Politechnika Śląska w Gliwicach
Wydział Automatyki, Elektroniki i Informatyki

Laboratorium Programowania Komputerów

Sokoban

autor	Paweł Zachara
prowadzący	dr inż. Adam Gudyś
rok akademicki	2018/2019
kierunek	informatyka
rodzaj studiów	SSI
semestr	2
grupa	5
sekcja	1
Ścieżka dostępu	Pawel-Zachara-gr51-repo/Projekt/app/Sokoban
termin oddania sprawozdania	2018-06-20
data oddania sprawozdania	2018-06-18

1 Treść zadania

Napisać grę logiczną typu Sokoban (wymagany graficzny interfejs użytkownika).

2 Analiza, projektowanie

2.1 Algorytmy, struktury danych, ograniczenia specyfikacji

- struktury danych :

Najlepsze rozwiązanie to użycie struktur dynamicznych - list, ponieważ ilość przechowywanych w niej elementów zmienia się w zależności od wczytanego poziomu

- algorytmy :

Najważniejsza w programie to tzw. 'Game Loop' czyli pętla gry która po kolei wykonuje poszczególne operacje : rysowanie obiektów w oknie i sprawdzanie 'zdarzeń' oraz kolizji elementów z otoczeniem.

- biblioteka graficzna :

W tego typu projekcie pisanym w języku C jednym z możliwych rozwiązań jest implementacja biblioteki graficznej takiej jak SDL. Jest ona przyjemna w obsłudze oraz intuicyjna ,szczególnie jeśli chodzi o wczytywanie tekstur w postaci plików z rozszerzeniem .png

3 Specyfikacja zewnętrzna

3.1 Obsługa programu

Gra wywoływana jest z konsoli komendą

```
Sokoban.exe 1
```

Gdzie '1' jest to numer poziomu od którego chcemy zacząć (można oczywiście zacząć od każdego dostępnego poziomu)

Sterowanie.
Sterowanie w programie odbywa się za pomocą 4 klawiszy klawiatury :
-strzałka do góry
-strzałka w dół
-strzałka w lewo
-strzałka w prawo

Dane wejściowe.
Do programu jest wczytywana mapa w postaci pliku tekstowego. W przypadku braku pliku, program nie wczyta poziomu i zakończy działanie.

3.2 Format danych wejściowych

Plik składa się z znaków tekstowych w wymiarach 12 wierszy na 16 kolumn.
Następujące znaki odpowiadają następującym elementowm w programie:
'.' - jest interpretowany jako podłoga po której porusza się gracz
'#' - jest interpretowany jako ściana
'o' - jest interpretowany jako docelowe miejsce na ktorym ma się znaleźć skrzynia
'C' - jest interpretowany jako skrzynia
'P' - jest interpretowany jako pozycja startowa gracza na mapie

Przykładowa zawartość pliku z mapą :

```
1  ################
2   #####..........#
3   #####.......C.##
4   #..o..###......#
5   #..o....#####..#
6   #..o ..........#
7   # ............C.#
8   # .............. #
9   #.....#####...#
10  ######.....C...#
11  #P ............. #
12  ################
```

4 Specyfikacja wewnętrzna

4.1 Zmienne

Struktury danych

```
1  typedef int bool;
2  #define true 1
3  #define false 0
4
5  #define WindowWidth 816
6  #define WindowHeight 612
7
8  #define TexturesWidthAndHeight 51
9
10 #define LogicInnerLoopCount 16
11 #define LogicOuterLoopCount 12
12 #define RenderingLoopCount  192
13
14 typedef struct Player
15 {
16     int x, y, w, h, dx, dy;
17 };
18
19 typedef struct Assets
20 {
21     int x, y, w, h;
22     struct Assets *next;
23
24 };
25
26 typedef struct Crate {
27     int x, y, w, h;
28     bool on_dot;
29     struct Crate *next;
30 };
31
32 enum Textures
33 {
34     TEX_WALL = 0,
35     TEX_FLOOR = 1,
```

```
36      TEX_FLOOR_DOT = 2,
37      TEX_CRATE = 3,
38      TEX_PLAYER_IDLE = 4,
39      TEX_PLAYER_RIGHT = 5,
40      TEX_PLAYER_LEFT = 6,
41      TEX_PLAYER_TOP = 7,
42      TEX_PLAYER_BOT = 8,
43      TEX_GAME_OVER = 9,
44      TEX_CRATE_ONDOT = 10,
45
46      MAP_NONE = -1,
47      MAP_WALL = TEX_WALL,
48      MAP_FLOOR = TEX_FLOOR,
49      MAP_FLOOR_DOT = TEX_FLOOR_DOT,
50      MAP_CRATE = TEX_CRATE,
51
52  };
53
54  typedef struct GameState
55  {
56      struct Assets *Wall, *Floor, *Floor_with_dot;
57      struct Crate *Crate;
58
59      struct Player player;
60
61      int map[12][16];
62
63      SDL_Texture *textures[32];
64      SDL_Renderer *renderer;
65
66      bool move_left, move_right, move_up, move_down, idle
            ;
67
68      int current_level;
69  };
70
71  typedef struct GUI
72  {
73      SDL_Window *window;
74  };
```

