

SCHOOL OF ENGINEERING AND TECHNOLOGY

FINAL ASSESSMENT FOR THE BSC (HONS) INFORMATION TECHNOLOGY; BSC (HONS) COMPUTER SCIENCE; BACHELOR OF SOFTWARE ENGINEERING (HONS) YEAR 2

ACADEMIC SESSION 2023; SEMESTER 3

PRG2104: OBJECT ORIENTED PROGRAMMING

Project

DEADLINE: Week 14

INSTRUCTIONS TO CANDIDATES

- ☐ This assignment will contribute 50% to your final grade.
- ☐ This is an individual assignment.

IMPORTANT

The University requires students to adhere to submission deadlines for any form of assessment. Penalties are applied in relation to unauthorized late submission of work.

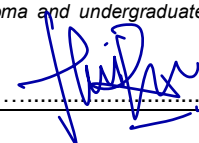
- Coursework submitted after the deadline will be awarded 0 marks
-

Lecturer's Remark (Use additional sheet if required)

I..... (Name)std. ID received the assignment and read the comments..... (Signature/date)

Academic Honesty Acknowledgement

"I Thanh Hai Ry.....(student name). verify that this paper contains entirely my own work. I have not consulted with any outside person or materials other than what was specified (an interviewee, for example) in the assignment or the syllabus requirements. Further, I have not copied or inadvertently copied ideas, sentences, or paragraphs from another student. I realize the penalties (refer to page 16, 5.5, Appendix 2, page 44 of the student handbook diploma and undergraduate programme) for any kind of copying or collaboration on any assignment."



22/08/2023

(Student's signature / Date)

| | |
|---|-----------|
| Introduction | 2 |
| Game Features | 3 |
| Main Menu | 3 |
| GameDisplay | 4 |
| Handle User Input | 4 |
| Timer, Lives and Score | 5 |
| Success/Game Over | 6 |
| UML Diagram | 8 |
| Personal Reflection | 9 |
| How I Applied Object-Oriented Concepts In My Assignment | 9 |
| Classes and Objects | 9 |
| Encapsulation and modularity | 10 |
| Abstraction | 10 |
| Inheritance | 10 |
| Polymorphism | 11 |
| Problems I've Encountered | 12 |
| Formula for jumping | 12 |
| Dynamic Binding | 12 |
| Separation of classes | 13 |
| Strength and Weaknesses | 14 |
| Strength | 14 |
| Declarations of constants makes it easier | 14 |
| Object classes are readily available | 14 |
| Separation of controller classes | 14 |
| Weaknesses | 15 |
| No infinite platforms and not dynamically created | 15 |
| Lack of inheritance and polymorphism | 15 |
| Conclusion | 17 |
| References | 18 |

Introduction

This project is a PRG2104 Object-Oriented Programming coursework, and the GUI system that I have chosen is a Keyboard Typing Game: A GUI game that promotes accurate and fast typing of players through engaging interactions. In this game, the player is in the form of a bunny, and platforms with words will be displayed on the screen. The bunny leaps onto each platform everytime a correct word is typed by the player. To keep players motivated to play this game, I have also incorporated some gamification elements. I have added a time constraint; for each word on each platform, the player will have to type out the words before the timer runs out else the game ends. Additionally, I have also added a live bar that automatically deducts a life everytime the player mistypes. The player would have three lives and the game ends if all three lives run out. A score is also displayed to keep track of the player's progress in the game.

