Przedmiot: Programowanie obiektowe (Java) - projekt	Temat: Gra Snake
Seweryn Cholewa	2ID12B, III rok informatyka

1. Ogólny opis projektu

W ramach projektu podjęto się stworzenia gry 2d przypominającą grę Snake. W ramach działania gry (programu) dostajemy możliwość sterowania wężem i naszym zadaniem jest jak najdłuższe zbieranie jedzenia co zwiększa rozmiar ciała węża bez ugryzienia się, czyli doprowadzenia do sytuacji, że głowa znajdzie się w miejscu, gdzie znajduje się już jakiś inny element ciała. W celu stworzenia gry wykorzystano język Java (wersja 8), bibliotekę libgdx przeznaczoną do tworzenia gier opierającą się na OpenGL, narzędzie konfiguracyjne Gradle (7.0) oraz system kontroli wersji w postaci githuba.

2. Funkcjonalności projektu

- Menu z możliwymi do kliknięcia myszką przyciskami
- Możliwość sterowania wężem na ekranie gry za pomocą klawiszy WSAD i strzałek
- Kolizje z ciałem węża skutkujące zakończeniem gry
- Możliwość ponownego uruchomienia rozgrywki po przegranej
- Możliwość włączenia pauzy w trakcie gry
- Pokazywanie aktualnego wyniku gracza na ekranie gry
- Zapisywanie stanu gry i możliwość kontynuowania ostatnio przerwanej rozgrywki (nawet po wyłączeniu programu)
- Zapisywanie najlepszego wyniku i pokazywanie go na ekranie gry
- Zwiększanie trudności rozgrywki po osiągnięciu konkretnych wyników punktowych przez zwiększenie prędkości poruszania węża

3. Sposób uruchomienia i obsługa

Istnieją dwa sposoby uruchomienia programu:

1. Korzystając z pliku snake.jar trzeba upewnić się, że wszelkie pliki zasobowe, czyli obrazki itp. były bezpośrednio razem z plikiem snake.jar i aby uruchomić program wystarczy go dwa razy kliknąć myszką.
2. Za pomocą IDE (w moim przypadku intelliJ IDEA) otwieramy plik build.gradle jako projekt, a następnie konfigurujemy opcje run ustawiając jako główny moduł Snake1.lwjgl3.main, klasę zawierającą główną metodę jako game.snake.lwjgl3.Lwjgl3Launcher oraz ustawić working directory jako Snake1\assets. Po ustawieniu tych wszystkich opcji możemy uruchomić program korzystając z opcji run.

4. Klasy, funkcje i metody

Ważna informacja:

W projekcie wykorzystałem jedną rzecz udostępnioną w ramach licencji open-source, które są dostępne do użytku dla wszystkich osób, które chcą ich użyć. Jedna z nich jest w folderze /assets/skin i są to pliki określające wygląd przycisków używanych w mojej grze, a wykorzystałem je, ponieważ biblioteka libgdx nie posiada wyglądu standardowych przycisków (a nie jest to proste, żeby zrobić je samemu) dlatego żeby przyśpieszyć dokończenie projektu skorzystałem z darmowych i ogólno dostępnych wzorców. Dodatkowo część klas, a dokładnie MainMenuScreen i GameScreen nie zostały przetestowane co wynika z dwóch rzeczy: po pierwsze klasy te wymagają uruchomienia aplikacji jak w klasie Lwjgl3Launcher (czego nie potrafię zrobić), aby funkcjonalności z biblioteki gdx działały, a po drugie to większość zastosowanych tam funkcji została przetestowana w testach innych klas.

Klasa Lwjgl3Launcher – klasa wymagana przez bibliotekę libgdx, która służy do konfiguracji i uruchamiania aplikacji, posiada funkcje takie jak:

- Lwjgl3ApplicationConfiguration getDefaultConfiguration() – funkcja, w której możemy ustawić podstawową konfiguracje aplikacji taką jak tytuł okna, jego wielkość, ikonę itp. , a która zwraca obiekt typu Lwjgl3ApplicationConfiguration

- Void createApplication() – funkcja, której jedynym zadaniem jest uruchomienie aplikacji z parametrami podanymi jej przez wcześniej wspomnianą funkcję, nie zwraca niczego

Klasa Apple – jest to klasa, której obiekt reprezentuje jedzenie (dokładnie jabłko) zjadane przez węża. Zawiera metody takie jak:

