Politechnika Wrocławska
Wydział Elektroniki

PROGRAMOWANIE OBIEKTOWE - DOKUMENTACJA

Bartosz Terka
nr albumu 253253

1 Użytkowanie programu

1.1 Instalacja

Aby uruchomić aplikację należy uruchomić plik gra.exe . Dla wygody korzystania z aplikacji można utworzyć skrót do aplikacji (klikając na aplikację prawym przyciskiem myszy - utwórz skrót)

1.2 Rozgrywka

1. Gra typu Tower defense polega na obronie zamku przed przeciwnikami, strzelając do nich oraz podkładając bomby. Jeśli gracz lub wieża straci życie(będzie ono równe 0) to rozgrywka zostaje zakończona, a wynikiem gracza jest wynik, który w momencie zakończenia gry był wyświetlani w górnym lewym rogu ekranu. Jest to gra typu High Score – należy zdobyć jak najwyższy wynik.

Po włączeniu aplikacji rozgrywka rozpoczyna się natychmiastowo. W lewym górnym rogu wyświetlane są informacje na temat aktualnej rozgrywki:

Score – informuje o dotychczas uzyskanym wyniku
Zdrowie – informuje o punktach zdrowia gracza (maksymalnie 100, minimalnie 0)
Kondycja zamku – informuje o punktach zdrowia zamku (maksymalnie 100, minimalnie 0)
Ilość bomb – informuje o ilości bomb, jakie użytkownik może podłożyć
Czas – czas aktualnej rozgrywki (w sekundach)

W prawym górnym rogu wyświetlany jest aktualny rekord – najwyższy, zdobyty dotychczas wynik.

Gracz porusza się po planszy. Jego zadaniem jest ochrona zamku – poprzez wystrzelane pociski oraz podłożone bomby.

Sterowanie:
Klawisze strzałek – poruszanie góra/dół/prawo/lewo
Klawisz spacji – wystrzelenie pocisku
Klawisz enter – podłożenie bomby w miejscu, w którym aktualnie znajduje się gracz.

2 Dokumentacja Techniczna

Środowisko: QT Creator 5.12.2 minGW 32-bit
Kompilator: MinGW 7.3.0 32-bit for C++
Debugger: GDB Engine using QT 5.12.2 MinGW 32-bit

1.Użyte biblioteki
-QGraphicsTextItem
-QObject
-QGraphicsView
-QWidget
-QGraphicsScene
-QGraphicsPixmapItem
-QTimer
-QFont
-QApplication
-QDir
-stdlib.h
-QKeyEvent
-QList
-QDebug

2.Spis klas

Czas
Gra
Gracz-zdrowie
Nowa-zapora
Nowe-zdrowie
Gracz
Pocisk
Przeszkoda
Tower
Tower-zdrowie
Wynik
Zapora
Zapora-ilosc

3.Działanie programu

W pliku źródłowym main.cpp tworzony jest obiekt klasy gra, w którym odbywa się całe działanie programu.

```
Gra *gra;
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    gra=new Gra();
    return a.exec();
}
```

Rysunek 1: Wnętrzne main.cpp

4.Gra

Klasa Gra dziedziczy z klasy QGraphicsView. Dzięki temu gra tworzy widok, w którym umieszczana jest scena. Na scene dodawane są wszystkie elementy widoczne na ekranie. Okno ekranu gry ma wymiary 800x600 pix.

5.Gracz oraz przeciwnicy

Klasa Gracz dziedziczy z klas QObject(umożliwia użycie timera) oraz QGraphicsPixmapItem. Gracz co określony odcinek czasu tworzy przeciwników po przeciwnej stronie mapy (na losowej szerokości, w coraz krótszych odcinkach czasu).
Zasada dziania przemieszczania się gracza, przeciwników oraz pocisków:
Poruszanie graczem po mapie bazuje na funkcji keyPressEvent dziedziczonej z klasy QGraphicsView. Jeśli zostanie wykryty któryś z klawiszów strzałek, gracz zmienia swoją pozycję na mapie – według współrzędnych. Ponadto każdorazowo sprawdzane jest, czy gracz nie jest na krańcu mapy – tak, aby jej nie opuścił.

Funckja keyPressEvent wykrywa jest określona również dla przycisków: - spacja – wówczas gracz tworzy pocisk, który przy pomocy timera cyklicznie porusza się do przodu
- enter – wówczas w miejscu, w którym znajduje się gracz (według współrzędnych) stawiana jest bomba.

Przeciwnicy poruszają się na takiej zasadzie jak pociski
Zarządzanie pamięcią: -każdorazowo po poruszeniu się pocisku lub przeciwnika sprawdzane jest, czy nie opuścił on mapy, jeśli tak obiekt jest usuwany – funkcja delete.

6.Wykrywanie kolizji

Do przeprowadzenia rozgrywki koniecznie jest wykrywanie kolizji obiektów różnych klas:
- Gracz – Przeszkoda
- Gracz – Nowe-zycie
- Gracz – Nowa-zapora
- Przeszkoda – Pocisk

- Przeszkoda – Tower.

