Gliwice, 15.06.2016

Laboratorium Programowania Komputerów

Temat:
Gra labirynt

Autor: Rafał Suszka
Informatyka, semestr II,
Grupa 5, sekcja 2
Prowadzący: mgr inż. Piotr Pecka

1. Treść zadania

Napisać grę, w której zadaniem użytkownika będzie chodzenie po labiryncie, zbieranie przedmiotów oraz unikanie ruchomych przeciwników. Jednym z elementów mapy powinno być wyjście, które skieruje użytkownika do kolejnego poziomu. Poziomy powinny być zapisywane w osobnych plikach; program powinien obsługiwać możliwość dodawania nowych poziomów. Program musi zapamiętywać oraz zapisywać w pliku wszystkie wyniki uzyskane przez graczy.

2. Analiza, projektowanie

2.1 Algorytmy, struktury danych, ograniczenia specyfikacji

W programie wykorzystano dwie struktury. Jedna z nich przechowuje informacje o obecnym etapie a druga o aktualnej grze. Głównym elementem programu jest dwuwymiarowa tablica alokowana dynamicznie, w której przechowywany jest etap aktualnie wczytany do programu. Jest to najłatwiejszy sposób do przechowywania oraz modyfikowani etapu podczas pracy programu

Program pracuje na tablicy dwuwymiarowej. Powoduje to konieczność częstego wyszukiwania w niej odpowiednich elementów, dlatego dominującym algorytmem jest algorytm przeszukiwania tablicy. Następuje on przez przejście po kolei przez wszystkie komórki tablicy aż do znalezienia właściwego elementu. W przypadku pesymistycznym czas może wynieść *O(n)*.

Program ma pewne ograniczenia specyfikacyjne. Nazwy etapów oraz graczy są przechowywane w tablicach statycznych co powoduje narzucenie maksymalnej długości ciągu znaków wprowadzonego przez gracza. Wynosi on 19 znaków.

2.2 Analiza problemu, podstawy teoretyczne

Program nie korzysta z żadnych skomplikowanych wzorów, wykonuje jedynie podstawowe operacje na tablicy dwuwymiarowej.

3. Specyfikacja zewnętrzna

3.1 Obsługa programu

Program jest uruchamiany z linii poleceń. Na początku zostaje wyświetlone główne menu programu. Do każdej opcji przypisana jest cyfra (od 1 do 4). Aby przejść w odpowiednie miejsce należy wpisać cyfrę odpowiadającą odpowiedniej opcji oraz zatwierdzić ją klawiszem ‘Enter’. Obsługa poszczególnych części programu:

1. „Nowa gra”
   Po wybraniu powyższej opcji program wyświetli krótką instrukcję dotyczącą rozgrywki. Po zapoznaniu się z nią należy podać nazwę gracza (imię). Uwaga nie może być ono dłuższe niż 19 znaków! Podanie dłuższej nazwy spowoduje nieprawidłowe działanie programu. Wybór zatwierdzamy klawiszem ‘Enter’.

Następnie na ekranie pojawia się lista dostępnych etapów. Wybieramy z niej jeden, przepisując jego nazwę. Wybór zatwierdzamy klawiszem ‘Enter. Podanie błędnej nazwy spowoduje zamknięcie programu. Po wybraniu etapu gra się uruchamia. Sterowanie postacią (a) polega na naciskaniu „strzałek” na klawiaturze. Celem gry jest zbieranie punktów (.), (*), unikanie przeciwników (x) – kontakt z nimi powoduje zakończenie gry (program wychodzi do głównego menu) oraz dojście do mety (M) – możliwość wybrania kolejnego etapu.

