Gra "Achtung, die Kurve!"

Jakub Barszczewski, Yuliya Barshcheuskaya, v.2.0
Dokumentacja końcowa

1 Temat oraz opis problematyki projektu

Projektem jest implementacja gry komputerowej zwaną ***"Achtung, die Kurve!"***. Gra polega na sterowaniu kropkami, poruszającymi się po planszy, w taki sposób, by uniknąć kolizji z liniami, zostawionymi przez innych graczy, oraz z krawędziami planszy.

2 Funkcjonalności zrealizowane

Nasz projekt realizuje następujące funkcjonalności:

- Sterowanie przez graczy kropkami, które zostawiają ślady na planszy
- Pojawianie dziur w pozostawianym przez kropkę śladzie
- Detekcja kolizji i reakcja na nie (zakończenie gry i wygrana jednego z graczy)
- Wyświetlanie na końcu koloru gracza, który wygrał
- Możliwość rozgrywki wieloosobowej (4-osobowej) na jednym komputerze i dobrania klawiszy sterujących kropkami (klawisze dostępne to litery oraz cyfry)

3 Funkcjonalności nie zrealizowane

- Różnorodne efekty pojawiające się na planszy - zdeycowaliśmy na odrzucenie pomysłu o zrobieniu efektów z powodu odejścia członka grupy projektowej i większego nakładu pracy niż przedwidywaliśmy.
- Zliczanie punktów - zdeycowaliśmy, aby nasza gra trwała jedną rundę i co każdą rundę można by było edytować graczy. Takie podejście pozwala na szybszą modyfikację klawiszy sterujących i dodawanie graczy, jednak przez to powstaje potrzeba uzupełniania klawiszy za każdą rundą.

4 Narzędzia i biblioteki

Do implementacji projektu użyliśmy podanych poniżej narzędzi i bibliotek - nic nie zmieniło się w stosunku do dokumentacji początkowej.

- Język programowania: **C++**
- System budowania: **CMake**
- Biblioteka graficzna: **RayLib**
- Biblioteka do testowania: **GoogleTest**

W trakcie wykonania projektu mieliśmy kilka kłopotów z systemem CMake i napisaniem go w taki sposób, aby móc poprawnie korzystać z testów i biblioteki RayLib.

5 Podjęte decyzje projektowe

5.1 Klasa Game

Klasa `Game` zarządza główną logiką gry "Achtung die Kurve". Jest odpowiedzialna za rozpoczęcie, zakończenie i aktualizację stanu gry. Korzysta z instancji klasy `GameState` do przechowywania stanu gry oraz instancji klasy `Board` do obsługi logiki gry i renderowania.

Atrybuty

- `state`: (`GameState`) - instancja przechowująca stan gry.
- `board`: (`Board`) - instancja obsługująca logikę i renderowanie gry.

Metody

- `Game()` - konstruktor inicjalizujący instancję `board` z `state`.
- `void run()` - metoda uruchamiająca główną pętlę gry.
- `GameState get_state() const` - getter zwracający bieżący stan gry (głównie do celów testowych).
- `void set_state(const GameState& new_state)` - setter ustawiający nowy stan gry (do celów testowych).

5.2 Klasa Board

Klasa Board odpowiada za wyświetlanie planszy oraz elementów na niej. Ogólnie zawiera ona informacje o planszy i zarządza przetwaraniem kolizji.

Atrybuty

- `state`: (`GameState&`) - referencja do współdzielonej instancji `GameState`.
- `screen_width`: (`const float`) - szerokość ekranu gry.
- `screen_height`: (`const float`) - wysokość ekranu gry.
- `buttons`: (`std::vector<ButtonData>`) - wektor przechowujący dane przycisków.

Metody

- `Board(GameState& state)` - konstruktor inicjalizujący instancję `state` oraz przyciski.
- `void handle_title_screen(Vector2 mousePoint)` - metoda obsługująca logikę ekranu tytułowego.
- `void handle_gameplay_screen()` - metoda obsługująca logikę ekranu rozgrywki.
- `void handle_score_screen()` - metoda obsługująca logikę ekranu wyników.
- `void draw_title_screen()` - metoda rysująca ekran tytułowy.
- `void draw_gameplay_screen()` - metoda rysująca ekran rozgrywki.
- `void draw_score_screen()` - metoda rysująca ekran wyników.
- `void check_collisions(std::vector<Snake>& Players)` - metoda sprawdzająca kolizje w grze.
- `float get_screen_width() const` - getter zwracający szerokość ekranu.
- `float get_screen_height() const` - getter zwracający wysokość ekranu.

