Android **Game**

CS2JA16: Java

University of Reading

Jason Jay Dookarun
Student Number ° 26017434
Word Count: 788
Date of Submission: April 19, 2021
Time Spent: 40 Hour(s)

Assignment Evaluation:

This assignment allowed me to experiment and apply my theoretical studies of Java from earlier stages of the module to develop a game. Moreover, by completing this assignment, I was able to further improve my coding skills and styles by focusing on efficiency. Finally, my documentation skills have improved as part of this project.

Table of Contents

Abstract

The following documentation will focus on the creation and development of an Android game using Android Studio 4 and Java, with set features. The said features include enemies and a level system. To view a user's progress within the game, a high score mechanism has been developed so that the score can be viewed. This documentation will illustrate the development of The Flying Dutchman game, a game focused upon a 2D shooter, whereby every shot on target would reward the user via a score.

Introduction

The Flying Dutchman is a game that involves the main client/user controlling a pilot (The Flying Dutchman) shooting birds down. The following game takes on from inspiration of Flappy Birds and previous 2D platformer games available on the application store. The objective of this game is to shoot as many birds as possible. This can be done via the user tapping on the plane to fire bullets, visible via tracer bullets visibly shown to the user. A score is then recorded and illustrated in live gameplay and later saved for "future reference". As part of this game, the birds are to be considered the "enemies" as they represent obstacles that the user must overcome. The speed at which enemies approach the user is varied throughout the game.

Figure 1: Homescreen

Figure 2: Level I Gameplay (SkyHighNine)

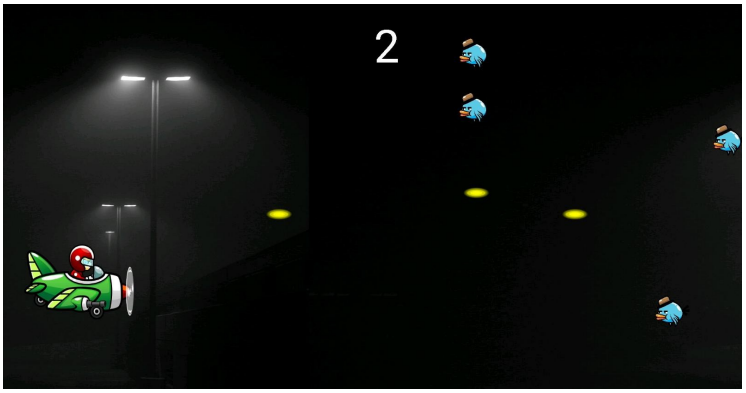

Figure 3: Level II Gameplay (GhostTown)

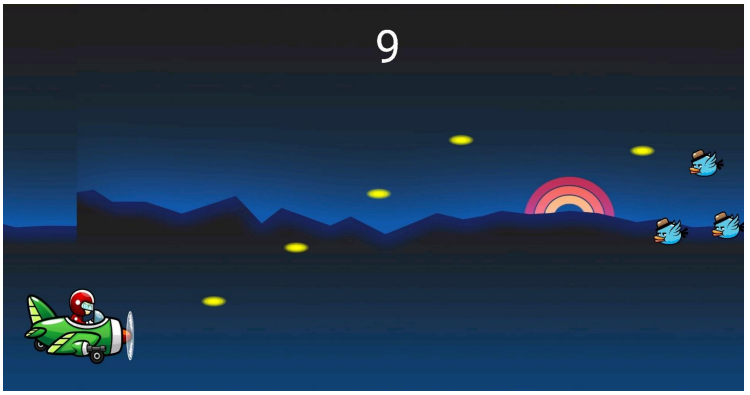

Figure 4: Level III Gameplay (Retro ‘84)

OOP Design

To comprehend the flow and interactions involved within the gameplay, it was essential to apply an object-oriented approach, via designs and principles. To apply this, a class diagram was developed to comprehend all the components that were to be attached to the game. The following OOP Class Diagram represents the basic flow involved within the game, with considerations to the set requirements provided in the brief.