Game Features

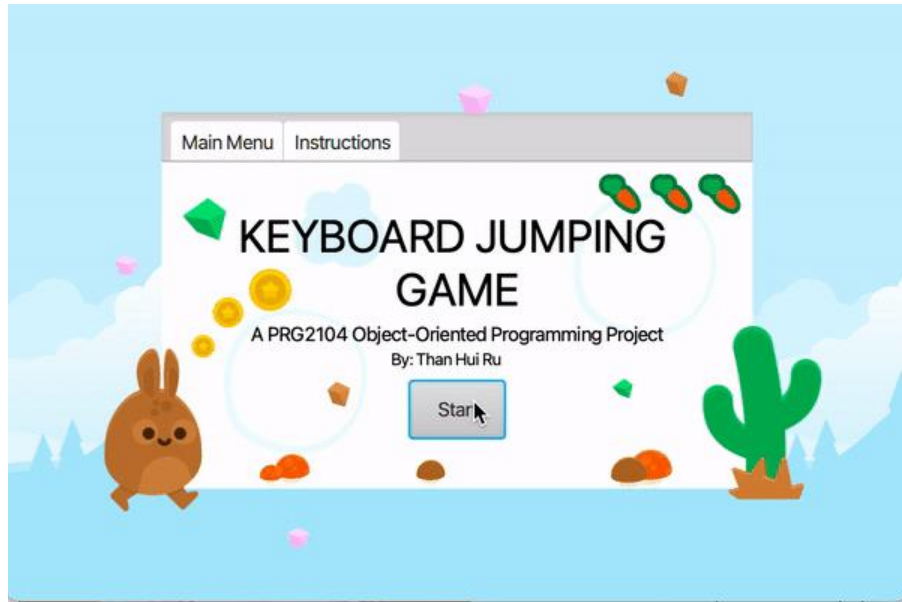
The app has a total of 3 interfaces - (1) Main Menu: entry point of the game, (2) Game Display: where the player plays the game, and (3) Success/Game Over: where the game ends and the player can choose to replay.

Main Menu

The main menu is the starting point of the game, this is the first screen that the user sees when it runs the app. The main menu has a tab pane in the middle and the user can choose to open up the instructions tab to look at the game instructions.



Alternatively, to start the game the player can click 'Start' and they will be brought to the game display interface where they can start playing.

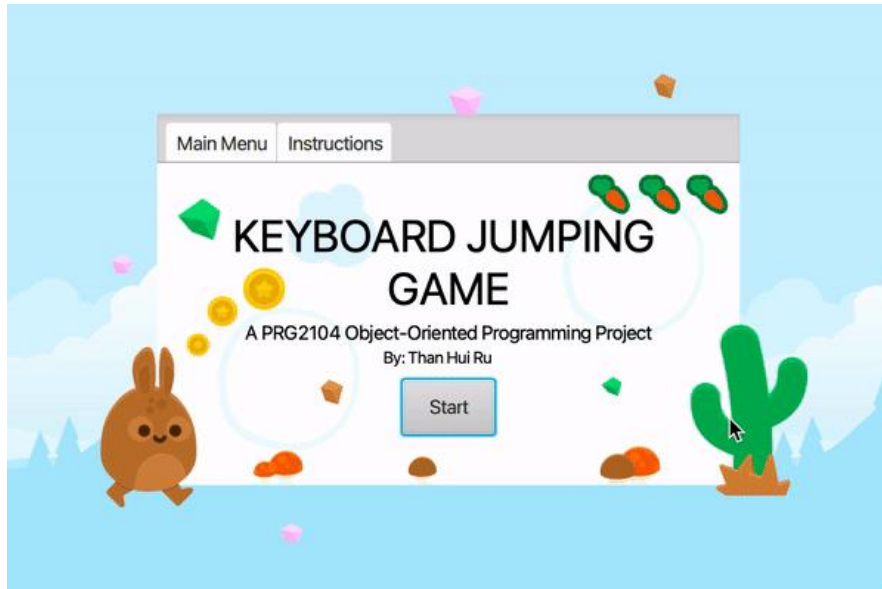


GameDisplay

The game display interface shows all the game components and this is where the player plays the game. For this specific game, I have implemented a few features which i will demonstrate here:

