Université des Mascareignes

Faculty of Information and Communication Technology

Licence Technologique en Informatique Appliquée

3ème année

Semestre 5

POO2

Projet Jeu Vidéo

Par : Khertish **Lobine**
Noreen **Sooltangos**
Ken Addison Chan Yin Shin **THIBAUD**

Destinataire : MR. K. Ramoth

Date: 17/03/2024

Table of Content

Introduction

At the Université des Mascareignes, we tackled the exciting world of game development using JavaFX programming. Guided by Mr. K. Ramoth, we built an engaging game that merges technology and fun. Our project starts with the StartScreen class, which acts as the entry point for players. Here, they find a user-friendly menu that reflects our focus on clarity and ease of use.

The heart of our project lies in the Main Game section. Here, players dive into a captivating gaming experience driven by clever programming. From creating game elements to managing player actions, we've crafted a game that's both visually appealing and easy to play. Through this report, we'll share our journey of creating this game, highlighting our design choices and how they came together to make an enjoyable gaming adventure.

StartScreen

In the development of the JavaFX application, a structured approach was adopted to ensure clarity, maintainability, and functionality. Each class was meticulously designed to serve a specific purpose within the application framework. The StartScreen class, for instance, serves as the initial user interface presented to players upon launching the application. In this class, a vivid start menu is created, featuring a distinctive title label and three interactive buttons: "Start", "Tutorial", and "Quit". These buttons are intelligently implemented to trigger corresponding actions upon being clicked, such as launching the game, accessing the tutorial section, or gracefully exiting the application. Let's delve deeper into the intricacies of the StartScreen class to comprehend its inner workings and significance within the broader context of the application architecture.

Package and Imports:

The code begins with the package declaration (package application;), indicating that the class StartScreen belongs to the application package.
It imports necessary classes from the javafx.application, javafx.geometry, javafx.scene, and javafx.stage packages.

Class Declaration:

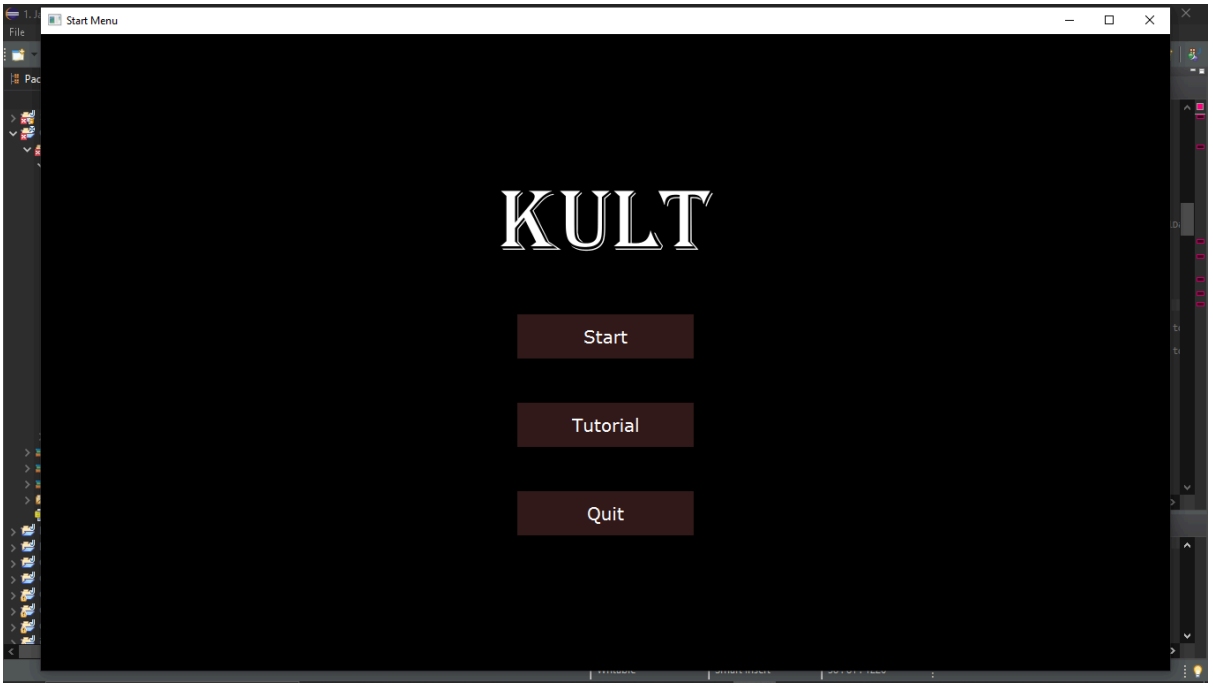
The class StartScreen extends Application, making it a JavaFX application.
Inside this class, the main method is overridden to launch the JavaFX application.

