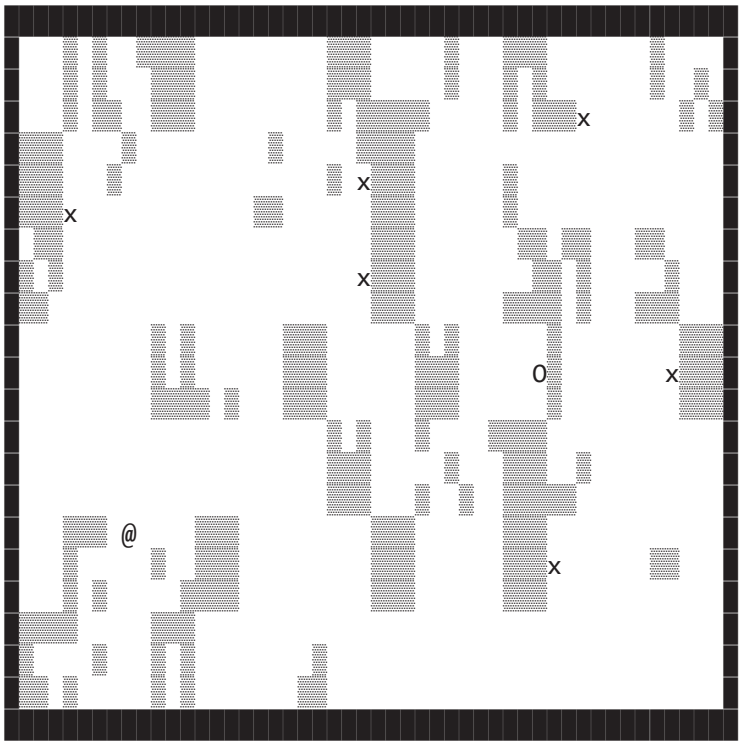
Gra konsolowa „Avoid”

Techniki Programowania - Etap 2



SPIS TREŚCI

OPIS I ZASADA DZIAŁANIA PROGRAMU (GRY)

Informacje ogólne

Program będzie napisany w całości w języku C. Jako output będzie wyświetlał konsolę ze znakami ASCII, a jako input pobierał dane z klawiatury.

Gra, będzie zaprogramowana tak, aby była w stanie wyświetlać animację, dzięki temu stanie się bardziej interaktywna i zręcznościowa.

Zasada działania

Aby wyświetlać animację w czasie trwania gry stworzyłęm własny moduł, który zawiera funkcję wyświetlające tablicę dwuwymiarową oraz funkcję czyszczące ekran konsoli. Przed następnym wyświetleniem i kasowaniem program dokonuje obliczeń logicznych, np. zmiana pozycji gracza, przeciwników itp. Cały proces się powtarza, dopóki gracz nie przegra. Dokładny opis tego modułu znajduję się w dalszej części dokumentu.

Do dynamicznego odczytywania danych wejściowych użyłem funkcji z biblioteki conio.h. Aby sterować swoją postacią gracz używa klawiszy W,A,S,D lub strzałek. po naciśnięciu danego klawisza sterowana przez gracza postać zaczyna przemieszczać się w danym kierunku, aż napotka obiekt (np. ścianę), przez który nie może przejść, wtedy się zatrzymuje. Podczas „lotu" z jednego miejsca do drugiego nie można zmienić już kierunku poruszania.

Poziomy w grze (rozmieszczenie ścian i innych obiektów) generowane są losowo. Gdy gracz przejdzie aktualny poziom wtedy generowany jest następny.

Zasady gry

Gra polega na tym, aby zdobyć jak największy wynik punktów, które zdobywa się przez przejście z miejsca startu do „portalu" końcowego, nie wpadając przy tym na przeciwników. Gracz ma skończoną ilość żyć, które ubywają przy kontakcie z przeciwnikiem. Gdy wszystkie życia znikną wtedy gra się kończy, a gracz przegrywa. Może on wtedy zacząć od nowa z 0 punktami.