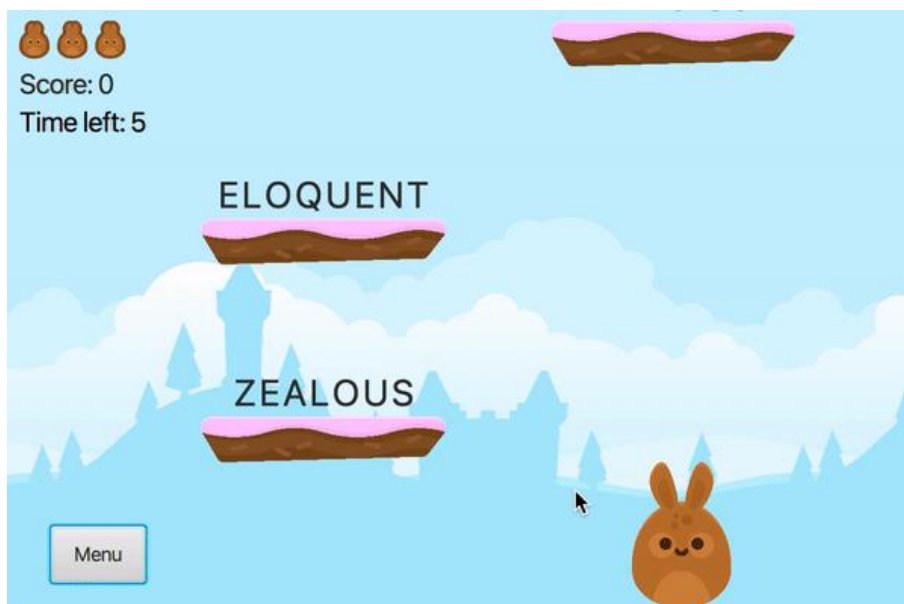
Handle User Input

This game centres around typing the correct words to propel the player forward. The game display is as shown above, words on platforms are displayed on the screen. The player should type in words and the correctness of the words would be reflected on the screen. Everytime a character is typed wrong, that character will change to red, in addition, the screen will also tremble to indicate to the player that he/she has inputted the wrong character. Each character correctly typed will turn green. After a word is correctly typed, the player leaps onto the platform and the words disappear.

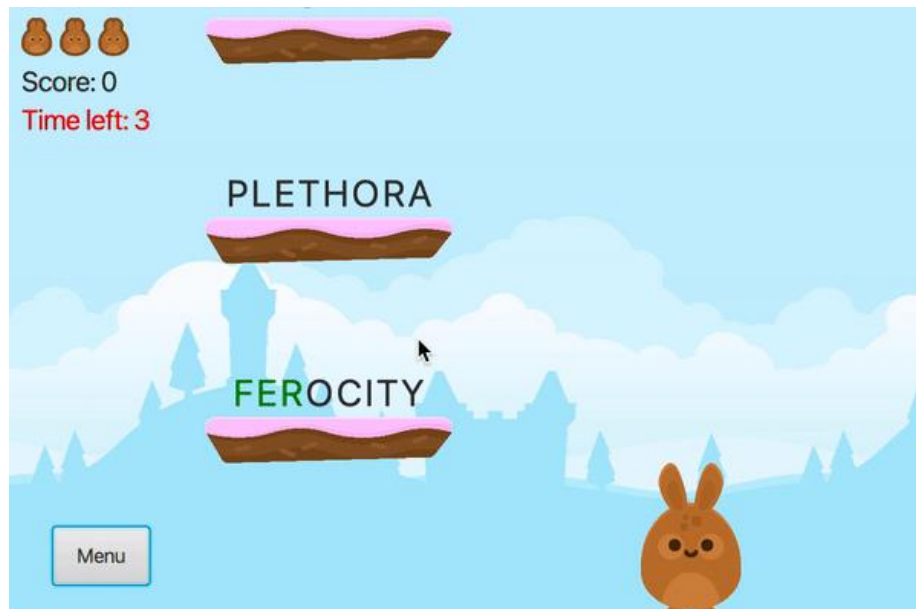


Timer, Lives and Score

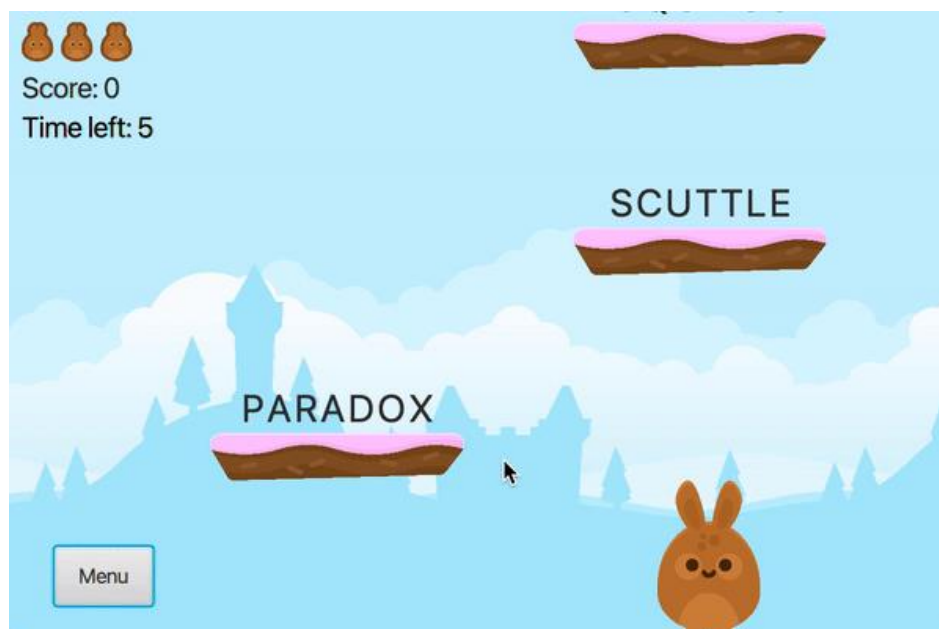
These are the player's metrics that dictate when the game should end or should it continue. The timer counts down the time the player has to complete typing out a complete. If the timer reaches 0 before the user can finish typing out the words then the player loses. The timer will turn red to alert the player when time is less than 3 seconds.



