wartości domyślnej parę odstępów, możemy też skorzystać z możliwości wprowadzenia kolejnego parametru polecenia **Czytaj z klawiatury** – liczby znaków, które powinny zmieścić się w polu wprowadzania danych.

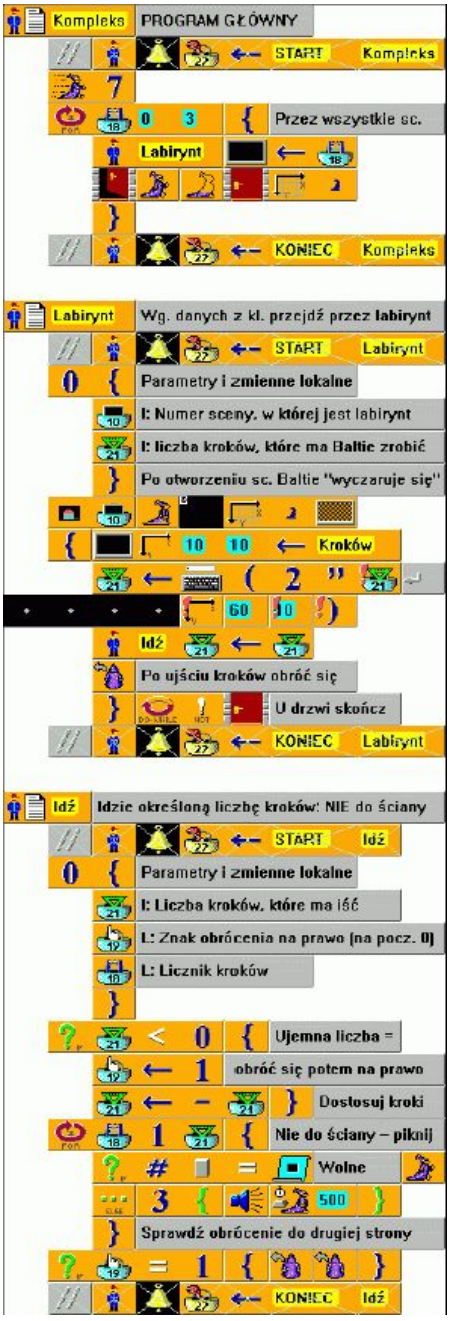
Spróbujmy teraz napisać program, w którym użytkownik poda, o ile pól ma się przesunąć Baltie w kierunku, w którym jest obrócony. Po odpowiedniej liczbie kroków Baltie wykona „w lewo zwrot" i ponownie zapyta o liczbę kroków. Zaproponuje jednocześnie użytkownikowi ostatnio wprowadzoną wartość. To działanie będzie powtarzać w pętli nieskończonej. Gdy program będzie gotowy można porównać rozwiązanie z programem z rysunku 19.3.

Rysunek 19.3
Baltie idzie tyle pól ile wprowadzono z klawiatury

Oto do przemyślenia zmienione zadanie: zaprogramujmy grę przechowującą w poszczególnych scenach labirynty, przez które Baltie powinien przejść. Wyjście z labiryntu zawsze będzie oznaczone drzwiami.

Użytkownik po kolei podaje liczbę kroków, które Baltie ma wykonać do następnego obrotu. Jeśli wprowadzi za dużo kroków Baltie zatrzyma się przed ścianą i chwilę będzie pikał. Po wykonaniu zadanej liczby kroków Baltie obróci się w lewo. Jeśli zostanie wprowadzona ujemna liczba kroków Baltie po ich przejściu obróci się na prawo. Jeżeli gracz potrzebuje tylko obrotu może wprowadzić 0 kroków. Jeżeli Baltie po wykonaniu obrotu znajdzie się przed drzwiami i będzie obrócony do nich czołem, otworzy drzwi i przejdzie do następnej sceny z kolejnym labiryntem.