2. „Tworzenie etapów”
   Po wybraniu powyższej opcji program wyświetli krótką instrukcję dotyczącą budowania etapów. Po zapoznaniu się z nią należy podać szerokość oraz długość mapy (w znakach). Wybór zatwierdzamy klawiszem ‘Enter’. Następnie należy wprowadzać ciągi znaków według poniższej tabelki każdorazowo zatwierdzać klawiszem ‘Enter’ na końcu wiersza.

l	ograniczenia etapu (bariery) w pionie i poziomie
a	postać, symbolu można użyć tylko raz
x	przeciwnicy
.	przedmiot za 1 punkt
*	przedmiot za 5 punktów
M	meta, symbolu można użyć tylko raz

   Uwaga: nie można zostawiać pustych pól.
   Po wprowadzeniu etapu należy wpisać jego nazwę. Musi się ona kończyć na **.txt**. Uwaga nie może być ona dłuższa niż 19 znaków! Podanie dłuższej nazwy spowoduje nieprawidłowe działanie programu.
3. „Tabela wyników”
   Po wybraniu powyższej opcji program wyświetli tabelę wyników wszystkich graczy.
4. „Wyście”
   Wybranie tej opcji spowoduje zamknięcie programu.

3.2 Format danych wejściowych

Wszystkie liczby wprowadzane w programie powinny być naturalne oraz dodatnie.
Inne dane wejściowe:

imię	ciąg znaków (nie zawierający polskich liter), nie dłuższy niż 19 znaków
nazwa etapu	ciąg znaków w formacie: nazwa.txt. Całość ciągu nie może przekraczać 19 znaków, nie może też zawierać polskich liter.

3.3 Komunikaty

L.p.	Komunikat	Objaśnienie
1.	„Witaj w grze LABIRYNY! Wybierz co chcesz zrobic i zatwierdz to klawiszem Enter 1. Nowa gra 2. Tworzenie etapow 3. Tabela wynikow 4. Wyjscie”	Informacja dotycząca obsługi menu programu. Należy wybrać jedną z opcji (przypisany do niej numer wpisać w konsolę) oraz zatwierdzić klawiszem ’Enter’

2.	„Celem gry jest dotarcie postacia ('a') do mety ('M') Po drodze nalezy zbierac przedmioty: - ('.') - 1 punkt - ('*') - 5 punktow oraz unikac przeciwnikow ('x') - po starciu z nim postac ('a') umiera. Postacia steruje sie za pomoca 'strzalek' na klawiaturze. Meta przekierowuje do kolejnego etapu. Podane przez uzytkownika imie nie moze przekraczac 20 znakow. Podaj swoje imie.”	Instrukcja dotycząca sposobu gry. Po zapoznaniu się z nią należy Podać swoje imię (nie dłuższe niż 19 znaków) oraz zatwierdzić je klawiszem ‘Enter’.
3.	„Lista nazw etapow: ……...”	Informacja o nazwa etapów już użytych/możliwych do wybrania przez użytkownika.
4.	„Wybierz etap z powyzszej listy lub nacisnić 'c' aby cofnac.”	Informacja o konieczności wybrania etapu z listy. Możliwość cofnięcia się do menu programu po wybraniu klawisza „c” oraz naciśnięcia klawisza ‘Enter’.
5.	„Brak etapow na liscie.”	Informacja o braku etapów w programie. Rozwiązanie: należy stworzyć etap za pomocą opcji „Tworzenie etapów”
6.	„Podaj swoje imie.”	Informacja o potrzebie podania imienia. Po wpisaniu z klawiatury należy zatwierdzić go klawiszem ‘Enter’.
7.	„Podaj szerokosc mapy.” „Podaj dlugosc mapy.”	Informacja dla użytkownika, że należy podać szerokość/długość tworzonego etapu. Zatwierdzenie wyboru klawiszem ‘Enter’.
8.	„Wybrales bledna opcje. Sprobuj ponownie”	Informacja o wybraniu opcji nie istniejącej w programie. Zalecenia: należy wybrać prawidłową opcję.
9.	„GAME OVER”	Informacja o przegranej grze. Program wyświetli menu.
10.	„Ukonczyles etap!”	Informacja o przejściu etapu, gra przekieruje użytkownika do wyboru kolejnego etapu.
11.	„Wybrales tworzenie etapow Na poczatku tworzenia etapu nalezy podac jego wymiary. Nastepnie nalezy podac ciagi znakow odpowiadajace tworzonemu etapowi oraz zatwierdzac je klawiszem 'enter'. Mozna uzywac nastpujacych symboli:	Informacja dotycząca sposobu tworzenia nowych etapów.