Przeciwnicy mogą się poruszać. Poziomy posiadają ściany, które można niszczyć uderzając w nie. Wyjątkiem jest ramka wokół wyświetlanej planszy.

Za każde przejście do portalu końcowego gracz zdobywa punkt. Gra toczy się w nieskończoność, po każdym przejściu poziomu gra generuje losowo następny, aż gracz straci wszystkie życia i przegra, wtedy na końcu wyświetlana jest ilość zdobytych punktów podczas całej gry. Jeśli to któraś gra z rzędu to gra będzie również pokazywała maksymalny zdobyty wynik przez czas trwania programu.

Przykład wyświetlania poziomu

'x' - przeciwnik
'@' - gracz
'O' - portal do następnego poziomu

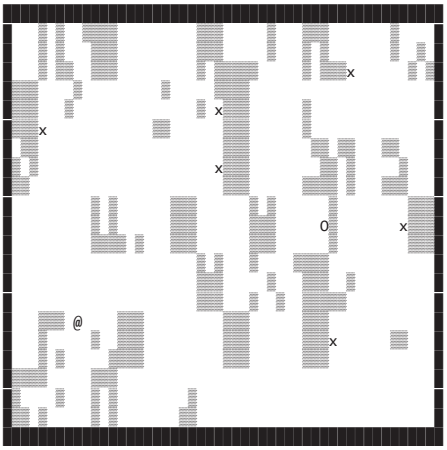
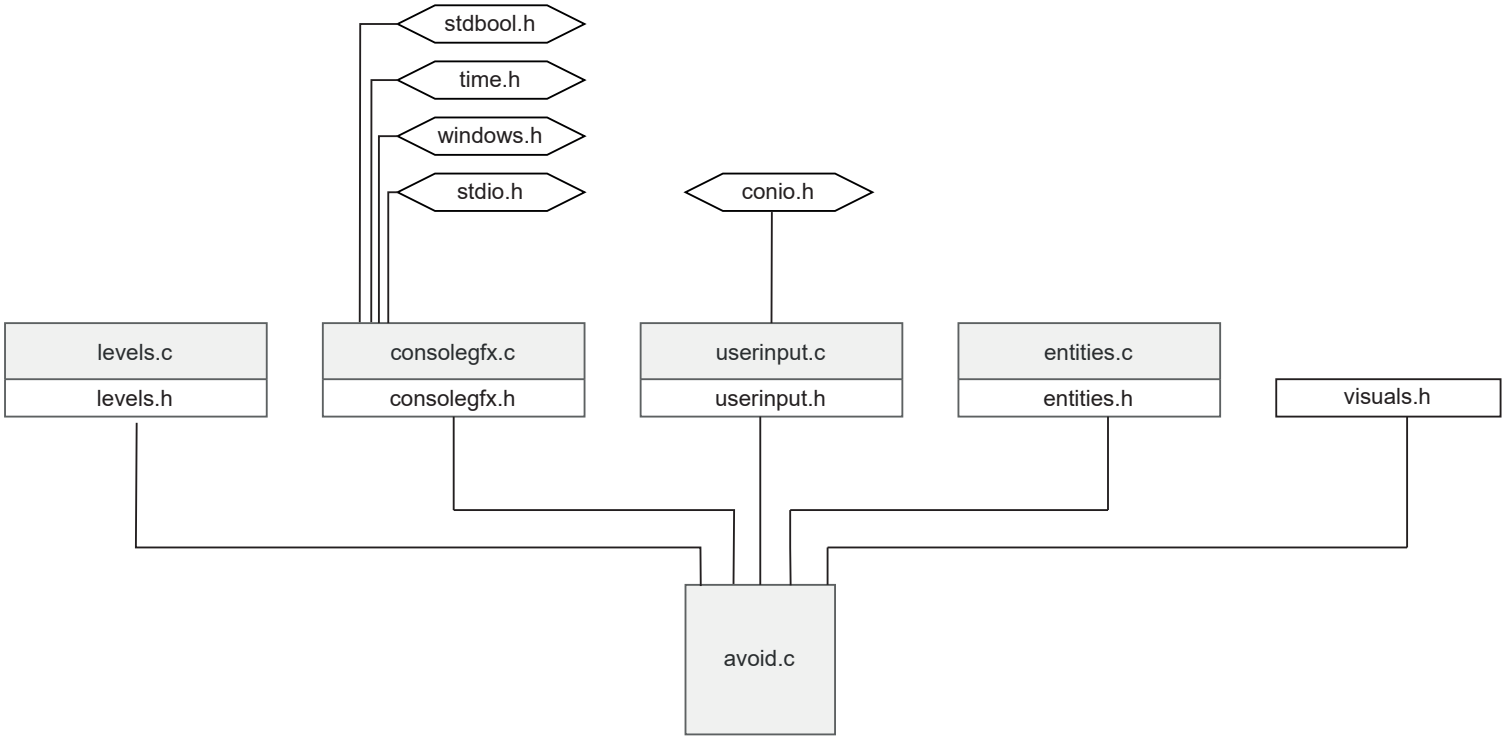


