Politechnika Śląska

Wydział Matematyk Stosowanej

Kierunek Informatyka

Gliwice, 01.10.2020

Programowanie I

**projekt zaliczeniowy**

**"*Gra w Węża*"**

**Piotr Karolak gr. lab. 2**

**1. Opis projektu.**

Prosta gra w wężyka, tworzona z użyciem biblioteki SFML i programowania obiektowego. Wężykiem sterujemy za pomocą strzałek. Porusza się on po siatkowanej planszy. Na niej pojawiają się owoce, które zwiększają długość węża. Gra kończy się w momencie trafienia na przeszkodę.

**2. Wymagania**

* Zmiana rodzaju mapy(pusta, brzegi to ściany, z przeszkodami)
* Cała mapa będzie posiadać tekstury
* W przypadku wyjechania poza brzeg, który nie jest ścianą, wąż wychodzi po drugiej stronie.
* Liczenie wyniku i zapisywanie go do pliku
* Owoce pojawiają się w wolnym i losowym miejscu

**3. Przebieg realizacji**

**Github:** [**https://github.com/P3T3R3/SnakeGame**](https://github.com/P3T3R3/SnakeGame)

Projekt składa się z 2 klas o nazwach „pole” i „terrarium”. Do obsługi grafiki użyto biblioteki SFML.

Klasa pole przechowuje informacje na temat pojedynczego pola po którym porusza się wąż, są to m.in. koordynaty pola, jego tekstura, czy jest ścianą, a także „wagę” węża, która jest istotna dla algorytmu gry.

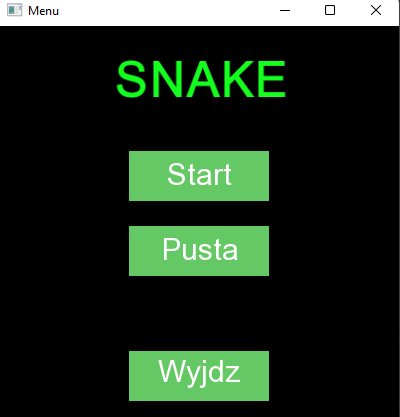
Klasa terrarium obsługuje całą logikę gry, przechowuje dynamiczną tablicę 8x8 obiektów typu pole, które inicjuje w zależności od typu wybranej mapy. Klasa posiada szereg funkcji obsługujących grę:

* Funkcja void Terrarium::move() zawiera algorytm, którego zadaniem jest sprawdzanie jak długo poszczególny segment węża pozostanie na danym polu. Sprawdza on „wagę” węża, jeśli waga osiągnie 1 to dane pole jest resetowane do stanu pustego, a jeśli jest większa od 1 to zostaje pomniejszona o 1. Funkcja sprawdza również czy wąż nie trafi w zajęte pole (swoje ciało lub ścianę), jeśli nie, to wąż porusza się na to miejsce, jeśli tak, to gra się kończy. Nowo zajęte pole otrzymuje wagę pola równą długości węża, a tekstura jest aktualizowana. Wejście węża na owoc zwiększa jego długość o 1.
* Funkcja void Terrarium::updateGameState(sf::RenderWindow \*window, sf::Clock \*gameClock, sf::Clock \* fruitClock) Odpowiada za wyświetlanie planszy i tempo gry(wywołuje move(); i generateFruit(); w odpowiednim czasie, ogranicza prędkość węża i częstotliwość pojawiania się owoców). Gdy gra się skończy gra jest zamrażana.
* Funkcja void Terrarium::generateFruit() generuje owoc w losowym i wolnym miejscu na mapie.

Wszystkie tekstury, czcionki i wartości początkowe są ładowane w momencie utworzenia obiektu typu terrarium. Tekstury znajdują się w folderze media/textures/

**4. Instrukcja użytkownika**

*Opis działania stworzonego programu ze zrzutami ekranów ilustrujące sposób działania programu. Krótka instrukcja obsługi*



Klikając w przycisk pod przyciskiem start możemy zmieniać typ mapy, klikając „Start” rozpoczynamy grę, przyciskiem „Wyjdź” wychodzimy. Do sterowania wężykiem używa się klawiszy strzałek. Celem gry jest zebranie jak największej ilości owoców bez trafienia na ciało węża lub ścianę(poza mapą pustą). Gdy gra się skończy, wyjście do menu następuje po naciśnięciu klawisza escape.Obraz zawierający tekst, stacjonarne, koperta

Opis wygenerowany automatycznie

Obraz zawierający tekst

Opis wygenerowany automatycznie

**5. Podsumowanie i wnioski.**

* Zmiana rodzaju mapy(pusta, brzegi to ściany, z przeszkodami) w menu
* Cała mapa będzie posiadać tekstury
* W przypadku wyjechania poza brzeg, który nie jest ścianą, wąż wychodzi po drugiej stronie.
* Owoce pojawiają się w wolnym i losowym miejscu
* Gra posiada licznik wyniku i wyświetla go po zakończonej rozgrywce oraz zapisuje go do pliku wyniki.txt

**Pomysły na rozwój:**

* Po wejściu na owoc stosowany jest losowy efekt(możliwość przechodzenia przez przeszkody, spowolnienie czasu, przyśpieszenie czasu, mnożnik długości węża)
* Możliwość zmiany rozmiaru mapy
* Animacja węża
* Możliwość personalizacji węża
* Efekty dźwiękowe