Zdefiniowany typ logiczny używany w programie

- **typedef int bool**
  true 1
  false 0

Stałe używane do tworzenia okna : wysokość i szerokość okna.

- WindowWidth
  WindowHeight

Podstawowe wymiary wszystkich tekstur wczytywanych do programu

- TexturesWidthAndHeight

Stałe używane w pętlach do inicjalizacji zmiennych oraz do rysowania mapy w oknie.

- LogicInnerLoopCount
  LogicOuterLoopCount
  RenderingLoopCount

Struktura opisująca Gracza : jego pozycja x i y ,jego przesunięcie podczas ruchu dx i dy oraz jego wymiary h - wysokość i w - szerokość.

- **struct** Player

Struktura w postaci listy jednokierunkowej opisująca wszystkie statyczne elementy na mapie czyli podłogi ,ściany , i miejsca docelowe dla skrzynek.Struktura posiada podstawowe parametry takich elementów czyli pozycja x i y oraz wymiary w i h.

- **struct** Assets

Struktura opisująca skrzynię którą przesuwa gracz, oprócz podstawowych parametrów,posiada zmienną bool on_dot dzięki której sprawdzamy czy skrzynia jest umieszczona na odpowidenim miejscu.

- **struct** Crate

Zdefiniowany typ wyliczeniowy , pomocny przy wczytywaniu i posługiwaniu sie teksturami oraz odwołaniem do elementów na mapie np podczas sprawdzania kolizji.

- **enum** Textures

Główna struktura programu w której zdefiniowane mamy wskaźniki do wszystkich obiektów i elementów gry.

- **struct** GameState

Zmienne używane w funkcji main:

- gameState

obiekt struktury GameState.

- gui

obiekt struktury GUI

- ∗renderer

wskaźnik typu SDL_Renderer używany do renderowania obrazu w oknie.

- **bool** done

zmienna typu **bool** służąca do obsługi Game Loop'a

- **bool** set

zmienna typu **bool** służąca do obsługi pętli programu

- **int** current_level

zmienna typu **int** służąca do przechowywania informacji o obecnym poziomie.

4.2 Funkcje

Funkcje Logiczne :

```
1 void InitializeGameState(struct GameState *game);
```

Funkcja służąca do inicjalizacji wszystkich elementów gry , ich wymiarów , położenia i wszystkich zmiennych logicznych Funkcja bez zwracania wartości.Parametrem jest wskaźnik na główną strukturę programu.

```
1 struct Crate *GetCrateIdx(int x, int y, struct Crate *
    head);
```

Funkcja służąca do wyznaczania miejsca aktualnie sprawdzanej skrzynki. Parametry przesyłane do funkcji to obecne położenie skrzyni oraz głowa listy w której przechowywane są informacje o skrzyniach(wymiary itp.).Funkcja zwraca wskaźnik typu **struct** Crate.

```
1 bool EventsMovementandCollisionDetection(struct
    GameState *game, struct GUI *window, bool *set);
```

Główna funkcja obsługująca wszystkie eventy dla okna oraz klawiszy. W tej funkcji sprawdzana jest także kolizja z otoczeniem czyli kolizja gracz-skrzynka , gracz-ściana, skrzynia-ściana , skrzynia-skrzynia. Funkcja zwraca wartość logiczną od której zależy czy pętla Game Loop zakończy działanie.Parametry przesyłane w funkcji to wskaźnik na główną strukture programu , wskaźnik na strukturę GUI oraz zmienna logiczna *set obsługująca główną pętlę programu.

```
1 bool IsLevelComplete(struct Crate *head);
```

