Programowanie obiektowe 2 (Projekt)	
Temat projektu: Gra Space Invaders	
Wiktor Górnicz **Jakub Karaś**	**Semestr zimowy 2023/24** **2ID12B**

1. **Ogólny opis projektu, technologie, frameworki i biblioteki**

Space Invaders to gra oparta na silniku LibGDX, która oferuje funkcje takie jak: menu graficzne z obsługą przycisków, obsługa gry, resetowanie poziomu, zliczanie punktów.

Technologie, Frameworki i Biblioteki wykorzystane w projekcie:

- LibGDX - główny silnik gry, który zapewnia podstawową infrastrukturę do stworzenia gry. W naszym przypadku obejmuje on grafikę, fizykę oraz wiele innych modułów.
- JUnit 5 – JUnit wykorzystywany jest do tworzenia testów jednostkowych w projekcie. Umożliwia on testowanie różnych komponentów gry, co pomaga w utrzymaniu wysokiej jakości kodu.
- Mockito – Mockito to narzędzie do tworzenia atrap obiektów podczas testowania. Pomaga w symulowaniu zachowań i testowaniu komponentów niezależnie.
- LWJGL (Lightweight Java Game Library) - to niskopoziomowa biblioteka do programowania gier w języku Java. Wykorzystywana jest do obsługi OpenGL i dostępu do niskopoziomowych funkcji systemu.

Wersje Technologii:

- LibGDX: 1.10.0
- JUnit: 5.9.0
- Mockito: 3.12.4
- LWJGL: 3.3.0

2. **Informacje na temat funkcjonalności projektu**

Projekt jest w pełni funkcjonalną grą Space Invaders, polegającej na kontrolowaniu statku i wystrzeliwania pocisków, przy pomocy których pokonujemy kosmitów. Projekt posiada obsługę myszy i klawiatury, którymi kontrolujemy statek oraz oddajemy strzał (Lewy przycisk myszy służy do oddania strzału, klawisze A i D do poruszania się statkiem). Gra obsługuje menu graficzne z dwoma przyciskami, przycisk Play przenosi nas do rozgrywki, przycisk Exit zamyka aplikację. Po wciśnięciu przycisku Play zostają uruchomione funkcje renderujące plansze, kosmitów, statek gracza. Gra jest kontynuowana aż do momentu przegranej gracza (zderzenie się z kosmitami, lub przesunięcie się kosmitów na sam dół planszy – gra posiada funkcję, dzięki której kosmici poruszają się po planszy, co jakiś czas zwiększając swoją prędkość). Głównym celem gracza jest zdobycie jak największej ilości punktów, które są zliczane, a następnie najwyższy wynik jest zapisywany do pliku tekstowego i jest on odczytywany w menu tekstowym. W przypadku zestrzelenia wszystkich kosmitów gra tworzy nowe postacie, które pojawiają się na pozycji początkowej. Po przegranej gra przenosi nas do menu końca gry i umożliwia nam ponowną rozgrywkę lub opuszczenie aplikacji.

3. **Informacje na temat sposobu uruchomienia oraz obsługi projektu**

Projekt został przetestowany w środowisku IntelliJ IDEA 2023.2.3, uruchamiamy go po przez wczytanie projektu a następnie uruchamiamy klasę DekstopLauncher.

4. **Informacje na temat stworzonych klas, metod, funkcji**

Klasa Alien:

- Klasa Alien służy do reprezentowania obiektów typu Alien w grze i umożliwia ich rysowanie na określonym SpriteBatch.

Pola klasy:

- position - aktualna pozycja obiektu typu Vector2.
- position_initial - pozycja początkowa obiektu typu Vector2.
- sprite - obiekt typu Sprite reprezentujący wygląd obiektu Alien.
- Alive - flaga określająca, czy obiekt Alien jest żywy.

Konstruktor klasy Alien:

- Parametry konstruktora: _position (typu Vector2) i img (typu Texture).
- Tworzy nowy obiekt Alien na podstawie dostarczonej pozycji i tekstury.
- Ustawia pozycję początkową na aktualną pozycję.
- Ustala skalę obiektu sprite na 2.

Metoda Draw:

- Parametr metody: batch (typu SpriteBatch).
- Aktualizuje pozycję sprite na podstawie aktualnej pozycji obiektu Alien.
- Rysuje obiekt sprite na danym SpriteBatch.

Klasa DesktopLauncher:

- Jest odpowiedzialna za uruchomienie gry na platformie desktopowej.