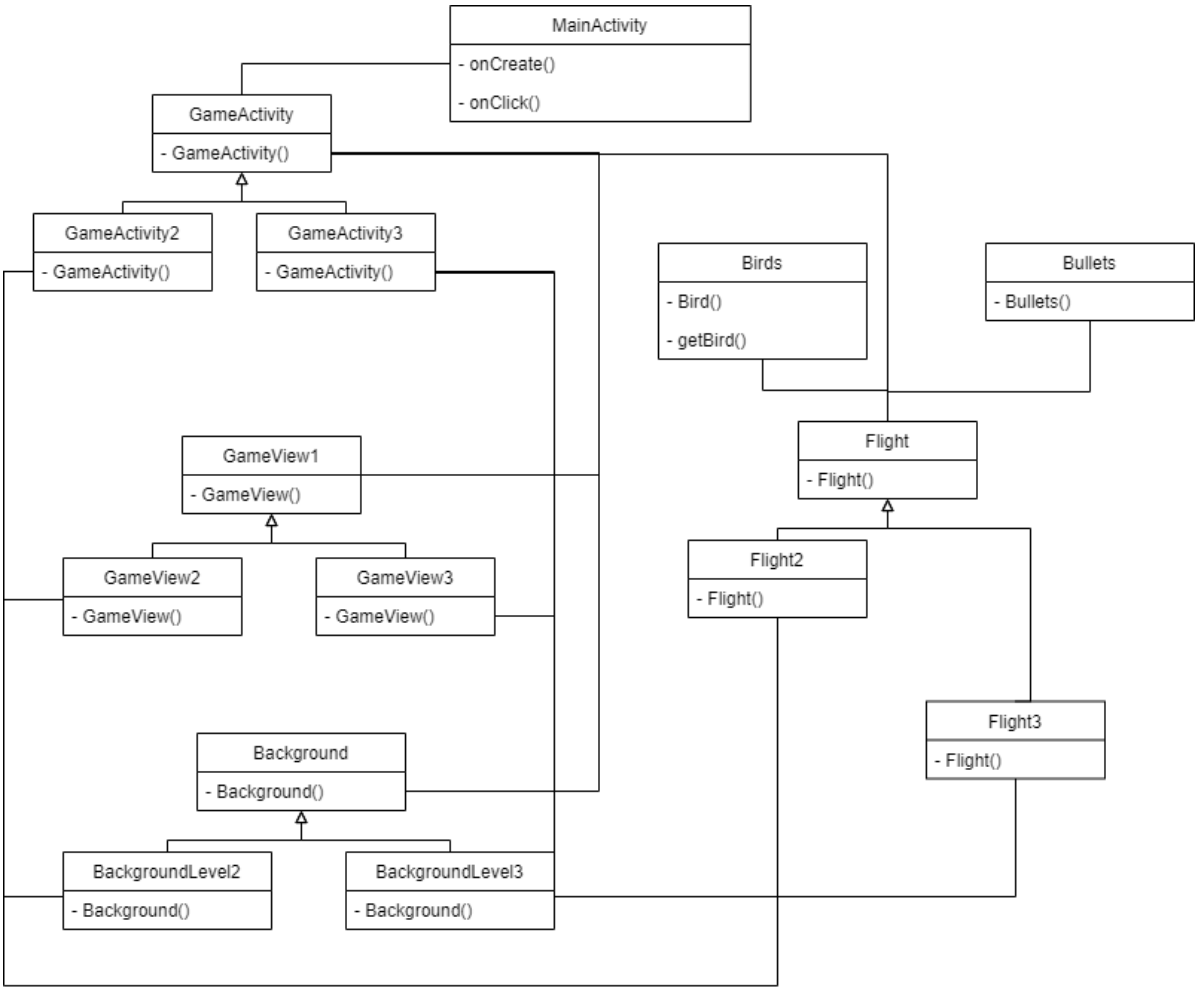

Figure 6: Class Diagram

As per the above illustration, several classes are referenced to one another via inheritance to refer to certain elements such as the use of sprites. This includes classes such as all GameViews that supported the development of the game, as each view represented a different level to the game.

Memory Usage and Speed Improvements

To conserve the amount of memory through the process of the game, it was important to comprehend methods to halt extreme usages of memory. Such as can be achieved via the removal of depreciated content, redundancy ensuring that efficiency is maintained. **[1]**

To trace and review the amount of memory used, a pre-built Android Profiler was used to trace the progression of memory usage. Following the removal of redundant data that was not necessary to the functionality of the project, and the removal of depreciated content, memory usage significantly improved.