Start Method:

This method is called when the JavaFX application is launched.
It receives a Stage object (primaryStage) as a parameter, which represents the primary stage of the application.
The title of the primary stage is set to "Start Menu".
A Label titled "KULT" is created with a specific font, size, and colour.
Three buttons are created: "Start", "Tutorial", and "Quit", using the createButton method.
These buttons are added to a vertical box (VBox) layout along with the title label.
The background colour of the layout is set to black.
The layout is then added to the scene.
Finally, the scene is set to the primary stage, and the primary stage is displayed.

CreateButton Method:

This method creates a button with the specified text, background colour, and action.
It takes three parameters: the button text, background colour, and the class corresponding to the action to be performed when the button is clicked.
The button's text color, font, size, and background colour are set accordingly.
If the provided class is not null, an action is assigned to the button that launches a new instance of the specified class when clicked. Otherwise, the primary stage is closed.



Main Game

Update Function

The game mainly loops the update function throughout the course of the level.
In the update functions, there are several important parts of the game which gets update after a specific event

1. Keyboard Input Handling:

This part of the code checks for keyboard input using the isPressed method, which likely checks whether a specific key is currently pressed. Depending on the key pressed (SPACE, A, or D), and certain conditions (player.getTranslateY() and player.getTranslateX()), it invokes corresponding methods (jumpPlayer() or movePlayerX()).

2. Player Movement:

It updates the player's position based on velocity (playerVelocity) and handles jumping (jumpPlayer()) and horizontal movement (movePlayerX()). It also limits the movement within the game boundaries.

3. Pickups Interaction:

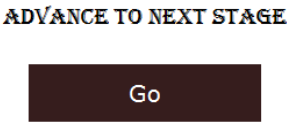
It checks if the player intersects with any pickups. If there's an intersection, it updates the score (point) and removes the pickup from the game.

4. Enemy Interaction:

Similar to pickups, it checks for intersections with enemies. If the player intersects with an enemy, it deducts points from the score and removes the enemy from the game.

5. Goal Interaction:

It checks if the player intersects with the goal position. If so, it updates the game state and allows the user to advance to the next level by calling the Goal() method.



6. Player Health Handling:

If the player's health drops to 0, it triggers the die() method.

Entity Creation

Parameters:
x, y: These are the initial coordinates (position) of the entity.
w, h: These are the width and height of the entity.
color: This specifies the colour of the entity.

Creating the Entity:
It creates a Rectangle object named entity with the specified width and height.
It sets the initial translation (position) of the entity using setTranslateX and setTranslateY methods.
It sets the fill colour of the entity using the setFill method.

Setting Properties:
It sets a property called "alive" to true for the entity. This property could be used later for game logic, such as checking whether the entity is still active or has been removed from the game.

Adding to the Scene Graph:
It adds the created entity to the gameRoot, which is presumably the root node of the game scene graph. This ensures that the entity will be displayed on the screen.

Return Value:
It returns the created entity, which allows the caller to keep a reference to it if needed.

Game Loop

The gamloop works using the initContent method which initialises the game content, including setting up the game background, creating graphical entities such as platforms, pickups, enemies, and goals based on a randomly selected level from LevelData, positioning the player, and adding everything to the game scene.

Here's a breakdown of what the method does:

1. Random Level Selection:

It creates a Random object to generate random numbers.
It generates a random number between 0 and 2 inclusive, which determines the selected level.

2. Text Labels:

It creates a label (hitPoint) to display the player's health points (point) on the UI.
It sets the font, colour, and text for the label.
It adds the label to the UI root.

3. Game Loop for Level Initialization:

It iterates over the lines of the selected level.
For each character in the line, it creates corresponding entities based on the character:
'0': Empty space (no entity).
'1': Platform entity.
'2': Pickup entity.
'3': Enemy entity.

'4': Goal entity.
It adds the created entities to their respective lists (platforms, pickups, game_enemies, goals).

4. Player Positioning:

It calls the player_position method to set up the player's initial position.
Inside player_position, it creates the player entity (player) and positions it at coordinates (80, 600).
It adds a listener to the player's translateXProperty to adjust the game root's layout based on the player's horizontal position.

5. Adding Nodes to the Scene:

It adds the background, game root, and UI root to the application root (appRoot).

Winning and Losing Methods

Die Method

The die() method is responsible for displaying a game over screen when the player dies.

Creating Game Over Screen :
It creates a Rectangle named gameOverMask with dimensions 1280x720 and fills it with black colour. This rectangle serves as a mask to darken the background.
It creates a vertical box (VBox) named endBox, which will hold the game over messages and buttons.
It sets padding and alignment for endBox to position it in the centre of the screen.

Game Over Message:
It creates a Label named endMessage with the text "YOU DIED!".
It sets the text colour, font size, and style for endMessage.

Creating Buttons:
It creates three buttons: "Retry", "Menu", and "Quit".

Each button is created using the createButton method, specifying the button label, background colour, and target class (for navigation purposes).

Adding Components to endBox:
It adds endMessage and the three buttons to endBox.

Displaying Game Over Screen:
It adds gameOverMask and endBox to the UI root (uiRoot). This effectively displays the game over screen on top of the existing game scene.