	- ('|') - ograniczenia etapu (bariery) w pionie i poziomie - ('a') - postac, symbolu mozna uzyc tylko raz! - ('x') - przeciwnicy - ('.') - przedmiot za 1 punkt - ('*') - przedmiot na 5 punktow - ('M') - meta, symblolu mozna uzyc tylko raz! Na samym koncu nalezy wybrac nazwe etapu (musi konczyc sie na '.txt'). Maksymalna dlugosc nazwy to 20 znakow”	

4. Specyfikacja wewnętrzna

Program został zrealizowany zgodnie z paradygmatem strukturalnym. W programie rozdzielono interfejs (komunikacje z użytkownikiem) od logiki aplikacji (operacje na tablicy dwuwymiarowej).

4.1 Typy danych, zmienne

W programie zdefiniowano następujące typy danych:

```
typedef struct Etap_ //struktura przechowująca etap w pamięci
{
	int szerokosc; //szerokość etapu
	int dlugosc; //dłgugość etapu
	char** etap_wsk; //wskaznik na etap
} Etap;
```

Struktura przechowuje informacje o obecnym etapie (jego szerokość, długość oraz wskaźnik na tablicę gdzie znajduje się w pamięci).

```
typedef struct Gra_ //struktura przechowująca informacje o obecnej grze w pamięci
{
	char nazwa_etapu[20];
	char nazwa_gracza[20];
	int liczba_punktow;
} Gra;
```

Struktura przechowuje informacje o obecnej grze (nazwę etapu, nazwę gracza oraz aktualną ilość zdobytych punktów)

4.2 Funkcje

void instrukcja_do_gry(void)
Funkcja wyświetla na ekranie komunikat z instrukcją dotyczącą gry. Funkcja jest bezparametryczna i nie przyjmuje żadnych wartości.

void instrukcja_tworzenia_etapu(void)
Funkcja wyświetla na ekranie komunikat z instrukcją dotyczącą tworzenia etapu. Funkcja jest bezparametryczna i nie przyjmuje żadnych wartości.

void komunikat(void)
Funkcja wyświetla na ekranie menu programu. Funkcja jest bezparametryczna i nie przyjmuje żadnych wartości.

void komunikat_koncowy(void)
Funkcja wyświetla na ekranie komunikat informujący o ukończeniu etapu. Funkcja jest bezparametryczna i nie przyjmuje żadnych wartości.

void komunikat_koncowy2(void)
Funkcja wyświetla na ekranie komunikat informujący o przegranej grze. Funkcja jest bezparametryczna i nie przyjmuje żadnych wartości.

void wyswietl_etap_c(Etap o_etap, int czysc)
Funkcja wyświetla na ekranie etap znajdujący się w tablicy dwuwymiarowej (**o_etap.etap_wsk)**. Informacje o etapie zostają przekazane parametrem **o_etap.** Parametr **czysc** może przyjmować 2 wartości: **1** – przed wyświetleniem etapu czyszczony jest ekran konsoli Liczba **inna** niż 1 – ekran nie jest czyszczony.

void wyswietl_liste_etapow(void)
Funkcja wyświetla listę nazw etapów znajdującą się w pliku „**log.txt**”. Nie pobiera ona z programu żadnych parametrów (podczas działania otwiera i zamyka plik). W przypadku braku plik informuje o tym użytkownika.

void czysc_pamiec(Etap* o_etap)
Funkcja usuwa z pamięci wczytany etap. Parametr **o_etap** jest wskaźnikiem na strukturę w której zapisany jest etap. Czyszczenie polega na usunięciu z pamięci każdego wiersza alokowanej dynamicznie tablicy dwuwymiarowej.

