**Dokumentacja techniczna**

Projekt gra w C - „Shooter”

Autor: Mateusz Szałowski
298976

**Opis projektu**

Projekt Shooter to gra zręcznościowa, jednopoziomowa i jednoosobowa. Kod programu został napisany w języku C (standard c99)
oraz z użyciem darmowej biblioteki graficznej SDL2.0. Do działania programu wymagana jest zainstalowana biblioteka SDL w wersji 2.0 oraz biblioteki SDL_image i SDL_ttf.

**Spis treści**



## Podział na moduły

Kod programu podzielony jest na 6 plików źródłowych oraz 6 plików nagłówkowych w celu lepszej nawigacji podczas jego tworzenia.

## Structures.h

W tym pliku znajdują się globalne struktury gracza, pocisku, wroga, listy pocisków oraz listy wrogów. Są zdefiniowane globalnie, ponieważ wspólnie korzystają z nich funkcje zawarte w plikach: player.c, enemy.c, bullet.c.

Oto struktury:

```
//strukura player
typedef struct {
  float x_pos;
  float y_pos;
  float x_vel;
  float y_vel;
  int center_x;
  int center_y;
  int rotation_angle;
  unsigned int hP;
  int score;
} playerStr;

//struktura enemy
typedef struct
{
  float x_pos;
  float y_pos;
  float x_vel;
  float y_vel;
  float center_x;
  float center_y;
  int rotation_angle;
  unsigned int hP;
} EnemyStr;

//struktura list of enemies
typedef struct ENEMY_LE Enemy_le;
extern struct ENEMY_LE
{
  EnemyStr* this_enemy;
  struct ENEMY_LE* next_enemy;
};
```

```
//struktura bullet
typedef struct {
  float x_pos;
  float y_pos;
  float x_vel;
  float y_vel;
  float target_x;
  float target_y;
} bulletStr;

//struktura list of bullets
typedef struct BULLET_NODE bulletNode;
extern struct BULLET_NODE
{
  bulletStr* this_bullet;
  struct BULLET_NODE* next_bulletNode;
};
```

## SDL_initialization_functions.c

Plik SDL_initialization_functions.c odpowiada za zainicjalizowanie biblioteki SDL, utworzenie wskaźników na odpowiednie struktury potrzebne do renderowania okna. Oprócz tego, znajdują się tam funkcje ładujące wszelkie media do programu tj. obrazki. Zdefiniowana jest wielkość ekranu oraz podlinkowane biblioteki SDL.

## screen_functions.c

W tym pliku znajduje się funkcja boolowska mouseOver(SDL_Rect), która zwraca prawdę jeśli myszka znajduje się w danym obszarze – rectangle.

## main3.c

Ten plik zawiera w sobie najważniejszy core całej gry. Tutaj renderowane są na bieżąco obiekty i świat oraz wywoływane wszystkie funkcje z innych plików. Ponadto plik obsługuje zdarzenia (events) z klawiatury i myszki i decyduje co gra w danym momencie ma zrobić. Tutaj też znajduje się (main loop) główna pętla całej gry, która wykonuje się nieustannie z opóźnieniem 1/60 sekundy.

## player.c

W tym pliku znajdują się wszystkie funkcje dotyczące obiektu gracza.

`createPlayer()` - tworzy strukturę playerStr i alokuje na nią pamięć.
`DestroyPlayer()` - zwalnia pamięć gracza.
`movePlayer()` - aktualizuje pozycję gracza w oparciu o wciśnięte na. klawiaturze przyciski oraz pozycję myszki.
`renderPlayer()` - renderuje texturę gracza na ekranie.
`renderPlayerScore()` - renderuje licznik punktów gracza.

## enemy.c

W tym pliku znajdują się wszystkie funkcje dotyczące obiektu wroga. Większość z nich jest podobna do tych z player.c, ale oprócz tego mamy jeszczę obsługę list jednokierunkowych.

`createEnemy()` - tworzy strukturę EnemyStr* i alokuje pamięć.

`checkCollision(SDL_Rect 1, SDL_Rect 2)` – funkcja boolowska, która sprawdza czy 2 obiekty kolidują ze sobą na mapie.

`createEnemyNode(EnemyStr* new)` – tworzy wierzchołek listy jednokierunkowej przechowującej struktury wroga.

`appendEnemyNode(EnemyStr* new)` – tworzy wierzchołek ze struktury wroga i dodaje go na koniec listy wrogów (zdefiniowanej globalnie).

`updateEnemyPos(EnemyStr* enemy, playerStr* player)` – podobnie do funkcji movePlayer() aktualizuje pozycje konkretnego wroga. Jednakże funkcja ta za ‘target’ obiektu obiera aktualną pozycję gracza.

`updateEnemiesPositions(playerStr* player)` – funkcja aktualizuje pozycje wszystkich wrogów znajdujących się obecnie na liście.

`handleCollisionBetweenEnemies()` – funkcja, która obsługuje kolizje pomiędzy wszystkimi wrogami na liście i nie dopuszcza do pokrywania się ich tekstur na ekranie.

`makeRoute(playerStr* player)` – gdy gracz próbuje przejść przez wrogów funkcja ta „rozpycha” ich na boki tworząc wolne przejście dla gracza.

`removeDeadEnemies()` - ta funkcja usuwa wszystkich zabitych wrogów (tych o zerowym hp) z listy jednokierunkowej. Zwalnia też pamięć potrzebną do ich przechowywania.

`renderAllEnemies()` - funkcja renderująca wszystkich graczy z listy na ekranie gry.

## Bullet.c

W tym pliku znajdują się wszystkie funkcje obsługujące zdarzenia, tworzenie, dodawanie na listę, usuwanie z niej oraz niszczenie pocisków, które mogą zostać wystrzelone przez gracza lub wroga.

`createBulletP(playerStr* player)` – funkcja tworzy pocisk wystrzelony z pozycji gracza i lecący w stronę kursora myszki.

`createBulletE(EnemyStr* enemy, playerStr* player)` – funkcja tworzy pocisk lecący w stronę gracza (w chwili wystrzelenia) oraz pojawiający początkowo w pozycji wroga, który go wystrzelił.

`createBulletNode(bulletStr* bullet)` – funkcja tworzy nowy Node listy jednokierunkowej pocisków.

`appendBulletNode(bulletStr* new_bullet)` – funkcja tworzy nowy Node i dodaje go na koniec listy pocisków. (zdefiniowanej globalnie).

`moveBullet(bulletStr* bullet)` - aktualizacja pozycji danego pocisku.

`moveAllBullets()` – aktualizacja pozycji wszystkich pocisków na liście.

`bool outside_screen(bulletStr* bullet)` – funkcja boolowska, która zwraca true jeśli pocisk znajduje się poza ekranem

`removeBulletNode(bulletStr* bullet_to_delete)` – usuwa konkretny Node z listy pocisków i zwalnia pamięć używaną do jego przechowywania.

`removeCollidingBulletsP(playerStr* player)` – usuwa z listy pocisków pociski, które są w kolizji z graczem na mapie.

`removeCollidingBulletsE(EnemyStr* enemy, playerStr* player)` – usuwa wszystkie pociski kolidujące z wrogiem na mapie oraz dodaje punkty graczowi.

`renderAllBullets()` – renderuje wszystkie pociski znajdujące się na liście na mapie gry.