DIAGRAM HIERARCHII MODUŁÓW



OPIS POSZCZEGÓLNYCH MODUŁÓW

avoid.c Głowny moduł. Zawiera funkcję main oraz 2 najważniejsze funkcje dotyczące wyświetlania jak i logikę działania programu. Tutaj będzie kod odpowiedzialny za złożenie wszystkich pozostałych modułów w całość. kontroluje przebieg gry, sprawdza, czy się przegrało, czy zaszły kolizję i kontroluje punkty oraz zleca generowanie następnych poziomów. Używając danych z modułu „userinput" kontroluję poruszanie się gracza.

zbudowany według poniższego schematu:

```
#include „consolegfx.h"

/* deklaracje, zmienne (...) */

int initialize()
{
	//Dzieje się raz na początku
}

void refresh()
{
	//Odświeżanie, aż gracz zdecyduje się zakończyć grę lub przegra
}

int main()
{
	Initialize();

	while (!gameover)
		refresh();

	return 0;
}
```

visuals.h Moduł zawierający „grafikę" do gry. Składa się tylko z instrukcji #define określających jak mają być wyświetlane znaki odpowiadające za dany obiekt. np:

```
#define PLAYER ,@'
#define ENEMY ,x'
```

entities Zawiera struktury „istot" - czyli przeciwników jak i gracza. struktury te zawierają atrybuty takie jak położenie kierunek itp. Moduł zawiera również kod z funkcjami odpowiedzialnymi za sprawdzanie kolizji z otoczeniem, uaktualnia pozycji istot czy schemat poruszania przeciwników.

Przykład ponizej:

```
struct entity{

  point pos;  /* Pozycja  */
  point dir;  /* Kierunek */

  int speed;  /* Predkosc */
  char icon;  /*  Ikona   */

};
```

userinput Sprawdza dane wejściowe z klawiatury. Używając funkcji z biblioteki conio.h dynamicznie przypisuje to co gracz nacisnął do zmiany kierunku poruszania się gracza lub zakończenia gry (Poprzez naciśnięcie x będzie się wychodziło z gry). Jeśli gracz wciśnie inne przyciski, niż te odpowiedzialne za jakieś funkcję, nie będzie to powodowało żadnych działań, ani komunikatów.

consolegfx Drugi najważniejszy moduł. Zawiera funkcję odpowiedzialne za wyświetlanie grafiki oraz animacji. Najważniejszą jego cechą jest to, że jest uniwersalny. Można dzięki niemu wykonać dowolną nieskomplikowaną grę bądź animację, prosty w obsłudze i konstrukcji. Moduł ten jest już praktycznie w całości ukończony.