- Void draw(Batch batch) – metoda przyjmująca obiekt typu batch jako parametr, a której efektem działania jest „narysowanie" obiektu jabłka na ekranie z podaną jej tekturą i koordynatami, nie zwraca niczego
- Void setPosition(GridPoint2 position) – funkcja ustawiająca pozycje danego obiektu w miejscu podanym mu przez parametr w postaci obiektu GridPoint2 (czyli po prostu punktu w siatce 2D), nie zwraca niczego
- GridPoint2 getPosition() – metoda służącą do pobrania położenia konkretnego obiektu typu Apple, zwraca obiekt typu GridPoint2, która zawiera współrzędne x i y

Klasa Snake – jest to klasa, której obiekt reprezentuje węża, którym gracz może poruszać. Zawiera metody takie jak:

- Void initialize() – funkcja, która odpowiada za odpowiednie ustawienie parametrów węża w zależności czy rozpoczynamy nową grę czy kontynuujemy poprzednią – czyli ustawia pozycje segmentów ciała węża, kierunek poruszania węża i jego ogona, nie zwraca niczego
- Void act(float deltaTime) –funkcja przyjmująca jako parametr zmienną typu float jaką jest deltaTime (czyli zmiennej informującej ile minęło od „ostatniej" klatki), która określa jakie czynności mają zostać wykonane po wykonania ruchu przez węża np. obsługa zmiany kierunku poruszania, określenie czy wąż może wykonać kolejny ruch itp.), nie zwraca niczego
- Boolean isAppleFound(GridPoint2 applePosition) – funkcja sprawdzająca czy wąż zjadł jabłko, przyjmuje jako parametr położenie jabłka i zwraca wartość true or false

- Void extendedSnake() – funkcja dodająca kolejny segment ciała węża w miejscu za ogonem, nie zwraca niczego
- Boolean hasHitHimself() – funkcja sprawdzająca czy wąż się ugryzł (czy położenie głowy pokrywa się z położeniem innych segmentów ciała), zwraca wartość true lub false w zależności czy do tego doszło czy nie
- draw(Batch batch) – funkcja działająca tak samo jak funkcja z klasy Apple.java czyli rysuje węża na ekranie, z tą różnicą że jest bardziej skomplikowana, ponieważ głowa i ogon węża się zmienia w zależności do kierunku ruchu, nie zwraca niczego
- void handleDirectionChange() – funkcja odpowiadająca za obsługę zmiany kierunku, w zależności od aktualnego kierunku ruchu i wciśniętych klawiszy zmienia zmienną kierunek na odpowiednią wartość enumeratora MovementDirection, nie zwraca niczego
- void move() – funkcja odpowiadająca za zmianę położenia ciała węża po ruchu, nie zwraca niczego
- void determineTailDirection() – funkcja, która służy do obsługi zmiennej wskazującej na kierunek ogona (potrzebna jest osobna zmienna do ogona i głowy, bo ogon działa trochę inaczej), nie zwraca niczego
- GridPoint2 head() – funkcja, która zwraca położenie głowy węża (czyli pierwszego fragmentu ciała)
- int tailIndex() – funkcja, która zwraca indeks ogonu węża (czyli ostatni element ciała)
- List<GridPoint2> getSnakeSegmentPositions() – funkcja zwracająca listę zawierającą współrzędne wszystkich elementów ciała węża
- MovementDirection getDirection() – funkcja zwracająca kierunek ruchu węża (kierunek na, którym opiera się ruch węża i tekstura głowy)
- MovementDirection getTailDirection() – funkcja zwracająca kierunek ruchu ogona

Klasa SnakeGame – klasa, która istnieje po to, aby móc zaimplementować zmiany ekrany opierające się na zmianie klas, które implementują interfejs Screen z biblioteki libgdx. Nie zawiera ona wiele oprócz stworzenia obiektu typu SpriteBatch czy font oraz metody, która się ich pozbywa.

Klasa MainMenuScreen – klasa odpowiadająca za ekran głównego menu. Zawiera funkcje takie jak (teoretycznie klasa ma więcej funkcji, które wynikają z budowy klas, które implementują klasę Screen z libgdx, ponieważ muszą one nadpisywać kilka konkretnych funkcji, ale część z nich nie była mi potrzebna i nie zawiera w sobie żadnego kodu):

- Void render(float deltaTime) – funkcja, która przyjmuje jako parametr deltaTime (który wytłumaczyłem wcześniej) i która określa co ma się zrobić gdy kolejna klatka ma zostać zrenderowana. W przypadku tej klasy powoduje ona „narysowanie" obrazku tła, tytułu gry oraz „aktywuje" obiekt klasy Stage razem z poleceniem narysowania rzeczy się w nim znajdujących (co wytłumaczę dalej), nie zwraca niczego
- Void show() – funkcja, która tworzy obiekt typu Stage, a następnie dodaje do niego aktorów w postaci guzików razem z przypisanymi do nich funkcjami, które mają się wykonać w razie naciśnięcia (następnie te guziki zostaną narysowane przez funkcje render(float deltaTime) o czym wspomniałem wcześniej)
- Void dispose() – funkcja, której zadaniem jest usunięcie z zasobów związanych z teksturami i obiektami, które były potrzebne do działania klasy MainMenuScreen, a które muszą być usunięte, gdy nie są już potrzebne, nie zwraca niczego
- Void hide() – funkcja, której jedynym zadaniem jest wyłączenie odbierania sygnałów od klawiatury i myszki, gdy ten ekran zostanie ukryty, nie zwraca niczego.

