Politechnika Śląska w Gliwicach
Wydział Automatyki, Elektroniki i Informatyki

Programowanie Komputerów 4

Strategia Turowa

autor	Adam Hudziak
prowadzący	dr hab. Roman Starosolski
rok akademicki	2019/2020
kierunek	informatyka
rodzaj studiów	SSI
semestr	4
grupa	2
sekcja	1
data oddania sprawozdania	2019-07-02

1 Analiza

Gra jest strategia turową. Rozgrywka dzieli się na 2 płaszczyzny: chodzenie po mapie oraz walka. Po mapie przemieszczamy się bohaterem, który ma w sobie jednostki. Bohater może pokonać w ciągu tury określoną liczbę ruchu. Każde pole ma określony koszt ruchu. Gdy spotkamy wrogiego bohatera możemy go zaatakować. Wtedy gra wchodzi w tryb walki.
Biblioteki zewnętrzne: SFML
IDE: JetBrains Clion
Kompilator:gcc 8.3
System operacyjny: Debian

2 Specyfikacja zewnętrzna

Grę obsługujemy za pomocą myszki. Mapę przewijamy przytrzymać PPM na oknie mapy i ruchamy myszka. Bohaterem oraz jednostkami poruszamy klikając na niego potem na sąsiadujące pole.

3 Specyfikacja zewnętrzna

Program jest uruchamiany z linii poleceń. Należy przekazać do programu nazwy plików: Pliku układu, pliku stanów wejściowych oraz pliku wyjściowego zgodnie z przykładem:

```
TUC plik_ukladku pilk_stanów_wejściowych pilk_wyjsciowy
```

Pliki są plikami tekstowymi, ale mogą mieć dowolne rozszerzenie (lub go nie mieć.) Przełączniki mogą być podane w dowolnej kolejności. Uruchomienie programu bez żadnego parametru:

```
TUC
```

powoduje wyświetlenie krótkiej pomocy.

Podanie nieprawidłowej nazwy pliku powoduje wyświetlenie odpowiedniego komunikatu:

```
Błedny plik
```

Plik wejściowy przedstawiający układ na następujący format:

```
węzeł_wejściowy1 węzeł_wejściowy2 wezeł_wyjściowy
```

np.

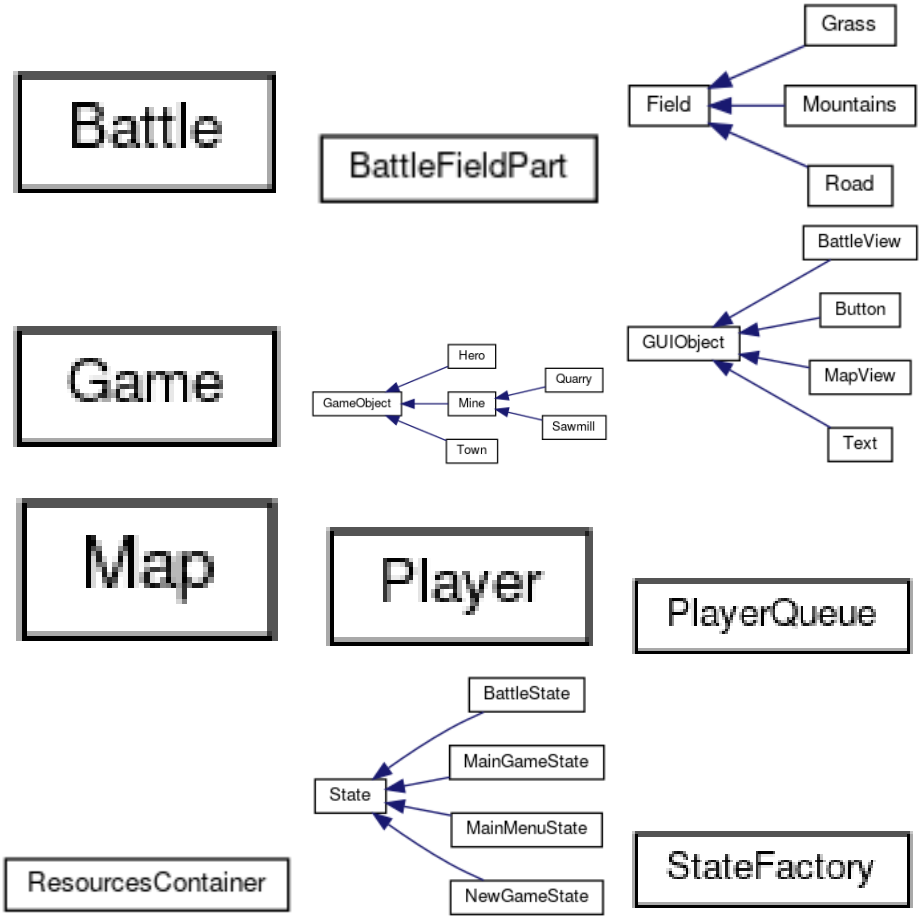
```
IN: 1 6
OUT: 3
NAND 1 6 5
NAND 1 5 2
NAND 5 6 4
NAND 2 4 3
```

Przykładowy plik stanów:

```
1:0 6:0
1:0 6:1
1:1 6:0
1:1 6:1
```

4 Specyfikacja wewnętrzna

4.1 Hierarchia klas



Rysunek 1: Diagram Klas

4.2 Ogólna struktura programu

4.2.1 Klasa Game **Game**

Głowna klasa gry.