Warto spróbować przygotować ten program samodzielnie. Rozwiązanie znajduje się na rysunku 19.4. Można dodać do gry mierzenie czasu podróży, co umożliwi grę z przyjaciółmi. W przykładowym rozwiązaniu pominięto ten element, bo i tak program nie mieści się na stronie i został zmniejszony bardziej niż zwykle. Stoper dodamy dopiero w następnej wersji programu.



Rysunek 19.4
Wędrowanie wśród labiryntów

19.2 Wprowadzanie wartości znak po znaku

Drugą możliwością otrzymywania danych z klawiatury jest czytanie wprowadzanych danych znak po znaku. Przedtem powinniśmy jednak opowiedzieć trochę o tym, jak komputer w rzeczywistości posługuje się klawiaturą.

Za każdym razem gdy na klawiaturze naciśniemy lub zwolnimy klawisz, klawiatura wysyła sygnał o tym zdarzeniu do komputera. Jeżeli trzymamy jakiś klawisz naciśnięty dłużej klawiatura będzie wysyłać do komputera wiadomość o naciśnięciu klawisza wielokrotnie przez cały czas, dopóki nie zwolnimy go lub nie naciśniemy innego. Szybkość tego powtarzania można ustawić nawet na 30 powtórzeń na sekundę. Domyślne jest ustawienie 10 powtórzeń.

Po otrzymaniu wiadomość o naciśnięciu klawisza komputer sprawdzi, które przełączniki są włączone i popatrzy w swoje tablice by sprawdzić, jakie znaczenie ma ten klawisz przy takim ustawieniu przełączników. Informację o znaczeniu klawisza schowa do specjalnej kolejki (bufora), gdzie czeka na moment, gdy program będzie ich potrzebować.

Gdy program sprawdza, co użytkownik wprowadził z klawiatury, patrzy do bufora klawiatury. Stąd odbiera potrzebne informacje. Jeśli w buforze nie ma żadnej informacji, są dwie możliwości: albo program na nią

poczeka, albo będzie robił coś innego i sprawdzi bufor za chwilę.

Używane elementy

Pokażemy teraz, że wszystko działa tak jak to opisaliśmy. Najpierw jednak jeszcze kilka ważnych elementów, których będziemy potrzebować do pracy z klawiaturą. Wszystkie pokazano na rysunku 19.5. Od lewej strony są to kolejno elementy: **Czytaj klawisz lub przycisk myszy bez czekania, Czytaj klawisz lub przycisk myszy z czekaniem, Wyczyść bufor klawiszy, Dowolny klawisz** i **Wprowadzony znak**. Ponieważ nazwy pierwszych dwóch elementów są za długie, będziemy nazywać je dla uproszczenia **Czytaj bez czekania** i **Czytaj z czekaniem**.