Metoda main:

- Jest główną metodą, która rozpoczyna uruchamianie gry.
- Tworzy instancję konfiguracji Lwjgl3ApplicationConfiguration (config) do dostosowania ustawień gry na platformie desktopowej.
- Ustawia maksymalną liczbę klatek na sekundę na 60 przy użyciu config.setForegroundFPS(60).
- Ustawia tytuł okna na "Space Invaders" za pomocą config.setTitle("Space Invaders").
- Ustawia rozmiar okna na 600x600 pikseli za pomocą config.setWindowedMode(600, 600).
- Następnie tworzy nową instancję klasy Main (głównej klasy gry) i uruchamia grę na platformie desktopowej, przekazując Main i obiekt konfiguracji config do Lwjgl3Application.

Klasa Game:

- Klasa Game jest odpowiedzialna za zarządzanie logiką i renderowaniem gry oraz śledzenie wyniku i kolizji między graczem a kosmitami.

Pola klasy Game:

- player: Obiekt gracza (Player).
- aliens: Tablica obiektów kosmitów (Alien).

- score: Zmienna przechowująca aktualny wynik gry.
- NumWidth_aliens: Liczba kosmitów w jednym wierszu.
- NumHeight_aliens: Liczba kosmitów w jednej kolumnie.
- spacing_aliens: Odległość między kosmitami.
- minX_aliens, minY_aliens, maxX_aliens, maxY_aliens: Minimalne i maksymalne współrzędne X i Y kosmitów.
- amount_alive_aliens: Liczba żywych kosmitów.
- direction_aliens: Kierunek ruchu kosmitów (-1 lub 1).
- speed_aliens: Prędkość poruszania się kosmitów.
- offset_aliens: Przesunięcie kosmitów.
- img_bullet: Tekstura pocisku.
- img_alien: Tekstura kosmita.
- batch: Obiekt SpriteBatch do renderowania.
- font: Obiekt BitmapFont do wyświetlania tekstu na ekranie.

Konstruktor klasy Game:

- Inicjalizuje wszystkie pola klasy, tworzy obiekt gracza, kosmitów i ładuje tekstury.

Metoda initializeAliens:

- Inicjalizuje tablicę kosmitów, ustawiając ich początkowe pozycje i ustawia flagę Alive na true.

Metoda resetGame:

- Resetuje stan gry, ustawiając wynik na 0 i wywołując resetAliens() oraz resetowanie gracza.

Metoda resetAliens:

- Resetuje pozycje kosmitów i ustawia flagę Alive na true.

Metoda getScore:

- Zwraca aktualny wynik gry.

Metoda renderGame:

- Renderuje stan gry na podstawie parametru deltaTime (czas między klatkami).
- Wykrywa kolizje między pociskami gracza a kosmitami, aktualizując wynik.
- Zarządza ruchem kosmitów w odpowiednich kierunkach.
- Wykrywa kolizje między graczem a kosmitami, co prowadzi do zakończenia gry.
- Wyświetla wynik na ekranie.

Metoda dispose:

- Zwalnia zasoby gry, takie jak tekstury i czcionka.

Enum GameState:

- GameState to enum, który reprezentuje różne stany gry.

Game State składa się z trzech wartości:

- MENU: Stan gry, w którym gracz znajduje się w menu głównym.
- PLAYING: Stan gry, w którym gra jest w trakcie, czyli gracz jest aktywny i kontroluje postać lub interaguje z grą.

- GAME_OVER: Stan gry, w którym gra została zakończona, na przykład gdy gracz przegrał lub osiągnął pewne warunki końca gry.

Enum pozwala na łatwe śledzenie i kontrolowanie stanu gry w aplikacji, co może być przydatne do wyświetlania odpowiednich ekranów, interfejsów użytkownika i logiki gry w zależności od aktualnego stanu. Na przykład można użyć enum GameState, aby zmieniać widoki między menu, rozgrywką a ekranem końca gry w zależności od tego, w którym stanie gra się znajduje.

Klasa GameStateManager:

- GameStateManager to klasa odpowiedzialna za zarządzanie stanem gry. Skupia się na śledzeniu i ustawianiu aktualnego stanu gry. Dzięki klasie GameStateManager można łatwo kontrolować i monitorować stan gry w różnych częściach aplikacji.

Pole klasy GameStateManager:

- currentState: Pole statyczne przechowujące aktualny stan gry (enum GameState).

Metoda getCurrentState:

- Jest to metoda statyczna, która zwraca aktualny stan gry.

Metoda setCurrentState:

- Jest to metoda statyczna, która pozwala na ustawienie nowego stanu gry na podstawie dostarczonego argumentu newState. Ta metoda może być używana do zmiany stanu gry z poziomu innych części aplikacji.

KlasaMain:

- Klasa Main pełni rolę centralnego punktu zarządzania całym przebiegiem gry, od menu początkowego, przez rozgrywkę, aż po ekran Game Over i zarządzanie wynikami gracza.