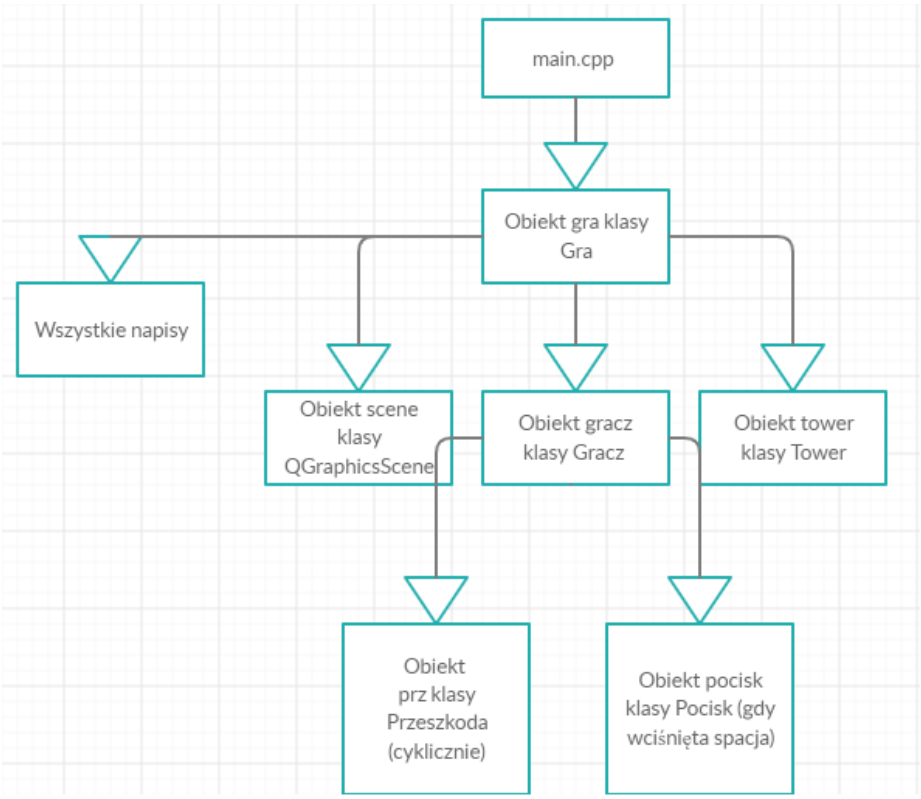
W tym celu wykorzystana jest biblioteka QList. Co każdy ruch danego obiektu sprawdzane jest, czy nie ma on kolizji z jednym z obiektów klas, z którym konieczna jest obsługa kolizji.

```
   QList<QGraphicsItem *> colliding_itemsX = collidingItems();
  for (int i = 0, n = colliding_itemsX.size(); i < n; ++i){
 if (typeid(*(colliding_itemsX[0])) == typeid(Nowe_zdrowie)){
          gra->gracz_zdrowie->dodaj();
          scene()->removeItem(colliding_itemsX[0]);
          delete colliding_items[0];
   }
 }
```

Rysunek 2: Wykrywanie kolizji

Wyżej pokazane jest wykrywanie kolizji gracza z nowym zdrowiem – obiektem, który zwiększa zdrowie gracza, o ile jest ono mniejsze od 100. Sprawdzane są wszystkie obiekty klasy, które chcemy sprawdzić. W tym przypadku, gdy następuje kolizja, to wywołana jest funkcja zwiększająca zdrowie gracza, a obiekt klasy nowe-zdrowie jest usuwany.

7.Struktura tworzenia obiektów



Rysunek 3: Tworzenie obiektów

8.Wykrywanie końca gry

Za każdym razem, gdy gracz lub zamek tracą zdrowie, sprawdzane jest, czy zdrowie gracza lub wieży nie jest równie lub mniejsze od 0. Jeśli tak jest, dana składowa obiektu gra sprawdz-koniec ustawiana jest na 1. Gracz oraz wieża w konstruktorach ustawiane mają timery, które co 100 ms sprawdzają czy zmienna obiektu gra - sprawdz-koniec jest ustawiona na 1. Jeśli tak, funkcja ta wywołuje funkcję Gra::sprawdz(). Gra sprawdza, czy zmienna sprawdz-koniec faktycznie równa jest 1. Jeśli tak, następuje koniec gry, wszystkie obiekty przestają się poruszać, a na ekranie wyświetlana jest informacja o końcu gry, zdobyty wynik, czas gry oraz sprawdzane jest, czy jest to nowy najwyższy wynik – jeśli tak, taka informacja również wyświetlana jest na ekranie.

9.Obsługa pliku

Przy pomocy biblioteki fstream gra, gdy się tworzy ładuje z pliku aktualny rekord i wyświetla go na ekranie. Jeśli po zakończonej rozgrywce ustanowiony jest nowy rekord, obiekt gra nadpisuje zawartość pliku z rekordem tak, że w pliku umieszczony jest jedynie nowy rekord.

10.Obsługa grafiki

Grafiki do gry znajdują się w folderze img. Do programu trafiają za pomocą pliku zasobowego zasb.qrc.