Rysunek 19.5
Elementy używane do czytania znaków

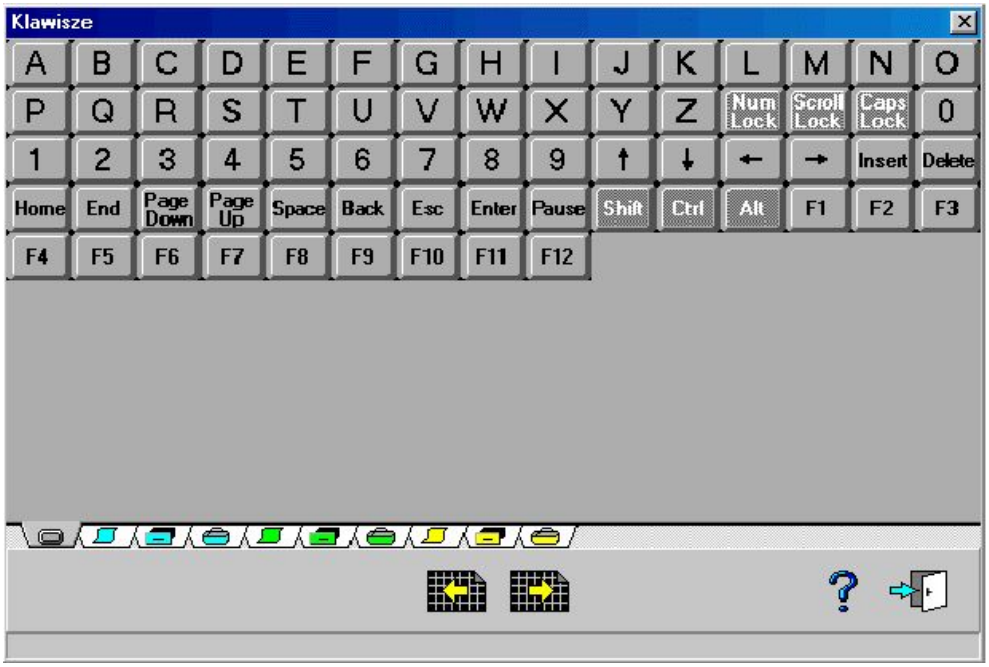
Element **Czytaj bez czekania** sprawdza bufor klawiatury i jeśli znajdzie tam coś, przydziela numer naciśniętego klawisza do zmiennej **Dowolny klawisz** a wprowadzony znak do zmiennej **Wprowadzony znak**. Jeśli w buforze nie znajdzie żadnej informacji przydziela do zmiennej **Dowolny klawisz** zero a do zmiennej **Wprowadzony znak** znak specjalny o kodzie 0.

Element **Czytaj z czekaniem** również patrzy do bufora klawiatury i jeśli tam coś znajdzie zachowuje się tak jak element **Czytaj bez czekania**. Jeżeli bufor jest pusty, „usiądzie" i zaczeka aż w buforze coś się pojawi. Chwyci informację, gdy tylko się pojawi i pomknie umieścić ją w odpowiednich zmiennych.

Element **Wyczyść bufor klawiszy** zachowuje się zgodnie z nazwą. Używamy go aby dowiedzieć się, jaki klawisz użytkownik właśnie nacisnął, ale jednocześnie nie obchodzą nas informacje wprowadzone wcześniej do bufora i dotychczas nie odczytane. Uważajmy na to, że czyszcząc bufor klawiatury tracimy wszystkie informacje umieszczone być może w buforze. Nie będzie już możliwości dotarcia do nich.

Element **Wprowadzony znak** to specjalna zmienna systemowa. Symbolizuje to mała szufladka w lewym górnym rogu rysunku na elemencie. Element ten przechowuje ostatnio wprowadzony znak. Nie można w niej zapisać żadnej wartości inaczej niż naciśnięciem klawisza lub przycisku myszy i przeczytaniem zawartości bufora klawiatury.

Podobnie jest z elementem **Dowolny klawisz**. Z tej zmiennej podczas wykonywania programu można informacje tylko czytać. Zapisywane są do zmiennej tylko informacje wysłane przez bufor klawiatury. Zmienna **Dowolny klawisz** przechowuje numer ostatnio naciśniętego klawisza. Jest to w rzeczywistości numer odpowiedniego klawisza w banku klawiszy. Bank ten znajduje się obok banków stałych i zmiennych – patrz rysunek 19.6.

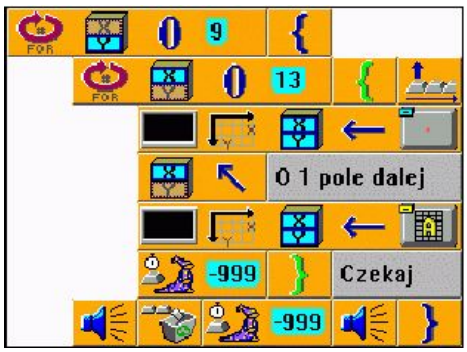


Rysunek 19.6
Bank klawiszy

Jak widać na rysunku w banku nie ma wszystkich klawiszy. Brakuje klawiszy ze znakami nie *alfanumerycznymi* (znaki *alfanumeryczne* to litery i cyfry) oraz znaków diakrytycznych (ą, ś, ł ...). Ma to swoją przyczynę i pozostaje nam przyjąć ten fakt do wiadomości.