5.3 Struktura Button

Wybraliśmy, aby guzik był strukturą, ponieważ nie posiada żadnych metod, jedynie wiele atrybutów. Dana struktura służy do przetrzymywania informacji o kliknięciu klawiszy, a też o wybranych klawiszach sterowania.

Atrybuty

- symbole klawiszy do sterowania
- granice guzika
- kolor guzika
- oraz flagi dzięki którym rysowane są na planszy klawisze i podświetlanie guziki

5.4 Klasa Snake

Przechowuje infromacje o graczu i jego wężu. Również rysuje węże na planszy i sprawdza kolizje.

Atrybuty

- klawisze sterujące
- prędkość, pozycja, kąt głowy węża
- ślad węża

Metody

- renderowanie węża i jego ruch
- sprawdzanie różnych typów kolizji

5.5 Klasa GameState

Klasa `GameState` przechowuje stan gry, który jest współdzielony pomiędzy klasami `Board` i `Game`. Zawiera informacje dotyczące aktualnego ekranu gry, stanu graczy oraz flagi kontrolujące różne aspekty gry.

Atrybuty

- `currentScreen`: (`GameScreen`) - aktualny ekran gry (możliwe wartości: `TITLE`, `GAMEPLAY`, `SCORE`).
- `Players`: (`std::vector<Snake>`) - wektor przechowujący graczy (węże).
- `gameInProgress`: (`bool`) - flaga wskazująca, czy gra jest w toku.
- `gameOver`: (`bool`) - flaga wskazująca, czy gra się zakończyła.
- `gameOverStartTime`: (`double`) - czas rozpoczęcia ekranu końca gry.
- `gameOverDuration`: (`const int`) - czas trwania ekranu końca gry (w sekundach).
- `countdownActive`: (`bool`) - flaga wskazująca, czy aktywne jest odliczanie przed rozpoczęciem gry.
- `countdownValue`: (`int`) - wartość odliczania.
- `countdownStartTime`: (`double`) - czas rozpoczęcia odliczania.
- `insufficientPlayersMessageTime`: (`double`) - czas rozpoczęcia wyświetlania komunikatu o niewystarczającej liczbie graczy.
- `insufficientPlayersMessageActive`: (`bool`) - flaga wskazująca, czy aktywny jest komunikat o niewystarczającej liczbie graczy.

Metody

- `GameState()` - konstruktor domyślny.
- `GameState(const GameState& other)` - konstruktor kopiujący.
- `GameState& operator=(const GameState& other)` - operator przypisania kopiującego.
- `std::tuple<Color, std::string> get_winner_color() const` - metoda zwracająca kolor i nazwę koloru zwycięzcy.

6 Opis realizacji projektu

Projekt w większości został zrealizowany zgodnie z dokumentacją wstępną. To, co nie zostało zrealizowane, zostało opisane w punkcie o nie zrealizowanych funckjonalnościach. Dużym problemem było to, że jedna osoba zrezygnowała z robienia projektu, więc cały kod był pisany przez pozostałe osoby, jedynie dokumentacja wstępna była robiona przez 3. Podzieliliśmy się w ten sposób, że ekran TITLE robi jedna osoba, a ekrany GAMEPLAY oraz SCORE druga. W taki sposób bez większych zmian pracowaliśmy do końca, zlecając sobie nawzajem co trzeba naprawić. Wzorowaliśmy się przykładowymi programami udostępnianamy przez autorów biblioteki Raylib ze strony raylib.com. Co do architektury, to użyliśmy nie dziedziczenia, ale kompozycji, ponieważ w naszym projekcje nie narzucało się nigdzie użycie dziedziczenia.

7 Uruchomienie

Program po zbudowaniu w CMake uruchamia się wraz z uruchomieniem kodu. Po odpaleniu gry pojawi się ekran dodawania graczy. Aby dodać gracza należy nacisnąć na przycisk o nazwie koloru (na przykład RED), i potem nacisnąć na klawiaturze dwa przyciski do sterowania wężem (na początku lewy, a potem prawy). Przyciskami mogą być znaki z tabeli ASCII od 32 do 126, czyli głownie cyfry i alfabet, bez strzałek. Po wybraniu graczy należy nacisnąć klawisz ENTER. Wtedy pojawią się gracze wraz z ukazanym początkowym kierunkiem ruchu. Po odczekaniu chwili strzałki zaczynają się ruszać. Sterować swoim graczem można za pomocą wybranych klawiszy aż do pozostania jednego gracza na planszy. Po tym pojawia się ekran, który wyświtla zwycięzcę. Po naciśnięciu ENTER można przejść z powrotem do ekranu początkowego i zacząć następną rundę gry.