Projekt: Space Shooter

Przedmiot: Programowanie Obiektowe

Autor: Klaudia Weigel

1. Spis klas

1. Game
2. Ship
3. GameObject
4. Enemy
5. ResourcesManager
6. Laser
7. Screens
8. Menu
9. EndScreen
10. Score
11. Rock
12. Collision

2. Opis programu

Gra Space Shooter napisana w języku C++ przy użyciu biblioteki graficznej SFML. Gracz ma za zadanie zestrzelić obiekty (statek wroga, kamienie). Gra działa dopóki nie stracone zostaną wszystkie życia.
Po skompilowaniu programu otwiera się menu gry, w którym użytkownik ma do wyboru rozpoczęcie gry lub wyjście. Po rozpoczęciu wyświetla ekran z grą. Na dole ekranu jest postać gracza który może poruszać się w prawo lub w lewo przyciskami odpowiedni D i A. Wystrzał laseru - SPACJA. Gracz ma za zadanie zestrzelić wszystkie kamienie (przepuszczenie trzech kamieni skutkuje utratą życia) oraz unikać laserów przeciwnika. Nie zestrzelenie statku wroga nie wpływa na ilość żyć. W miarę upływu czasu wrogie obiekty nabierają większej prędkości co zwiększa poziom trudności.
Po utracie wszystkich żyć wyświetla się ekran z wynikiem i opcją ponownej gry lub wyjścia.

Kompilacja:

- Linux

Instrukcja instalacji SFML jest na stronie https://www.sfml-dev.org/tutorials/2.4/start-linux.php. (Najlepiej poleceniem `sudo apt-get install libsfml-dev`)
Aby uruchomić program na Linuxie:

```
g++ -c -std=c++11 *.cpp
g++ -std=c++11 *.o -o gra -lsfml-graphics -lsfml-window -lsfml-system -lsfml-audio
./gra
```

Jeśli zostania zainstalowana nowsza wersja SFML, to może się wyświetlić warning, który nie ma wpływu na działanie programu.

3. Opis klas

- Game

Klasa Game kontroluje przebieg gry. Przetwarza przyjęte od użytkownika dane i odpowiednio uaktualnia stan gry.
Atrybuty:

- playerShip: Ship

- menu: Menu
- game_time: sf::Clock
- game_clock: sf::Clock
- current_screen: ScreenId
- laser: Laser
- rocks: Rock
- enemy: Enemy
- collisions: Collision
- end_screen: EndScreen
- game_time_index: int
- elapsed: sf::Time
- game_window: sf::RenderWindow

Metody:
+ Game(ResourceManager&) - konstruktor
+ run() - uruchomienie gry
- run_screen(Screens&, ScreedId) - uruchamia odpowiedni ekran (menu lub ekran po przegraniu)
- reset() - resetuje grę po przegranej, w przypadku jeśli chcemy grać ponownie
- initialize() - inicjalizuje nową grę

- GameObject

Klasa po której dziedziczą wszystkie ruchome obiekty gry.
Atrybuty:
velocity
acceleration
Metody:
+ get_velocity()
+ accelerate()
+ set_velocity()
+ set_acceleration()

- Ship

Klasa Ship zawiera informacje dotyczące gracza i wykonuje na tym obiekcie. Dziedziczy po klasie GameObject, zawiera więc dodatkowe atrybuty i metody opisane w klasie GameObject.
Atrybuty:
- player: sf::Sprite
Metody:
+ Ship(ResourceManager&) - konstruktor
+ move(float, float) - przemieszcza gracza o żądaną wartość
+ get_rect() - zwraca prostokąt (sf::FloatRect) wyznczający krawędzie atrybuty player
+ set_position(sf::Vector2f) - ustawia gracza na nowej pozycji
+ get_position() - zwraca pozycję gracza

- Enemy

Klasa wykonująca operacje na statku wroga. Dziedziczy po GameObject.
Atrybuty:
- distance_to_travel: float
- distance_traveled: float

- direction_index: int
- laser_clock: sf::Clock
- enemy: sf::Sprite
- enem_laser: Laser
- movement: std::vector
+ hit: bool

Metody:
+ Enemy(ResourceManager&) - konstruktor
+ draw()
+ set_position(sf::Vector2f)
+ get_rect()
+ draw_enemies(sf::RenderWindow, sf::Time)
+ create_eniemies()
+ shoot_laser(sf::RenderWindow, sf::Time)
+ reset()