4.2.2 Klasa Map **LogicGate**

Reprezentuje mapę i logikę z nią związaną

```
1 void addHero (const std::shared_ptr< Hero > &obj)
```

Dodaje bohatera do mapy

```
1 void generatorTileMap ()
```

Generuje tilemap, który pozwana rysować tło mapy.

```
1 const TileMap & getTileMap () const
```

```
1 const std::vector< std::shared_ptr< GameObject > > &
    getBuildings () const
```

```
1 const std::unique_ptr< Battle > & getCurrentBattle ()
    const
```

```
1 void select (unsigned int x, unsigned int y)
```

Zaznacza pole o współrzędnych x, y.

```
1 const std::vector< std::shared_ptr< GameObject > > &
    getDynamicObjects () const
```

```
1 Player * currentPlayer () const
```

```
1 void moveHero ()
```

Przemieszcza o ile to możliwe bohatera na pole kursora.

```
1 void newTurn ()
```

```
1 void removeHero (Hero *hero)
```

4.3 Klasa Battle

Obiekt reprezentuje bitwę w grze.

```
1 void addAttackerUnit (const std::shared_ptr< Unit > &
    unit)
```

Dodaje jednostki atakujacego do bitwy.

```
1 void addAttackedUnit (const std::shared_ptr< Unit > &
    unit)
```

Dodaje jednostki atakowanego do bitwy.

```
1 void select (unsigned int x, unsigned int y)
```

```
1 const sf::Vector2u & getSelection () const
```

```
1 void changePlayer ()
```

```
1 void moveUnit ()
```

4.4 Klasa Hero

```
1 Player * getOwner () const
```

```
1 bool move (Field &field)
```

```
1 void resetMove ()
```

```
1 bool addUnit (std::shared_ptr< Unit > unit)
```

```
1 const std::vector< std::shared_ptr< Unit > > &
    getUnits () const
```

4.5 Klasa Unit

Jest klasą bazową dla wszystkich jednostek w grze.

```
const sf::Sprite &    getSprite () const
```

```
void   setSprite (const sf::Sprite &sprite)
```

```
void   setPoz (unsigned _x, unsigned _y)
```

```
void   setX (unsigned int x)
```

```
void   setY (unsigned int y)
```

```
bool   move ()
```

```
void   attack (Unit &target)
```

Atakuje inna jednostkę i odejmuje je punkty życia.

```
bool   isDead ()
```

Zwraca czy jednostka ma 0¿= HP.

```
Player *     getOwner () const
```

```
unsigned int    getCurrentMove () const
```

Zwraca ilość pozostały punktów ruchu

4.6 Klasa Field

Reprezentuje podstawowe pole gry(klasa bazowa). Od klasy dziedzicznej zależy koszt tego pola(w punktach ruchu).

```
double    getCost ()
```

```
const sf::Vector2i &     getOffset () const
```

```
virtual void    resetCost ()=0
```

```
void   addBuilding (const std::shared_ptr< GameObject >
   &ptr)
```

```
std::optional< std::shared_ptr< GameObject > >
   getVisitedObject ()
```

4.7 Klasa Player

```
1 void addTown (std::shared_ptr< Town > ptr)
```

```
1 void addHero (std::shared_ptr< Hero > ptr)
```

```
1 void addMine (std::shared_ptr< Mine > ptr)
```

```
1 const std::string & getName () const
```

```
1 void newTurn ()
```

4.8 Klasa StateMachine

Służy do zarzadzania stanami gry takimi jak Menu, mapa, walka itp.

```
1 void render ()
```

```
1 void input ()
```

```
1 void pullState ()
```

Przywraca poprzedni stan gry.

```
1 void pushState (const std::shared_ptr< State > &state)
```

Zmienia stan i daje go na stos.

4.9 Klasa State

Reprezentuje stan.

```
1 const std::shared_ptr< State > & getPrevState ()
    const
```

```
1 void setPrevState (const std::shared_ptr< State > &
    prevState)
```

```
1 virtual void render ()
```

```
1 virtual void input ()
```

4.10 Klasa GUIObject

Klasa bazowa dla wszytstkich elementów interfejsu.

```
virtual void draw ()
```

```
virtual bool isMouseHover ()
```

Zwraca czy myszka najechała na obiekt.

```
std::function<void(StateMachine &)> onClick
```

Callback do kliknięcia LPM

4.11 Button

Reprezentuje przycisk.

4.12 Klasa MapView

Służy do wyświetlania oraz kontrolowania mapy.

5 Testowanie

Program został sprawdzony pod kątem wycieków pamięci programem Valgrind.