void tworzenie_etapu(int x, int y)
Funkcja tworzy nowy etap. Z programy pobrane zostają **x** – szerokość etapu oraz **y** – długość etapu. Na ich podstawie dynamicznie zostaje zaalokowana pamięć na tablicę dwuwymiarową mająca przechowywać etap w pamięci. Następnie sczytywane są od użytkownika ciągi znaków opisujące etap (najpierw do bufora, dopiero z niego kopiowane są do tablicy). Bufor również jest alokowany i dealokowany dynamicznie. Następnie wywoływana jest funkcja ***wyswietl_liste_etapow()*.** Użytkownik podaje nazwę pod jaką etap ma być zapisany (nie może przekraczać 19 znaków – tablica statyczna). Funkcja zapisuje etap do pliku tekstowego o podanej nazwie (najpierw wymiary a następnie zawartość tablicy). Na koniec etap jest wyświetlany – ***wyswietl_etap_c(tworzony_etap, 0)*** oraz usuwany z pamięci – ***czysc_pamiec(&tworzony_etap).*** Funkcja wyświetla odpowiednie komunikaty dla użytkownika przy problemach z otwieraniem plików.

void wczytaj_etap(char* nazwa_etapu, Etap* o_etap)
Funkcja wczytuje do struktury **o_etap** etap o nazwie **nazwa_etapu.** Parametr **nazwa_etapu** jest wskaźnikiem na ciąg znaków z nazwą etapu. Parametr **o_etap** jest wskaźnikiem na strukturę docelową (musi być wcześniej utworzona). W pierwszym etapie wczytywane są rozmiary etapu. Na ich podstawie alokowana jest pamięć na tablicę dwuwymiarowa. Następnie jest ona uzupełniana znak po znaku. Uzupełniana jest również struktura **o_etap.** Funkcja dba o otwarcie i zamknięcie pliku oraz wyświetla odpowiednie komunikaty w razie niepowodzenia.

int znajdz_a_K(Etap o_etap)
Funkcja znajduje kolumnę w której znajduje się postać (litera ***a***) w tablicy z etapem. Informacje o etapie przekazywane są parametrem **o_etap.** Tablica ta jest przeszukiwana sekwencyjnie aż do znalezienie odpowiedniej wartości. **Wynik zwracany:** numer kolumny w której znajduje się postać lub wartość **0** gdy nie została znaleziona.

int znajdz_a_W(Etap o_etap)
Funkcja znajduje wiersz w którym znajduje się postać (litera ***a***) w tablicy z etapem. Informacje o etapie przekazywane są parametrem **o_etap.** Tablica ta jest przeszukiwana sekwencyjnie aż do znalezienie odpowiedniej wartości. **Wynik zwracany:** numer wiersza w którym znajduje się postać lub wartość **0** gdy nie została znaleziona.

void ruch_w_gore(Etap* o_etap, int* obecne_polozenie_W, int* obecne_polozenie_K)
Funkcja przesuwa element znajdujący się w tablicy **o_etap.etap_wsk** w wierszu przekazanym przez parametr **obecne_polozenie_W** oraz kolumnie przekazanej parametrem **obecne_polozenie_K** w pole: **o_etap->etap_wsk[*(obecne_polozenie_W)-1][*obecne_polozenie_K]** („o jedną kolumnę wyżej"). W obecnym miejscu zostaje umieszczona spacja. Na koniec zostaje uaktualnione obecne położenie (parametry dotyczące położenie są przekazane przez wskaźnik).