Funkcja sprawdzająca czy poziom został ukończony , czyli czy wszystkie skrzynki zostały ustawione na odpowiednim miejscu. Funkcja zwraca wartość logiczną od której zależy czy program załaduje kolejny poziom.W parametrach przesyłana jest głowa listy w której znajdują się informacje o skrzyniach (położenie, wymiary ,itd.).

```
1 void AddAsset(struct Assets **head, int x, int  y, int
    w, int h);
```

Funkcja dodająca element do listy jednokierunkowej czyli nowy asset (element podłogi , ściany itp.).W parametrach przesyłana jest głowa Listy do której ma zostać dodana nowy element na mapie, jego położenie na mapie, oraz wymiary.

```
1 void AddCrate(struct Crate **head, int x, int  y, int w
    , int h);
```

Funkcja dodająca element do listy jednokierunkowej czyli nową skrzynię.W parametrach przesyłana jest głowa Listy do której ma zostać dodana nowa skrzynia, jej położenie na mapie, oraz wymiary.

```
1 void DestroyAsset(struct Assets **head);
2 void DestroyCrate(struct Crate **cratehead);
```

Obie funkcje służą do dealokacji pamięci czyli usunięcia list w których przechowywane są informacje o elementach na mapie.Obie w parametrach otrzymują głowę listy jednokierunkowej.

Funkcje obsługujące dane wejściowe.

```
1 bool LoadMap(struct GameState *game, char *filename);
```

Funkcja służąca do załadowania mapy z pliku o odpowiednim formacie.W parametrach otrzymuje wskaźnik na główną strukturę programu oraz nazwę pliku.Zwraca wartość logiczną która informuje o tym czy mapa została wczytana czy nie.

```
1 bool LoadNextLevel(struct GameState *game, int
    current_level);
```

Funkcja służąca do załadowania kolejnego poziomu.W parametrach przesyłany jest wskaźnik na główną strukturę programu oraz zmienna przechowująca informację o obecnym poziomie.Funkcja zwraca wartość logiczną która mówi o tym czy jest możliwośc załadowania kolejnego poziomu.

Funkcje służące do obsługi zdarzeń dla GUI.

```
1 void GUIWindowClose(SDL_Window *window, bool *set, bool
    *done);
2 void GUIQuit(bool *set, bool *done);
```

Obie funkcje służa do obsługi zdarzeń dla GUI czyli w tym przypadku zamykanie okna. W parametrach otrzymują wskaźniki na zmienne logiczne obsługujące główne pętle programu , decydujące o zakończenia tych pętli. W pierwszej funkcji w parametrach przesyłany jest też wskaźnik na zmienną window używaną przy tworzeniu okna gry.

Funkcje obsługujące rysowanie mapy oraz innych elementów na ekranie.

```
1 void doRender(struct GameState *game);
```

Funkcja służąca do rysowania elementów mapy na ekranie. W parametrach otrzymuje wskaźnik na główną strukturę programu.

```
1 void DrawGameOver(struct SDL_Renderer *renderer, struct
    SDL_Texture *textures[]);
```

Funkcja służąca do rysowania napisu Game Over po ukończeniu wszystkich poziomów.W parametrach otrzymuje wskaźnik na zmienną renderer używaną przy rysowaniu elementów na mapie oraz wskaźnik na tablicę textures [] przchowującą załadowane do programu tekstury.

```
1 void LoadTextures(struct GameState *game);
```

Funkcja służąca do załadowania tekstur do programu.W parametrach orzymuje wskaźnik na główną strukturę programu

```
1 void DestroyTextures(struct GameState *game);
```

Funkcja służąca do dealokacji pamięci , czyli w tym przypadku zniszczenia tekstur załadowanych do programu.W parametrach orzymuje wskaźnik na główną strukturę programu.

5 Testowanie

Program został przetestowany na różnego rodzaju plikach. Pliki niepoprawne (o niepoprawnym formacie) powodują wyświetlenie błędu w konsoli. Program został przetestowany na 3 różnych mapach o różnym stopniu złożoności.

6 Wnioski

Gra typu Sokoban nie jest trudna do zaimplementowania. Największym wyzwaniem było zaznajomienie się z dokumentacją implementowanej biblioteki graficznej, oraz stworzenie algorytmu kolizji z otoczeniem. Pozatym praca nad projektem przebiegała sprawnie i bez większych problemów.

Literatura

www.cpluplus.com
www.cpp0x.pl