The player starts off with three lives and the lives are deducted every time an incorrect character is typed from the player. If the same character is typed wrong the lives will not be deducted.



Everytime the player completes a word, 10 points are added to the total score, this is to keep track of the progress of the player.



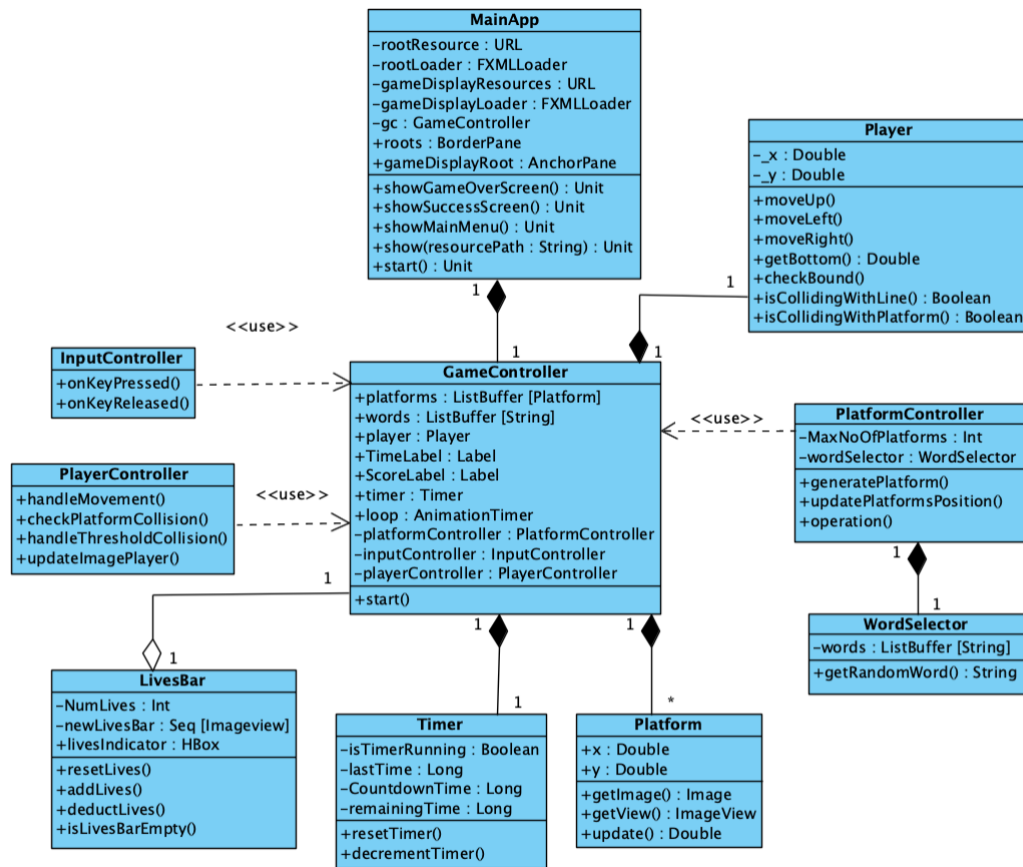
Success/Game Over

This is the screen that shows if the player has succeeded/failed to complete the game. If the player has completed all 20 platforms of the game it will show the success screen else the game over screen is shown. Both perform the same function, it has a replay button that players can click and renavigate them back to the main menu page.