- Rock

Zawiera informacje dotyczące kamieni i odpowiada za wykonywane na nich operacje. Dziedziczy po GameObject.
Atrybuty:
- rocks: std::vector
- rocks_index: int
- medium_rock: sf::Sprite
- big_rock: sf::Sprite
- previous_distance: float

Metody:
+ Rock(ResourceManager&)
+ create_rocks();
+ draw_rocks(sf::RenderWindow, sf::Time)
+ reset()

- Laser

Zawiera informacje dotyczące lasera i odpowiada za operacje na nim wykonywane. Dziedziczy po GameObject
Atrybuty:
- laser: sf::Sprite
- laser_sound: sf::Sound

Metody:
+ Laser(ResourceManager&)
+ move(sf::Vector2f)
+ get_rect()
+ draw(sf::RenderWindow)
+ set_position(sf::Vector2f)
+ get_position()
+ play()

- Collision

Zawiera metody wykrywające kolizje między obiektami. W razie kolizji podejmuje odpowiednie działania.
Atrybuty:

- collision_sound:
Metody:
+ player_hit(Ship&, Enemy&) - wykrywa kolizję pomiędzy wrogim statkiem lub laserem wroga
+ enemy_hit(Laser&, Enemy&) - wykrywa kolizję pomiędzy strzałem gracza i wrogiem
+ rock_hit(Rock&, Laser&) - wykrywa kolizję pomiędzy strzałem gracza i kamieniem
+ player_hit_rock(Rock&, Ship&) - sprawdza czy gracz został trafiony przez kamień

- Screens

Klasa opakowująca klasy Menu i EndScreen.
Metody (wszytkie metody abstrakcyjne):
+ move_up() - przesunięcie kursora w górę
+ move_down() - przesunięcie kursora w dół
+ quit() - zwraca prawdę jeśli użytkownik chce zakończyć grę
+ draw(sf::RenderWindow&) - rysuje zawartość ekranu
Atrybuty:
width: int - szerokość głownego okna gry
height: int - odpowiednio wysokość

- Menu

Implementuje metody opisane w klasie Screens. Odpowiada za menu które wyświetla się po uruchomieniu gry. Użytkownik ma do wyboru rozpoczęnie gry, bądź wyjście.
Metody(poza dziedziczonymi):
+ Menu(float, float, ResourceManager&) - konstruktor
Atrybuty:
- current_option:int
- background: sf::Sprite
- play_text: sf::Text
- exit_text: sf::Text

- EndScreen

Wyświetlany kiedy gra zakończy się. Pokazuje wynik gracza i oferuje opcję ponownej gry lub wyjścia. Implementuje metody klasy Screens.
Metody:
+ EndScreen(float, float, ResourceManager&) - konstruktor
Atrybuty:
- current:int
- background: sf::Sprite
- play_again: sf::Text
- exit: sf::Text
- score: sf::Text

- Score

Zarząda wynikiem gracza.
Atrybuty:
- lives_vec: std::vector
- score: float
- lives: int
- live: sf::Sprite

- score_text: sf::Text
Metody:
+ Score(int, ResourceManager&)
+ update_lives(int)
+ get_score()
+ draw(sf::RenderWindow&)
+ reset()

- ResourceManager

Klasa jest pojemnikiem na używane w programie zasoby. Wszystkie ładowane są na początku działania programu. Dzięki zastosowaniu tej klasy są ładowane tylko raz.
Atrybuty:
- textures: std::map
- audio: std::map
- fonts: std::map
Metody:
- void load_from_file_texture(TextureId, std::string)
- void load_from_file_audio(AudioId, std::string)
- void load_from_file_font(FontId, std::string)
+ ResourceManager() - konstruktor
+ load_resources()
+ get_texture(TextureId)
+ get_audio(AudioId)
+ get_font(FontId id)

4. Zastosowane wzorce projektowe

- Pyłek (ang. *flyweight*) - wszysktie zasoby ładowane są tylko raz w klasie ResourceManager a potem wykożystywane przy tworzeniu większych obiektów. Dzięki takiemu zastosowaniu znacząco skraca się czas działania programu, zwłaszcza jeśli wiele obiektów wykorzystuje te same zasoby.
- Metoda wytwórcza (ang. *factory method*) - zwłasza w instancji klasy Screens, która jest w większej części wirtualna i zawiera tylko niezbędne metody potrzebne do obsługi ekranu. W klasach, które
- Łańcuch odpowiedzialności (ang. *chain of responsibility*) - funkcja reset(), zawarta w klasie Game delegująca zadanie zresotowania do kolejnych klas.