Program do testowania

To na tyle teorii. Popatrzmy teraz na program na rysunku 19.7, który pozwoli nam wypróbować wszystko o czym mówiliśmy powyżej. Program raz na sekundę popatrzy w bufor klawiatury, wyświetli znaleziony klawisz i znak. Jeśli niczego nie znajdzie wyświetli zamiast klawisza rysunek elementu **Dowolny klawisz**. Zamiast znaku nie wyświetli niczego.



Rysunek 19.7
Test czytania znaków z klawiatury

Element **Dowolny klawisz** zostanie wyświetlony również po naciśnięciu klawisza, który nie ma swojego odpowiednika w banku klawiszy.

Po przeprowadzeniu sześciu testów program dotrze do końca wiersza. Piknie, wyczyści bufor klawiatury i poczeka sekundę. Potem ponownie zacznie testować zawartość bufora klawiatury, wyświetlać go będzie w następnym wierszu.

Przed rozpoczęciem kopiowania programu do komputera zwróćmy uwagę na ujemny czas czekania – na rysunku ustawiono -999 milisekund. Pozwoli to na osiągnięcie podczas czekania ignorowania przez Baltiego zdarzeń pochodzących od klawiatury i myszki. Gdybyśmy wprowadzili dodatni czas oczekiwania, każde naciśnięcie klawisza zakończyłoby czekanie i nie dowiedzielibyśmy się niczego o buforze klawiatury.

Wypróbujmy jak program będzie reagować na serię szybko naciśniętych klawiszy, jak będzie reagować na długie naciśnięcie pojedynczego klawisza i jak zareaguje na naciśnięcie dowolnego przycisku myszy. Czas oczekiwania programu możemy dostosować do własnych upodobań. Warto jednak sprawdzić działanie przy liczbie ujemnej.

Program samodzielny

Teraz powinniśmy wypróbować wszystko, czego dowiedzieliśmy się na jakimś sensownym przykładzie. Poprawimy nasz ostatni program, w którym chodziliśmy z Baltiem po labiryncie. Teraz nie będziemy już określać liczby kroków, które Baltie powinien zrobić, ale będziemy sterować nim za pomocą strzałek kursora. Będziemy przy tym mierzyć czas potrzebny do przejścia całego systemu labiryntów. Dla utrudnienia umieścimy w labiryncie kilka budzików, które gracz powinien pozbierać klikając je myszą. Za każdy zebrany budzik odejmiemy graczowi jedną sekundę od zmierzonego czasu.

Zanim zaczniemy poprawiać samodzielnie program jeszcze dwie notatki:

Mierzenie czasu

Aby dać graczowi premię za każdy usunięty budzik, musimy zmienić czas na stoperze. Zrobimy to po prostu ustawiając na stoperze nowy czas obliczony poprzez odjęcie od czasu na stoperze przydzielanej premii – w naszym przypadku jedna sekunda.