UML Diagram

This section is the class diagram for my GUI system that outlines my main classes and their relationship. These include:



Personal Reflection

How I Applied Object-Oriented Concepts In My Assignment

Classes and Objects

Classes and objects can be clearly identified throughout the project code. The project code is organised into a Model-View-Controller(MVC) architectural pattern thus the classes can be distinguished through three different types of classes which are model classes, controller classes and views which are also the FXML files. Here is an overview of the classes.

| Model | Controller | View |
|--|--|--|
| 1) LivesBar 2) Platform 3) Player 4) Timer 5) WordSelector | 1) GameController 2) GameDisplayController 3) GameOverController 4) InputController 5) MainMenuController 6) PlatformController 7) PlayerController 8) RootLayoutController | 1) GameDisplay.fxml 2) GameOver.fxml 3) MainMenu.fxml 4) RootLayout.fxml 5) Success.fxml |

Model classes are created for individual components, the player that you see on the screen is a player object which is controlled by a PlayerController class. Platforms created on the screen also demonstrate the usage of platform class. Platforms shown on the screen are individual platform objects which are created using the PlatformController class and the WordSelector class for attaching specific words.

Encapsulation and modularity

The GameController acts as the central hub for all the game logic. The GameController encapsulates all different variables and controllers that manage specific functions in the game. The GameController initialises pivotal elements such as LivesBar, Player, Timer and uses controller objects such as platformController(to handle all platform generation and updates), inputController(to handle responses towards user's keyboard input) and playerController(to handle the character's movement on the screen). The GameController demonstrates encapsulation while all the individual controller classes demonstrate modularity by separating functionalities from each other, allowing for modifications without affecting other areas.

Abstraction

All of the classes above provide us an abstraction from all the complexities, we can simply call the GameController's start() method to start the game without having to worry how the underlying mechanics work. This is also true for the controller classes encapsulated within the GameController class, each providing a different tier of abstraction. For example, we can simply invoke generatePlatform() to display platforms with words on the screen. Another example would be handling the character's movement: if we wish to see if the character is colliding with the platform, we just need to call the isCollidingWithPlatforms() function and do not need to worry on the implementations to see how the player element and platform element actually interacts. Such abstractions hide away low-level implementations and provide us with a high-level interface to manage the game's platforms, words, player movement and collisions.

Inheritance

The MainApp extends the JFXApp class provided by the JavaFX library which enables me to create a GUI app much easier through the access of its methods, properties and functionalities. This includes methods for setting up main stages and switching between graphical components. JFXApp was able to provide me with the basic application framework, and I had further coded my GUI app using my own application logic and features. An example of an inherited object is the

AnimationTime in the GameController app where it is used to provide seamless animation when the platforms and player move on the screen. Using inherited classes such as provided by JFXApp is preferable since they are built to run efficiently on hardware resources.

Polymorphism

Polymorphism is involved when you interact with different views (GameOver.fxml, Success.fxml, MainMenu.fxml) using the same show method. This method accepts different resource paths, demonstrating a form of polymorphism.

Problems I've Encountered

Formula for jumping

One of the significant problems that I have faced during this project revolves around the difficulty in constructing the mechanics for the character to jump onto specific platforms. While the internet has a lot of resources for jumping mechanics, there is very little resource on how to automate jumps - specifically how to program my bunny to leap from one platform to another after a correct input. I initially referred to the resources for jumping, and created a bouncing bunny whose movements can be controlled by keyboard inputs - the player can start jumping, move left and move right. Collision detection was added and the bunny was able to detect and collide on the platforms. However this is still far from what i want to achieve, which is having my player jump on the platforms without any player intervention. This is particularly difficult because a simple transition of the player to the next platform would not suffice, I would like to create the animation of the player jumping(with gravity) and landing onto the platform. After considerable experimentation, I managed a formula that triggers the player's movements(moveLeft, moveRight, and moveUp) to reach the platforms as shown below. This code judges the position of the next platform, and triggers the player's movement to jump to the next platform.

```
def handleMovement(): Unit = {  
  if (reachCoordinate == false) {  
    if (m > 0 && platforms(m).x == platforms(m - 1).x) {  
      player.moveUp()  
      gc.isJumping = true  
    }  
    else if (platforms(m).x == 130 && platforms(m).x - 10 < player.x) {  
      player.moveUp()  
      player.moveLeft()  
      gc.isJumping = true  
    }  
    else if (platforms(m).x == 360 && platforms(m).x + platforms(m).width / 2 + 10 > player.x) {  
      player.moveUp()  
      player.moveRight()  
      gc.isJumping = true  
    }  
  }  
  if (player.isCollidingWithPlatform(platforms(m)) && (player.x + 160 > platforms(m).x) && (player.x < platforms(m).x + 160) && (player.y + 90 < platforms(m).y)) {  
    reachCoordinate = true  
    m = m + 1  
    gc.isJumping = false  
  }  
}
```

