PSIO projekt

Michał Skoczylas, Jan Rębacz

Maj 2024

1 Opis i prezentacja aplikacji

1.1 Pomysł na rozgrywkę

Będzie to implementacja gry przeglądarkowej Ogień i Woda w SFMLu. Elementem odróżniającym od zwykłych platformerów ma być to że w grze będzie brało udział dwóch graczy na raz. Przechodząc poziomy będą odblokowywali kolejny (ok. 5 łącznie) na każdej z plansz będzie nowa mechanika typu: śliska podłoga, przyciski na których trzeba stanąć żeby drugi gracz coś mógł zrobić itd.
Gracze będą zbierali punkty przyznawane w zależności od czasu ukończenia poziomu i zebranych bonusów(gwiazdek lub czegoś podobnego).
Sterowanie przy pomocy strzałek i WSAD-u . Poziom trudności będzie się skalował wraz z kolejnymi etapami i nowymi mechanikami wymienionymi wcześniej.
Wybór poziomów będzie z dedykowanej aplikacji GUI napisanej w QT.

2 Makieta intefejsu

2.1 Makieta aplikacji:

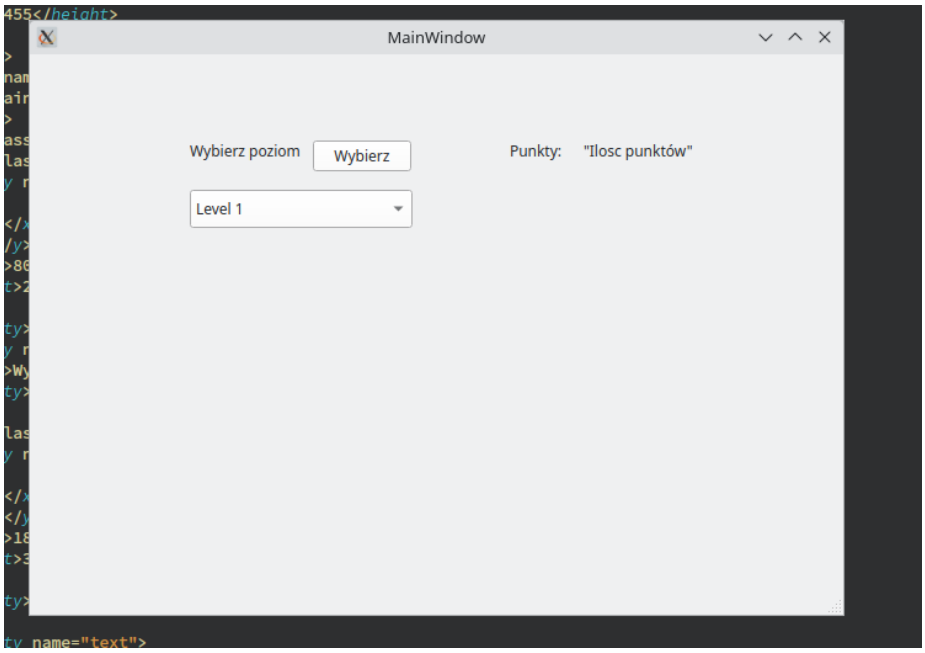

Figure 1: Szkic aplikacji w QT

3 Wstępny diagram klas

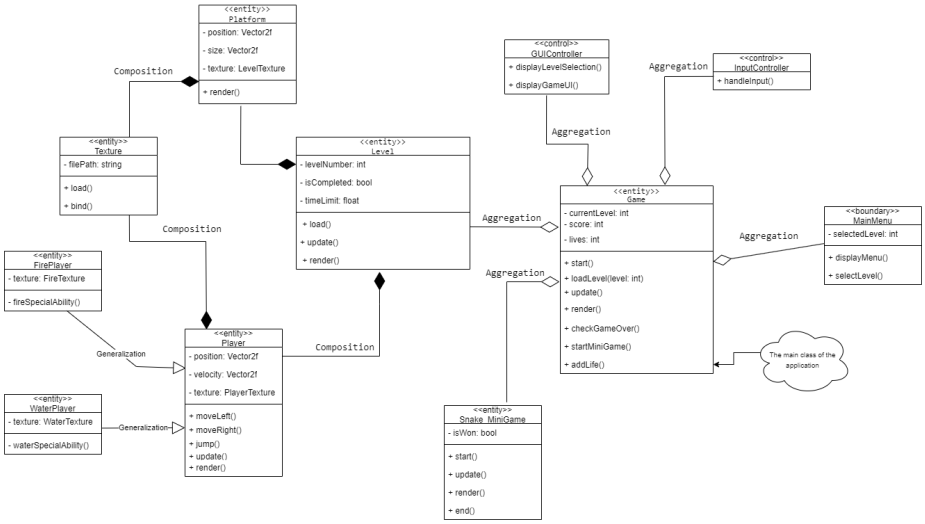

Figure 2: Szkic diagramu klas

3.1 Opis diagramu:

- **Game**: Klasa główna, która zarządza całą grą, obsługującą ładowanie poziomów, aktualizację stanu gry i renderowanie grafiki, dodatkowo sprawdza, czy gra się zakończyła, startMiniGame() uruchamia minigrę, a addLife() dodaje życie po wygraniu minigry.
- **Level**: Reprezentuje pojedynczy poziom gry, zawierając informacje o numerze poziomu, czy został ukończony, oraz limit czasu.
- **MainMenu**: Klasa odpowiedzialna za wyświetlanie i obsługę głównego menu gry, w tym wybór poziomów.
- **InputController**: Klasa odpowiedzialna za obsługę wejścia użytkownika (strzałki i WSAD).
- **Player**: Klasa reprezentująca gracza, zawierająca pozycję, prędkość oraz metody poruszania się i renderowania.
  - **PlayerFire**: Klasa dziedzicząca po klasie Player, dodająca unikalne umiejętności dla postaci Ognia.
