Projekt - gra Tower defense
Wstęp do Programowania w Języku C

Michał Kępa

7 lutego 2019

1 Opis rozgrywki

Gra polega na obronie przed kolejnymi falami wrogów. Użytkownik może, jeśli ma wystarczająco monet, budować nowe wieżyczki oraz ulepszać stare (każdą z wieżyczek można ulepszyć maksymalnie 3 razy). Istnieje także możliwość sprzedaży wcześniej postawionych budynków za połowę kosztów ich wybudowania. Aby ułatwić rozgrywkę, gracz ma dostęp do 2 prędkości gry i pauzy, które pomagają lepiej rozplanować stategię. Każdy statek, który przedostanie się na koniec ścieżki obniża poziom wytrzymałości o jeden punkt, a gdy spadnie on do zera, to gracz przegrywa. Wygrana następuje, jeśli po przejściu wszystkich fal pozostanie co najmniej jeden punkt wytrzymałości.

2 Opis programu

2.1 Menu główne

Po włączeniu gry użytkownik zobaczy menu główne, gdzie przy pomocy myszki może wybrać poziom, w który chce zagrać albo wyjść z aplikacji.

2.2 Okno gry

Po kliknięciu w jeden z przycisków wyboru poziomu rozpocznie się rozgrywka. Główną część ekranu zajmuje plansza do gry, zaś po prawej stronie użytkownik ma dostęp do opcji i informacji dotyczących rozgrywki:

- powrót do Menu
- wybór szybkości rozgrywki
- aktualna ilość punktów wytrzymałości i monet
- pole opisu wybranej jednostki
- wybór akcji:

- budowa jednej z 3 rodzajów wieżyczek
- sprzedaż wieżyczki
- ulepszenie wieżyczki
- informacja o jednostce

Użytkownik wybiera opcję poprzez kliknięcie w nią lewym przyciskiem myszy. Następnie, po kliknięciu na planszy zostanie wykonana odpowiednia akcja. Przy wybraniu akcji informującej i wybraniu jednostki wroga, użytkownik dowie się ile puntków wytrzymałości ma aktulanie dany statek i jaką osiąga predkość. Przy wyborze wieżyczki uzyska on informacje o wartości obrażeń wieży i jej szybkostrzelność.

3 Kompilacja i uruchamianie

Do kompilacji wymagana jest biblioteki CSFML, którą można zainstalować, poprzez wpisanie w terminalu: `sudo apt-get install libcsfml-dev libcsfml-doc`. Aby skompilować program wystarczy napsiać: `make game`, a żeby uruchomić grę: `./game`. Poleceniem `make clean` usuniemy plik wykonywalny.

4 Implementacja

Funkcje pomocnicze, które są używane w tylko jednym pliku, są zadeklarowane jako static. Jeżeli funkcja działa na jakiejś strukturze, to jej nazwa składa się z: nazwyStruktry_nazwyFunkcji, np Engine_update to funkcja, która jest odpowiedzialna za aktualizowanie silnika gry. Wszystkie nazwy, które zaczynają się od sf pochodzą z biblioteki CSFML.

4.1 Consts

Plik zawiera stałe używane w programie.

4.2 EnemyManager

Struktura EnemyManager jest zaimplementowana jako lista dwukierunkowa, która przechowuje elementy typu EnemyNode. Każdy element przechowuje informacje o przeciwnikach oraz wskaźniki na kolejny i poprzedni element listy. Funkcje w tym pliku służą do obsługi listy.

4.3 Engine

Ta struktura przechowuje wskaźniki na wszystkie klasy mające w nazwie Manager, dynamicznie alokowaną tablicę, w której zapisane są informacje dotyczące aktualnej sceny, ścieżki do plików, z których należy wczytać poziom oraz informacje o kliknięciach przycisków myszki.

4.4 Game

W tym pliku są 2 główne funkcje:

- Game_update która odpowiada, poprzez uruchamianie innych funkcji, za poruszanie się jednostek, wykonywanie ataków, obsługę akcji i przycisków
- Game_init w której tworzy się scena z grą i wszystkie przyciski, wczytuje mapę oraz opis kolejnych fal wroga

4.5 Main

Plik w którym znajduje się funkcja main, a w niej główna pętla gry.

4.6 MainMenu

Podobnie jak w module GAME, tu też są 2 funkcje MainMenu_update i MainMenu_init, które obsługują i tworzą główne menu gry.

4.7 MapManager

W tej strukturze przechowywane są wszystkie informacje dotyczące mapy. Funkcje z tego pliku odpowiadają za obsługę mapy: wczytanie z pliku, utworzenie ścieżki i zwracanie informacji dotyczących pól planszy. Zaimplementowane są tutaj funkcje pomagające przy zamianie koordynatów.

4.8 ProjectileManager

Struktura podobna do ENEMYMANAGER, służy do tworzenia i obsługi pocisków.

4.9 SpriteManager

Struktura podobna do ENEMYMANAGER, służy do tworzenia, przechowywania i rysowania sprite'ów. Zaimplementowana jest tutaj pomocnicza funkcja centrująca środek sprite'a.

4.10 TextManager

Odpowiednik struktury SPRITEMANAGER, ale działa na sfText, a nie sfSprite.

4.11 TextureManager

Struktura, która ma w sobie tablicę, do której wczytywane są kolejne tekstury. Każda tekstura ma swoją nazwę zapisaną w typie wyliczeniowym TextureNames.

4.12 TurretManager

Odpowiednik struktury EnemyManager, ale zajmuje się wszystkim związanym z wieżami. Dla ułatwienia zaimplementowane są tutaj 2 funkcje, które ułatwiają zakup i ulepszanie wież.

4.13 WaveManager

Ta struktura przechowuje opis kolejnych fal, aktualną ilość monet i punktów wytrzymałości oraz aktualny postęp przejścia poziomu.

5 Możliwe modyfikacje

W pliku *EnemyManger.c* użytkownik może zmieniać statystki okrętów. Podobnie, w pliku *TurretManager.c* ma on dostęp do statystyk wież. Przy zachowaniu odpowiedniej struktury można zmieniać tekstury, które znajdują się w folderze *Assets*. Istnieje też możliwość zmiany map i opisów fal w folderze *Maps*. Każdy pli opisujący mapę składa się z:

- prostokąta 10x18, w którym cyfra 1 oznacza ląd, a 0 wodę
- dwóch współrzędnych, które wyznaczają miejsce pojawiania się przeciwników
- początkowego kierunku przeciwnika (0 - w dół, 1 - w prawo itd.)
- dwóch współrzędnych, które wyznaczają cel, do którego dąży przeciwnik

Jeżeli mapa będzie poprawna, to program sam wyznaczy najkrótszą trasę dla wrogich jednostek.
Każdy plik opisujące fale wrogów składa się z:

- początkowej ilości puntków wytrzymałości
- początkowej ilości monet
- ilości fal
- dwulinijkowych opisów fal:
  - pierwsza linia wyznacza ilość przeciwników w fali i czas oczekiwania do kolejnej fali
  - druga lina to opis kolejnych przeciwników w postaci jednej cyfry od 0 do 2, która określa rodzaj przeciwnika