Klasa GameScreen – klasa odpowiadająca za właściwy ekran gry (gdy rozpoczniemy grę i zaczniemy poruszać wężem). Zawiera konstruktor w, którym ładowane są wszystkie potrzebne tekstury i tworzone są potrzebne obiekty innych klas jak np. apple czy snake oraz funkcje takie jak:

- void render (float delta) – funkcja działająca na tej samej zasadzie jak ta w klasie MainMenuScreen, z tą różnicą że rysuje na ekranie inne rzeczy (dokładnie te, które są potrzebne w grze, czyli węża, jabłko itp.), nie zwraca niczego

- void show() – funkcja, której zadaniem jest ustawienie listenera, który reaguje na przyciśnięcie przycisku ESC wracając użytkownika do głównego menu, nie zwraca niczego
- void hide() – funkcja określająca co ma zostać zrobione w razie „ukrycia" ekranu, czyli zapisanie gry, zatrzymanie muzyki oraz wyłączenie odbierania sygnałów od klawiatury i myszki, nie zwraca niczego
- void dispose () – podobnie jak w klasie MainMenuScreen funkcja ta odpowiada za pozbycie się rzeczy, które muszą być manualnie usunięte w momencie, kiedy nie są już potrzebne, nie zwraca niczego
- void initializeNewGame() – funkcja określająca co ma zostać wykonane w przypadku rozpoczęcia gry, czyli wywołanie metody ogarniającej obiekt snake oraz obsługującej pozycje jabłka i wynik gracza w zależności czy została rozpoczęta nowa gra, czy kontynuowana jest poprzednia, nie zwraca niczego
- void randomizeApplePosition() – funkcja, która odpowiada za ustawienie pozycji jedzenia (jabłka) w losowym miejscu przy wykorzystaniu funkcji z klasy foodPositionRandomizer, nie zwraca niczego
- void updateGame() – funkcja, której zadaniem jest określenie czynności, które mają zostać wykonane po wyrenderowaniu klatki, a dokładnie sprawdzenie czy wąż zjadł jedzenie i jeśli do tego doszło to zwiększyć ilość punktów gracza, wydłużyć jego ciało i wylosować nową pozycję jedzenia, sprawdzić czy wąż się ugryzł itp., nie zwraca niczego
- void saveGame() – funkcja, która zapisuje pozycje jedzenia, wynik i pozycje ciała węża w celu umożliwienia kontynuacji ostatnio skończonej gry (nawet po zamknięciu aplikacji), nie zwraca niczego
- boolean getContinued() – zwykły getter zwracający wartość flagi continued, która mówi nam czy gra ma być kontynuowana i wczytany ma być zapis czy ma się rozpocząć od nowa
- void setContinued(boolean val) – zwykły setter ustawiający wartość flagi continued
- Preferences getPrefs() – zwykły getter zwracający obiekt typu Preferences (która działa jak hash map), a który jest używany do zapisywania stanu gry
- void saveBestScore() – funkcja służąca zapisywaniu najlepszego wyniku, który został do tej pory osiągnięty przez któregokolwiek gracza, nie zwraca niczego

Klasa FoodPositionRandomizer – klasa odpowiadająca za określenie wolnego pola dla narysowania nowego jedzenia. Zawiera konstruktor i funkcje takie jak:

- GridPoint2 getRandomAvailablePosition(List<GridPoint2> occupiedPositions) – funkcja, która przyjmuje jako parametr listę punktów na ekranie, które są zajęte (czyli po prostu lista punktów, w którym znajduje się ciało węża), określa ona które elementy siatki na ekranie są nie zajęte, a potem losuje jeden z nich, aby na nim narysować kolejne jabłko

Klasa Score – klasa odpowiadająca za obsługę zmiennej zawierającej aktualny wynik gracza. Zawiera ona bardzo proste funkcje, których nie będę opisywał, bo są to jedynie proste settery i gettery.

Oprócz tych klas w projekcie znajduje się publiczny enumerator MovementDirection, który określa możliwe kierunki ruchu.