void ruch_w_dol(Etap* o_etap, int* obecne_polozenie_W, int* obecne_polozenie_K)
Funkcja przesuwa element znajdujący się w tablicy **o_etap.etap_wsk** w wierszu przekazanym przez parametr **obecne_polozenie_W** oraz kolumnie przekazanej parametrem **obecne_polozenie_K** w pole: **o_etap->etap_wsk[*(obecne_polozenie_W)+1][*obecne_polozenie_K]** („o jedną kolumnę niżej"). W obecnym miejscu zostaje umieszczona spacja. Na koniec zostaje uaktualnione obecne położenie (parametry dotyczące położenie są przekazane przez wskaźnik).

void ruch_w_lewo(Etap* o_etap, int* obecne_polozenie_W, int* obecne_polozenie_K)
Funkcja przesuwa element znajdujący się w tablicy **o_etap.etap_wsk** w wierszu przekazanym przez parametr **obecne_polozenie_W** oraz kolumnie przekazanej parametrem **obecne_polozenie_K** w pole: **o_etap->etap_wsk[*obecne_polozenie_W][*(obecne_polozenie_K)-1]** („o jeden wiersz w lewo"). W obecnym miejscu zostaje umieszczona spacja. Na koniec zostaje uaktualnione obecne położenie (parametry dotyczące położenie są przekazane przez wskaźnik).

void ruch_w_prawo(Etap* o_etap, int* obecne_polozenie_W, int* obecne_polozenie_K)
Funkcja przesuwa element znajdujący się w tablicy **o_etap.etap_wsk** w wierszu przekazanym przez parametr **obecne_polozenie_W** oraz kolumnie przekazanej parametrem **obecne_polozenie_K** w pole: **o_etap->etap_wsk[*obecne_polozenie_W][*(obecne_polozenie_K)+1]** („o jeden wiersz w prawo"). W obecnym miejscu zostaje umieszczona spacja. Na koniec zostaje uaktualnione obecne położenie (parametry dotyczące położenie są przekazane przez wskaźnik).

int sterowanie(Etap* o_etap, Gra* n_gra)
Funkcja odpowiada za ruch postaci po mapie (etapie). Do funkcji przekazywany jest wskaźnik na strukturę z obecnym etapem **o_etap** oraz wskaźnik na strukturę z obecną grą **n_gra.** Na początku każdego ruchu (w nieskończonej pętli) sczytywany jest znak z klawiatury za pomocą dwukrotnego wywołania funkcji **_getch()**. Jeżeli wyryje ona „strzałki" (kod **224** a następnie: 72,75,77,80) następuje dalsze działania funkcji, przeciwnym wypadku znaki sczytywane są do skutku. Przed zmianą położenia elementu za pomocą jednej z funkcji ruchu (**ruch_w_gore, ruch_w_dół, ruch_w_lewo, ruch_w_prawo),** sprawdzana jest możliwość ruchu na to pole (kolejno: czy nie następuje wyciek pamięci oraz czy nie znajduje się tam element blokujący ruch („|")). Następnie sprawdzanie jest czy w komórce na którą przemieszczamy element nie znajduje się przeciwnik „**x**", Jeśli tak **n_gra->liczba_punktow** jest ustawiana na 0 oraz wyświetlany komunikat o przegranej grze. W przeciwnym wypadku sprawdzana jest czy w komórce na którą przemieszczamy element nie znajduje meta (za pomocą funkcji (**meta**) jeśli tak gracz wygrywa), podliczana jest ilość punktów (**punkty()**) oraz element zostaje przemieszczony na nowe miejsce. **Wynik zwracany:** wartość **1** w przypadku ukończenia etapu lub wartość **-1** w przypadku porażki. Oba wyniki komentowane są odpowiedniki komunikatami.

int meta(Etap o_etap, int znak, int wiersz, int kolumna)
Funkcja sprawdza czy na polu w tablicy **o_etap.etap_wsk** , na które przesuwa się postać znajduje się meta. Parametr **znak** informuje o kierunku ruchu (możliwe wartości do przyjęcia: 72 ruch w górę, 75 ruch w lewo, 77 ruch w prawo, 80 ruch w dół). Natomiast parametry **wiersz** i **kolumna** informują o obecnym położeniu postaci (kolejno wierszu i kolumnie). **Wynik zwracany:** wartość **1** jeżeli meta znajduje się w komórce tablicy na którą „przesuwa się" postać lub **0** w przeciwnym wypadku.