Goal Method

The goal method triggers when the player reaches the goal in the game.

Creating Goal Screen:
It creates a Rectangle named Goal with dimensions 1280x720 and fills it with black color. This rectangle serves as a background overlay to darken the rest of the game scene.

Creating Goal Box:
It creates a vertical box (VBox) named goalBox, which will hold the message and button related to reaching the goal.
It sets padding and alignment for goalBox to position it in the center of the screen.

Goal Message:
It creates a Label named endMessage with the text "Advance to Next Stage".
It sets the text color, font size, and style for endMessage.

Creating Advance Button:
It creates a button named advanceButton with the label "GO". This button will allow the player to advance to the next stage.
The button is created using the createButton method, specifying the button label, background color, and target class (for navigation purposes).

Adding Components to goalBox:
It adds endMessage and advanceButton to goalBox.

Displaying Goal Screen:

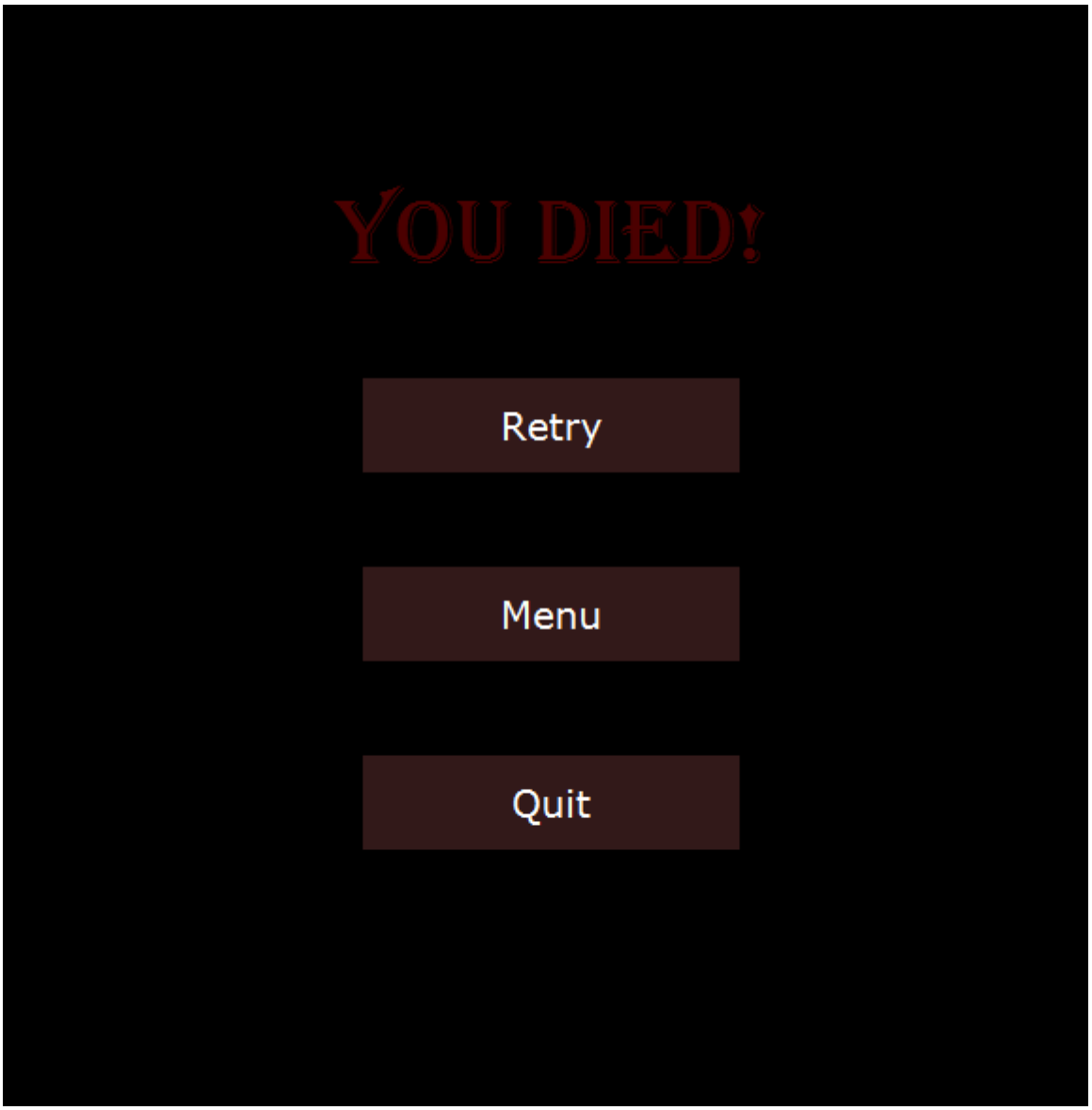
It adds Goal and goalBox to the UI root (uiRoot). This effectively displays the goal screen on top of the existing game scene.

Annex

A.1 - Start of the game

A.2 - collect yellow box to finish

A.3 - Death screen



A.4 - Next stage screen