Dynamic Binding

Another problem that I have faced during this project was dynamically binding the FinalScoreLabel to the GameOver and SuccessScreen. While all of the necessary steps have been made, the FinalScoreLabel variable could not properly bind with the label object on the screen. The problem stemmed from using the same controller class to bind different FXML files, where the FXML files did not share the same label objects. This mismatch led to a null pointer exception since the controller could not locate the label to bind with. I consulted Dr. Chin for guidance on this problem who not only was able to identify the problem but also highlighted a broader concern of my code which is the lack of separation of my MainApp with my game logic which led to the complications of dynamic binding.

Separation of classes

Initially, all of my game logic was concentrated within my MainApp file. This included the declaration of variables, objects and controllers that the game required to function. During the consultation with Dr. Chin, he pointed out that this is rather inaccurate and does not align with the concept of MVC software architecture and the MainApp should ideally only consist of abstract code that runs the application's execution, rather than containing the intricate details of the game's inner workings. I refracted the code and resituated all code pertaining the game logic (starting of the game, and loop of the game logic, etc.) into a separate class called GameController. By separating out the class, the MainApp requires only to create the controller object and invoke its start method to start the game, providing simple abstractions. Through this more modular approach was many benefits, the program now adheres to the MVC architecture by segregation functionalities based on their roles - the MainApp focusing on launching the application, while the gameControl handles the game-specific operations. I was able to fix the dynamic binding problem by only binding the controller to FXML files that share the same variable. My experience with this problem underlines the importance of modularity and separation of concerns. By clearly separating out functionalities into a different controller I was able to get a clear overview of the entire code's execution, which also improves maintainability and the ease of debugging in the future.

Strength and Weaknesses

Strength

Declarations of constants makes it easier

Throughout this project, to ensure the game can be easily altered, constants had been used to ensure uniformity and maintainability of the code base. Constants that I have declared include MaxNumOfPlatforms, CountdownTimer, PlayerHeight, PlayerWidth, PlatformHeight, PlatformWidth and many other values that are fundamental to the game's mechanics, appearance and overall behaviour. Having such declarations improves readability as rather than looking at the integer 20, the coder is indicated that this integer is associated with the number of platforms being generated. Having constants ease maintenance and consistency since updating values is centralised in only changing that specific constant, after update the change would be reflected throughout the codebase.

Object classes are readily available

The creation of classes akin to the creation of constants also enhances maintainability. Model classes include Platform, Player, Timer, WordSelector and LivesBar which allows users to fine-tune game characteristics easily. If the coder wishes to tweak the game characteristics such as increasing the difficulty, they can easily do so by extending these classes to contain more methods. The coder can create different variations of behaviour of these classes with ease by introducing new methods of existing methods. For example, to increase difficulty, users can extend the Platform class to create a disposablePlatform, where the user only has one chance to jump and if a second jump happens the platform breaks. By extending, the DisposablePlatform would have all characteristics of the platform class and coders do not have to worry about how it interacts with the player to create collision detection. Having these classes also means that the change to its private variables is reflected throughout the entire codebase, again improving maintainability. Such a modular approach also allow coders to tailor the game's behaviour to their preferences without extensive code modification.