  - **PlayerWater**: Klasa dziedzicząca po klasie Player, dodająca unikalne umiejętności dla postaci Wody.
- **Platform**: Klasa reprezentująca platformy w grze, które mogą mieć różne mechaniki, np. śliską podłogę.

- **GUIController**: Klasa zarządzająca interfejsem użytkownika, wyświetlająca wybór poziomów oraz UI gry.
- **Texture**: Nowa klasa reprezentująca tekstury używane w grze. Zawiera atrybut filePath przechowujący ścieżkę do pliku tekstury oraz metody load() i bind() do ładowania i wiązania tekstury.
- **MiniGame**: Nowa klasa reprezentująca minigrę. Zawiera atrybut isWon (czy minigra została wygrana) oraz metody start(), update(), render(), i end().

3.2 Relacje między klasami

- **Game** → **Level**: Strzałka agregacji, ponieważ klasa Game agreguje obiekty klasy Level.
- **Game** → **MainMenu**: Strzałka agregacji, ponieważ klasa Game może mieć referencję do obiektu MainMenu.
- **Game** → **InputController**: Strzałka agregacji, ponieważ Game może agregować InputController do obsługi wejścia użytkownika.
- **Game** → **GUIController**: Strzałka agregacji, ponieważ GUIController jest odpowiedzialny za wyświetlanie interfejsu użytkownika w grze.
- **Level** → **Player**: Strzałka kompozycji, ponieważ poziom zawiera obiekty graczy.
- **Level** → **Platform**: Strzałka kompozycji, ponieważ poziom zawiera obiekty platform.
- **Player** → **PlayerFire**: Strzałka generalizacji, ponieważ PlayerFire dziedziczy po klasie Player.
- **Player** → **PlayerWater**: Strzałka generalizacji, ponieważ PlayerWater dziedziczy po klasie Player.
- **Player** → **Texture**: Kompozycja, ponieważ każdy obiekt gracza ma teksturę.
- **Platform** → **Texture**: Kompozycja, ponieważ każda platforma ma teksturę.
- **Game** → **MiniGame**: Agregacja, ponieważ instancja Game zarządza obiektem MiniGame w przypadku utraty życia.