Większość funkcji tego modułu zostało opisanych poniżej:

```
void ClearConsoleScreen(int CursorPosX, int CursorPosY)
```

Funkcja służąca do czyszczenia ekranu konsoli. W praktyce, dzięki zastosowaniu biblioteki windows.h ustawia ona kursor w lewym górnym rogu. Jest to lepsze rozwiązanie, niż faktyczne czyszczenie konsoli, gdyż redukuje migotanie obrazu.

```
// struktura płótna
struct canvas{

  int sizeX;
  int sizeY;

  char background;
  char outline;

  char fill[WINDOW_WIDTH][WINDOW_HEIGHT];
};
```

```
void FillEmptyCanva(struct canvas *c, char background, char outline)
```

Funkcja służąca do wypełniania płótna tylko znakami tła i obramówką. Tworzenie czegoś w rodzaju pustego obrazu z ramką, na którym można dopiero malować.

```
bool createCanva(struct canvas *c, int size_x, int size_y, char background, char outline)
{
  if (sizeX > WINDOW_WIDTH || sizeY > WINDOW_HEIGHT || c == NULL || sizeX < 0 || sizeY < 0)
    return false;

  c -> sizeX = sizeX;
  c -> sizeY = sizeY;
  c -> background = background;
  c -> outline = outline;

  FillEmptyCanva(c, background, outline);

  return true;
}
```

Funkcja służąca do tworzenia nowego płótna i wypełnienia jego atrybutów podanymi wartościami. Zwraca fałsz, jeśli dla podanych wartości płótna nie da się utworzyć.

```
void printCanva(struct canvas *c)
```

Funkcja służąca do wyświetlanie płótna na ekranie konsoli.