Separation of controller classes

The separation of controller classes modularised the mechanics of the game into distinct controller classes, with each controller in charge of a specific game mechanic. This allows the code to be more organised:

- 1) GameController: Coordinate and collaborate all different controllers to function the game
- 2) InputController: Handles user input and trigger appropriate actions
- 3) PlayerController: Handles player's movement given a situation
- 4) PlatformController: Handles the generation and updates of platforms attached with words on the screen
- 5) MainMenuController: Controls the Main Menu page on key events
- 6) GameDisplayController: Controls the GameDisplay page on key events
- 7) GameOverController: Controls the GameOver and Success page on key events

The encapsulation of game logic into distinct classes shields the rest of the application from the intricacies of the mechanics. This reduces the complexity of the codebase and each controller also adheres to the Single Responsibility Principle (SRP) which promotes comprehension and simplifies the design debugging and maintenance process. The coder is able to locate and modify specific functionality without having to go through unrelated sections.

Weaknesses

No infinite platforms and not dynamically created

This game features a limited number of platforms the character can jump on, a number that has been set to 20 platforms. These platforms are created statically at the start of the game rather than extend infinitely. As a result, the more platforms you wish to create, the longer it takes for the game to start. Moreover, the moving animation of the platforms is created by updating the all platform's coordinates according to the player's movement. A large amount of platforms would lead to a less seamless animation causing a perceivable lag. The problem of extended start

time and animation issues can be improved for future versions by tweaking how these platforms are generated: rather than generating the platforms all at once at the beginning of the game, it can be generated as the game executes by recycling the platforms - if the platform is out of the screen recycle it back to the top. This is able to provide an virtually infinite number of platforms without exhausting too much computer resources.

Lack of inheritance and polymorphism

Since this game is fairly simple, I was not able to fully demonstrate the use of inheritance and polymorphism in the code. However, the groundwork for their utilisation has been thoughtfully laid out in the existing codebase allowing future leverage of inheritance and polymorphism. In future versions, the addition of other difficulty levels will definitely demonstrate the usage of inheritance. For example, creating a different character with special abilities or platforms with different characteristics. These new variations of game elements would inherit the base class to create new elements and the new elements would still have the behaviour of the base class, making it extensible and easy for further developments. Polymorphism can also be demonstrated in such instances, for example creating a platform that slowly disappears as the user is typing the word. Polymorphism is used to handle how the player interacts with the platform differently in comparison to the base platform case. Such application is possible for classes such as Timer and WordSelector, each object can be inherited to create different variations of game elements to increase or reduce the difficulty. The possibility of using inheritance and polymorphism to showcase diverse gameplay is endless as the foundational class has already been created.

Conclusion

In a nutshell, this project was pivotal in enhancing my knowledge (1) about object-oriented programming and (2) game development. Not only did it only taught me principles and fundamentals of using OOP principles but more importantly it served as a comprehensive tutorial on the whole programming process, from conceptualisation to execution. It highlighted to me the importance of planning and design in outlining each element, class and interaction within the application. It is crucial that this step is done properly to prevent extensive refactoring. I realised that having an organised plan/blueprint that I can follow along avoids a lot of confusion and also makes development much easier - easier to track bugs and knowing where everything is.

Another important lesson I learned is how to properly implement the MVC architectural pattern. This understanding enables me to organise code into distinct classes so that their functionalities remain separated. This helps ensure that the codebase remains clean and modular, allowing effective maintenance.

This project has utilised Scala as the programming language and the ScalaFX GUI library to design an appealing UI. I managed to learn quite a lot on how to use the library and the syntax of the language itself despite a learning curve. The library consists of many functions that simplify the process of designing and implementing the user interface and interaction. Aside from that, I also used SceneBuilder to drag and drop the layouts and with it. This has deepened my understanding about concepts such as dynamic binding, and even more so as I have encountered some mistakes during the process.

This project has been a transformative experience, not only did it brush up my programming skills but it taught me how to coordinate between concepts and tools to develop a functional GUI application.

References

Graphic Source:

- 1) <https://kenney.nl/assets/jumper-pack>

Code

- 1) Dr Chin addressAppG1
- 2) DoodleJump Game; <https://github.com/ahaque/Doodle-Jump/blob/master/Doodle.java>
- 3) Collision Detection: <https://www.youtube.com/watch?v=RMmo3SktDJo&t=6s>
- 4) Animation: https://www.youtube.com/watch?v=sPE8STqy_ns
- 5) Pygame Jumpy Tutorial Playlist: https://www.youtube.com/playlist?list=PLjcN1EyupaQIBSrfP4_9SdpJlcfnSJgzL