Rysunek 19.8
Zmiana czasu na stoperze

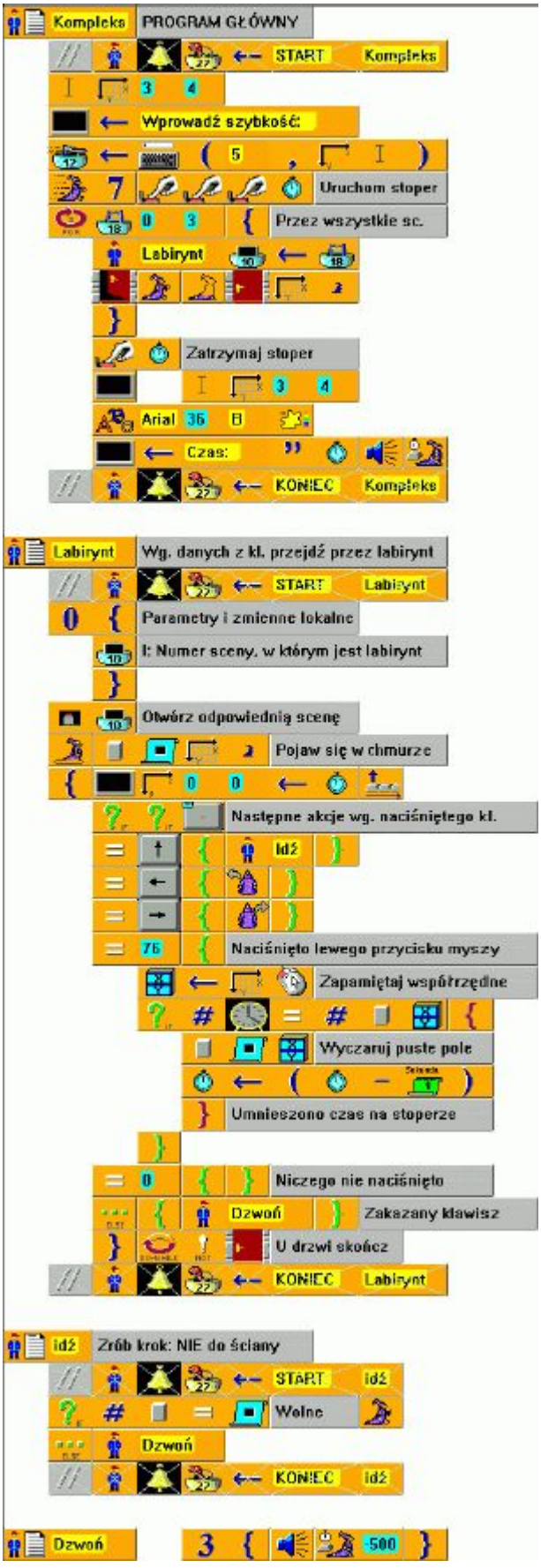
Tu jednak dochodzimy do drobnego problemu. Stoper nie przechowuje czasu jako liczby upływających milisekund lecz jako ułamek dnia. Oznacza to, że po tygodniu na stoperze będzie liczba 7, jeden dzień pamiętany jest jako liczba 1. Sześć 6 godzin to liczba 0,25 (6 godzin jest jedną czwartą dnia). Ponieważ dzień ma 86 400 sekund jedna sekunda będzie przechowana na stoperze jako 1/86400, tzn. około 0,000011574 (błąd zaokrągleń wynosi ok. 0,5 sekundy na dzień).

Zdefiniujmy więc stałą rzeczywistą (zieloną) i przydzielmy jej wartość 0.000011574

UWAGA! Baltie zamiast przecinka dziesiętnego używa kropki! Dlatego liczba wygląda tak a nie inaczej. Tej stałej będziemy używać do zmniejszania wartości na stoperze o jedną sekundę.

Bufor klawiatury

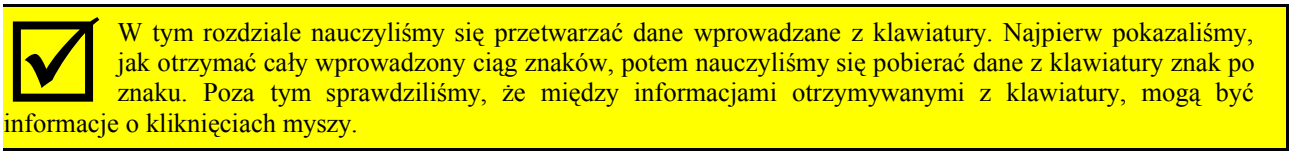
Druga uwaga dotyczy bufora klawiatury. Jak można było zauważyć przeglądając opis elementów obsługujących bufor, Baltie sprawdza w buforze klawiatury nie tylko naciśnięcia klawiszy lecz także kliknięcia myszy. Kliknięcia nie są jednak ustawiane w buforze jedno po drugim (jak klawisze) lecz w buforze pozostaje tylko ostatnie kliknięcie. Może się zdarzyć, gdy nie będziemy sprawdzać bufora wystarczająco szybko, że niektóre kliknięcia zostaną zagubione. Proponowana wersja programu pamięta o tym i celowo ustawia małą szybkość Baltiego. Gracz może więc namyślić się podczas wędrówki w jaki sposób będzie naciskać strzałki kursora a gdzie będzie klikać myszą.