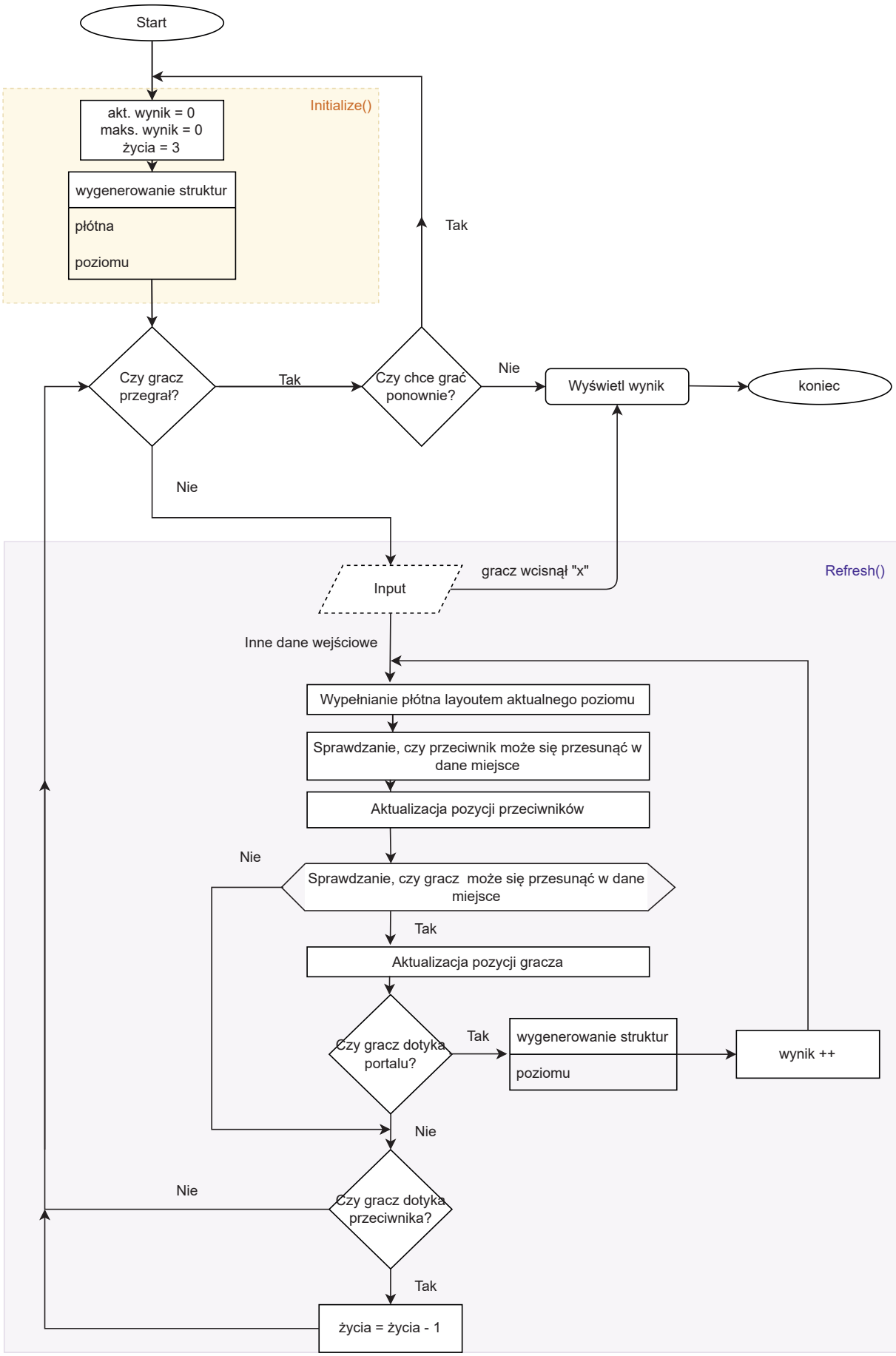
levels

Modół odpowiedzialny za generowanie nowych, losowych poziomów. Zawiera struktury poziomów oraz funkcję z kodem generującym nowy poziom na podstawie mniejszych segmentów - tj. „płytek" o wymiarach 3x3. Składają się tylko ze ścian i tła (po tle gracz może się poruszać). Płytki posiadają poziom „otwartości", czyli tego ile posiadają wyjść. Najbardziej zamknięta płytkia nie ma wyjść, a najbardziej otwarta jest całkowicie pusta.

Przy generowaniu poziomu tworzy on losowo płytki o różnych poziomach otwartości i właśnie z nich składa poziom. Kod jest napisany tak, aby płytki bardziej otwarte pojawiały się częściej, w celu zapobiegnięcia ciasnych poziomów.

W module zawarte są wszystkie dane na temat danego poziomu - miejsce startu, portalu końcowego oraz ilosć, typy i rozmieszczenie przeciwników.

SCHEMAT BLOKOWY UPROSZCZONEGO DZIAŁANIA PROGRAMU



PSEUDOKOD UPROSZCZONEGO DZIAŁANIA PROGRAMU

```
#include <>

int initialize()
{
        gameover = false
        score = 0
        max_score = 0
        lives = 3

        createCanva()
        generatelevel(level)
}

void refresh()
{
        FillEmptyCanva(level)


        for i = 0 to nr_of_enemies
        {
                if (worldCollisions(enemies[i]) == false)
                        moveEntity(entity[i])
        }


        if (worldCollisions(player) == false)
        {
                moveEntity(player)

                if (player.pos = finishportal.pos)
                        generatelevel(level)
        }


        for i = 0 to nr_of_enemies
        {
                if (collides(enemies[i], player) == true)
                        lives--
        }


        if (lives < 0)
                gameover = true

}

int main()
{
        Initialize();

        while (true)
        {
                while (!gameover)
                {
                        refresh()
                        if Input() == ,x'
                                break
                }

                print(score)
                print(do you want to play again? (Y/N))

                if (getchar() == Y)
                {
                        gameover = false
                        score = 0
                        lives = 3
                }
                else
                   break
        }

        return 0
}
```

Ze względu na złożoność programu powyższe schematy przedstawiają jego przybliżone działanie.

Schematy blokowe zostały wykonane na stronie: https://app.diagrams.net/