Pola klasy Main:

- batch: Obiekt SpriteBatch do renderowania grafiki.
- playButtonTexture: Tekstura przycisku "Play".
- exitButtonTexture: Tekstura przycisku "Exit".
- backgroundTexture: Tekstura tła gry.
- game: Obiekt klasy Game reprezentujący logikę gry.
- menu: Obiekt klasy Menu reprezentujący menu główne gry.
- menuGameOver: Obiekt klasy MenuGameOver reprezentujący ekran końca gry.
- scoreManager: Obiekt klasy ScoreManager do zarządzania wynikami gry.
- font: Obiekt BitmapFont do wyświetlania tekstu na ekranie.

Metoda create:

- Metoda inicjalizująca grę i jej komponenty.
- Ustawia początkowy stan gry na MENU za pomocą GameStateManager.setCurrentState(GameState.MENU).
- Tworzy obiekty SpriteBatch, wczytuje tekstury, inicjalizuje obiekty gry, wyników i menu.

Metoda render:

- Metoda renderująca całą grę i odpowiedzialna za zmiany stanów gry.

- Rozpoczyna rysowanie na ekranie za pomocą batch.begin().
- Renderuje tło gry i obsługuje logikę gry w zależności od aktualnego stanu gry (GameStateManager.getCurrentState()).
- W przypadku stanu MENU obsługuje interakcje w menu i renderuje menu.
- W przypadku stanu PLAYING renderuje grę.
- W przypadku stanu GAME_OVER obsługuje interakcje na ekranie Game Over, renderuje ten ekran, zapisuje wynik gracza i resetuje stan gry.
- Kończy rysowanie na ekranie za pomocą batch.end().

Metoda dispose:

- Metoda zwalniająca zasoby używane w grze, takie jak obiekt SpriteBatch i tekstury.

Klasa Menu:

- Klasa Menu obsługuje więc interakcje użytkownika w menu głównym gry i renderuje menu na ekranie, wraz z wyświetlaniem najlepszego wyniku.

Pola klasy Menu:

- batch: Obiekt SpriteBatch do renderowania menu.
- buttonWidth: Szerokość przycisków menu.
- buttonHeight: Wysokość przycisków menu.
- font: Obiekt BitmapFont do wyświetlania tekstu na ekranie.
- playButtonTexture: Tekstura przycisku "Play".
- exitButtonTexture: Tekstura przycisku "Exit".
- scoreManager: Obiekt klasy ScoreManager do zarządzania wynikami gry.

Konstruktor klasy Menu:

- Inicjalizuje pola klasy, tworzy obiekt BitmapFont i wczytuje tekstury przycisków "Play" i "Exit".

Metoda handleMenuInput:

- Obsługuje interakcje użytkownika w menu.
- Sprawdza, czy użytkownik kliknął lewym przyciskiem myszy na przycisk "Play" lub "Exit" i podejmuje odpowiednie akcje w zależności od kliknięcia.
- Jeśli użytkownik kliknie "Play", ustawia stan gry na PLAYING.
- Jeśli użytkownik kliknie "Exit", zamyka aplikację.

Metoda renderMenu:

- Renderuje menu na ekranie.
- Wyświetla przyciski "Play" i "Exit" w odpowiednich pozycjach.
- Wczytuje najlepszy wynik gry z scoreManager i wyświetla go na ekranie.

Klasa MenuGameOver:

- Klasa MenuGameOver obsługuje interakcje użytkownika na ekranie Game Over i renderuje menu na ekranie, wraz z wyświetlaniem najlepszego wyniku.

Pola klasy MenuGameOver:

- batch: Obiekt SpriteBatch do renderowania menu.
- buttonWidth: Szerokość przycisków menu.

- buttonHeight: Wysokość przycisków menu.
- font: Obiekt BitmapFont do wyświetlania tekstu na ekranie.
- playButtonTexture: Tekstura przycisku "Play".
- exitButtonTexture: Tekstura przycisku "Exit".
- scoreManager: Obiekt klasy ScoreManager do zarządzania wynikami gry.

Konstruktor klasy MenuGameOver:

- Inicjalizuje pola klasy, tworzy obiekt BitmapFont i wczytuje tekstury przycisków "Play" i "Exit".

Metoda handleGameOverInput:

- Obsługuje interakcje użytkownika na ekranie Game Over.
- Sprawdza, czy użytkownik kliknął lewym przyciskiem myszy na przycisk "Play" lub "Exit" i podejmuje odpowiednie akcje w zależności od kliknięcia.
- Jeśli użytkownik kliknie "Play", ustawia stan gry na PLAYING.
- Jeśli użytkownik kliknie "Exit", zamyka aplikację.