Rysunek 19.9

Przechodzenie labiryntu sterowane strzałkami kursora

19.3 Czego nauczyliśmy się



W tym rozdziale nauczyliśmy się przetwarzać dane wprowadzane z klawiatury. Najpierw pokazaliśmy, jak otrzymać cały wprowadzony ciąg znaków, potem nauczyliśmy się pobierać dane z klawiatury znak po znaku. Poza tym sprawdziliśmy, że między informacjami otrzymywanymi z klawiatury, mogą być informacje o kliknięciach myszy.

20. Zakończenie – co dalej

Kończy się nasza przygoda z podstawami programowania. Nauczyliśmy się wielu rzeczy dotyczących programowania. Po uważnej lekturze i samodzielnym rozwiązaniu zadań nie można już czuć się nowicjuszem. Uzyskaliśmy umiejętności nie znane większości innych ludzi. Jeśli praca z Baltiem okazałą się naprawdę interesująca. mamy sporo możliwości, by dowiedzieć się więcej. Jedną z najlepszych jest skorzystanie ze strony WWW firmy SGP – jej adres to ***http://www.sgp.cz***. Oprócz tego, że można stąd pobrać nowe wersje programu Baltie, znajdziemy tu najnowsze informacje o wszystkim, dotyczącym Baltiego.

Na tej stronie czekają również wzorowe przykłady, które pokazują, jak można rozwiązywać różne problemy. Ponadto są to podręczniki elektroniczne i kursy programowania. Na pewno każdego ucieszy lista najlepszych programów, które można za darmo pobrać i wypróbować. Po ulepszeniu programu można wysłać go z powrotem – taka wersja może zająć jeszcze lepsze miejsce. Swoje programy możemy porównywać z programami z całego świata, bo Baltie już znalazł swoje miejsce na wszystkich kontynentach.

Może zainteresują nas najróżniejsze rywalizacje, które są skierowane zarówno do nowicjuszy, jak i dla zaawansowanych programistów. Najciekawszy jest międzynarodowa konkurs twórczości, którego finaliści co roku pokazują swoje programy na międzynarodowych targach technologii informacyjnych i komunikacyjnych Invex a zwycięzcy poszczególnych kategorii mogą nawet wygrać notebooka. Warto włączyć się do światowej sieci Baltie.

21. Słownik wyrazów obcych

Analiza – zbadanie zjawiska, wyodrębnienie jego składników, części.

Bonus – premia, nadzwyczajna nagroda.

Debugowanie – proces wyszukiwania i usuwania błędów w programie.

Deklaracja – określenie właściwości. Na przykład w deklaracji zmiennej określa się, jaki będzie miała typ. Dobrym zwyczajem jest dodać przy deklaracji informację o tym, do czego zmienne będzie używana.

Dekompozycja – rozkład na mniejsze elementy składowe.

Dyspozycja – mieć coś do dyspozycji = mieć możliwość używać tego, móc tego używać.

Ekwiwalent – odpowiednik, coś, co ma takie samo znaczenie czy wartość.

Fikcyjny – wymyślony, nierzeczywisty.

Generator – coś co tworzy. Generator liczb losowych tworzy liczby losowe.

Globalny – ogólnie znany, dostępny. W programowaniu oznacza obiekty znane w całym programie.

Inicjalizujący – ustawiający stan początkowy.

Interpretować – tłumaczyć, wyjaśniać.

Konwencje – zwyczaje, ustalone reguły postępowania.

Lokalny – miejscowy, istniejący w szczególnym miejscu, związany z określonym miejscem. W programowaniu oznacz obiekty znane i dostępne tylko w pewnej części programu.

Modyfikacja – zmiana.

Specyfikować – określić.

Indeks

Indeks alfabetyczny