int punkty(Etap o_etap, int znak, int wiersz, int kolumna)
Funkcja sprawdza czy na polu w tablicy **o_etap.etap_wsk** , na które przesuwa się postać znajduje się przedmioty za które przyznaje się punkty. Parametr **znak** informuje o kierunku ruchu (możliwe wartości do przyjęcia: 72 ruch w górę, 75 ruch w lewo, 77 ruch w prawo, 80 ruch w dół). Natomiast parametry **wiersz** i **kolumna** informują o obecnym położeniu postaci (kolejno wierszu i kolumnie). **Wynik zwracany:** wartość **1** jeżeli w komórce tablicy na którą „przesuwa się" postać znajduje się „**.**", **5** gdy znajduje się „*****". W przeciwnym wypadku zwraca **0**.

int ruch_przeciwnikow(Etap* o_etap, Gra* n_gra)
Funkcja steruje ruchem przeciwników po mapie. Do funkcji przekazywany jest wskaźnik na strukturę z obecnym etapem **o_etap** oraz wskaźnik na strukturę z obecną grą **n_gra.** Na początku losowana jest wartość od **1** do **4** za pomocą funkcji **rand()**. (0 ruch w górę, 1 ruch w lewo, 2 ruch w dół, 3 ruch w prawo). Następnie tablica **o_etap->etap_wsk** jest przeszukiwana. Szukaną wartością jest znak **x** (który oznacz przeciwnika). Przed zmianą położenia elementu za pomocą jednej z funkcji ruchu (**ruch_w_gore, ruch_w_dół, ruch_w_lewo, ruch_w_prawo),** sprawdzana jest możliwość ruchu na to pole (kolejno: czy nie następuje wyciek pamięci oraz czy nie znajduje się tam element blokujący ruch („|"), („**x**"), („*****") lub („**M**")). Funkcja postępuje tak dla wszystkich **x** znalezionych w tablicy. **Wynik zwracany:** wartość **0** jeżeli wszystkie ruchy zakończono powodzeniem lub wartość **-1** gdy któryś z ruchów przemieścił **x** do komórki z graczem („**a**"). W przypadku przegranej zerowana jest również liczba punktów gracza (**n_gra->liczba punktów**).

void zapisz_wyniki(Gra n_gra)
Funkcja zapisuje wynik uzyskany podczas gry (ze struktury przekazanej parametrem **n_gra**) do pliku tekstowego „**wyniki.txt**". Jeżeli plik nie istnieje zostanie utworzony, w przeciwnym wypadku zostaje zaktualizowany. Do pliku kolejno zapisywane są nazwa etapu, nazwa gracza oraz ilość uzyskanych punktów. Funkcja dba o otwarcie oraz zamknięcie pliku, wszelkie problemy zgłasza odpowiednimi komunikatami.

void wyswietl_wyniki(void)
Funkcja wyświetla po kolei rekordami wszystkie wyniki zapisane w pliku „**wyniki.txt**". Początkowo wczytywane są one do buforów a następnie wyświetlane na ekran. Kolejno: nazwa etapu, nazwa gracza oraz ilość uzyskanych punktów. Funkcja dba o otwarcie oraz zamknięcie pliku, wszelkie problemy zgłasza odpowiednimi komunikatami.

5. Testowanie

Program został przetestowany poprzez wielokrotne uruchamianie go i przejście przez wszystkie jego gałęzie. Przetestowano również działanie wszystkich funkcji. Za każdym razem gdy dane były poprawne (wprowadzane zgodnie z instrukcją) program działał prawidłowo.

Program został również sprawdzony pod kątem wycieków pamięci.

6. Wnioski

Zadanie zostało zrealizowane.