Metoda renderGameOver:

- Renderuje menu po zakończeniu gry na ekranie.
- Wyświetla przyciski "Play" i "Exit" w odpowiednich pozycjach.
- Wczytuje najlepszy wynik gry z scoreManager i wyświetla go na ekranie.

Klasa Player:

- Klasa Player obsługuje logikę i renderowanie gracza oraz jego pocisków w grze.

Pola klasy Player:

- position: Aktualna pozycja gracza jako obiekt Vector2.
- position_bullet: Aktualna pozycja pocisku gracza jako obiekt Vector2.
- sprite: Sprite reprezentujący gracza.
- sprite_bullet: Sprite reprezentujący pocisk gracza.
- speed: Prędkość poruszania się gracza.
- speed_bullet: Prędkość poruszania się pocisku gracza.

Konstruktor klasy Player:

- Inicjalizuje sprite gracza i sprite pocisku gracza.
- Ustawia skalę i kolor gracza.
- Ustawia początkową pozycję gracza na środku ekranu poziomo i na wysokości 10 pikseli od góry ekranu.
- Ustawia początkową pozycję pocisku gracza poza widokiem ekranu (na wysokości 10 000 pikseli), aby początek gry nie wyświetlał się od razu pocisk.

Metoda update:

- Aktualizuje logikę gracza na podstawie wejścia użytkownika i czasu deltaTime.
- Obsługuje strzelanie gracza w lewo (Keys.A) i w prawo (Keys.D), ustawiając odpowiednie prędkości pocisku.
- Ogranicza ruch gracza do widocznego obszaru ekranu.
- Aktualizuje pozycję pocisku gracza, aby przesuwał się w górę ekranu.

Metoda draw:

- Aktualizuje pozycję gracza i pocisku gracza na podstawie logiki gry.
- Rysuje gracza i pocisk gracza na ekranie za pomocą obiektu SpriteBatch.

Metoda reset:

- Resetuje pozycję gracza i pocisku gracza do pozycji startowej.
- Pozycja gracza jest ustawiana na środku ekranu poziomo i na wysokości 10 pikseli od góry ekranu.
- Pozycja pocisku gracza jest ustawiana poza widokiem ekranu (na wysokości 10 000 pikseli).

ScoreManager:

- Klasa ScoreManager pozwala na wczytywanie, zapisywanie i pobieranie aktualnie najlepszego wyniku gry. Wczytywanie odbywa się przy inicjalizacji obiektu ScoreManager, a zapisywanie zachodzi tylko wtedy, gdy gracz uzyska wynik wyższy od aktualnie najlepszego.

Pole klasy ScoreManager:

- currentTopScore: Przechowuje aktualnie najlepszy wynik gry.
- highScoreFileName: Przechowuje nazwę pliku, w którym jest przechowywany najlepszy wynik (domyślnie "highscore.txt").

Konstruktor klasy ScoreManager:

- Inicjalizuje obiekt ScoreManager i wczytuje aktualnie najlepszy wynik z pliku za pomocą metody loadHighScore().

Metoda loadHighScore:

- Wczytuje aktualnie najlepszy wynik z pliku o nazwie przechowywanej w polu highScoreFileName.
- Jeśli plik nie istnieje lub nie zawiera liczby całkowitej, to aktualny najlepszy wynik pozostaje niezmieniony.

Metoda saveTopScore:

- Zapisuje nowy najlepszy wynik do pliku tylko wtedy, gdy nowy wynik jest wyższy od aktualnie przechowywanego wyniku.
- Jeśli nowy wynik jest zapisywany, to nadpisuje poprzednią wartość w pliku.

Metoda getCurrentTopScore:

- Zwraca aktualnie najlepszy wynik gry.

5. Informacje na temat ilości pracy

Praca nad projektem była przeprowadzana równomiernie, z równym wkładem każdego z uczestników. Każdy członek zespołu aktywnie uczestniczył w procesie tworzenia kodu i przykładał się do projektu w równym stopniu. Choć niektóre klasy mogły być początkowo pisane przez konkretne osoby, to cały zespół wspólnie dokładnie przejrzał, poprawił i dopracował każdy aspekt projektu. Jakub w głównej mierze odpowiada za stworzenie Menu (klasy Menu, MenuGameOver) oraz poprawną obsługę wyniku (ScoreManager), natomiast Wiktor przygotował podstawową funkcjonalność gry (Alien, Game,

GameState, GameStateManager, Player). Nasza współpraca była oparta na równości i zaangażowaniu każdego uczestnika, co przyczyniło się do sukcesu całego przedsięwzięcia.