Additionally, to further enhance the speed progression of the application, certain principles were adopted including compression of files, removal of unused resources, adding necessary dependencies, using vector drawables where applicable and the removal of unused code.**[2]** By doing the mentioned, the application load speed drastically improved through the early stages of testing and following final compilation and exportation.

Improvements/Extensions

After being provided with a base framework from the University of Reading via the FutureLearn platform, I reviewed all elements that could've been further added in. An example of additions within this project includes an improved menu panel whereby the user can select a level, a help button available on the main home screen for user support before commencing the game. Likewise, to further improve the base framework, additional levels were added as per the set requirements with a scoreboard active. It can be noted that the scoreboard is a locally based system and not online, yet functional.

Conclusion

To conclude, this project allowed me to further develop my Java coding knowledge with a way to replicate my interests in Java coding through this application. By merging and applying my theoretical knowledge of Java into this project, I was able to further tailor my concepts into a live visual.

Despite applying all my knowledge as part of this project as well as my previous coding and object-oriented programming knowledge, I believe I could've further developed the project to produce a greater quality product. These extensions would include a live database leaderboard that would record the user's name and score, additional enemies and variations and a level-based system whereby if a score was to be achieved, another level would be unlocked. If given a further time extension, and if I applied a better time management schedule on this project, I believe I would've been able to further improve the project.

References

All coding elements involved in the development of the Flying Dutchman game can be found within the said repository. The contents found within the repository include coding components, assets and documentation made and developed by Jason Jay Dookarun.

GitLab Repository: https://csgitlab.reading.ac.uk/qt017434/cs2ja-game

[1] Android Developers. 2021. *Manage your app's memory | Android Developers*. [online] Available at: <https://developer.android.com/topic/performance/memory> [Accessed 4 March 2021].

[2] Medium. 2021. *Build your Android app Faster and Smaller than ever*. [online] Available at: <https://medium.com/linedevth/build-your-android-app-faster-and-smaller-than-ever-25f53fdd3cdc> [Accessed 14 March 2021].

[3] Visual-paradigm.com. 2021. *What is Class Diagram?*. [online] Available at: <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-class-diagram/> [Accessed 9 February 2021].

Marking Scheme

Report and Demo Mark Percentage

Section	Percentage of Grade
Report	**30%**
Demonstration See demonstration self assessment for breakdown	**70%**

Page No.	Section	Mark
#1	**Title** **Abstract** Max 50 words overview of the entire project covering important features/results	**10**
#1	**Section 1 – Introduction and Showcase** A short informal presentation of the application (screenshots with some description).	**20**
#2	**Section 2 – OOP design** A short (a few sentences) description of the design document of the app (marks are also given for describing unimplemented and/or unfinished features). Include: - OOP design with a critical description of how the design aided you in the implementation. - A statement on finished and unfinished classes	**20**
#3	**Section 3 – Memory usage and Speed improvements** A short (a few sentences) descriptions of your research and improvements of memory and speed efficiency of your game.	**20**
#3	**Section 4 – Improvements/extensions** A short (a few sentences) descriptions If you researched and implemented an improvement above the other specifications then add a description of the work.	**20**
#4	**Section 5 – Conclusions** A max 50 words summary of the important results and conclusions from throughout the rest of the document. **References** Link to repository Please make your repository private shared with both Mohammed and Varun. If you are in doubt, create a test repository and ask us during the Weekly session if we can access your repository.	**10**

Appendix A: Demonstration Sheet

Each tick-box represents one mark unless otherwise specified.		Range
Code style (2 marks each points): *Following Java code conventions for all the project* [✓] variable and class names [✓] method names [✓] indentation rules [✓] using inline comments frequently [✓] JavaDoc comments for every function	**Overall OOP design and API usage**: [✓] Appropriate use of inheritance (2 marks) [✓] Appropriate use of Abstract classes (2 marks) [✓] Use of Android API classes/methods (Other than in base code) (3 marks) [✓] Greater than 3 original classes (3 marks)	0-20
Functionality: [✓] On-screen menus (1 mark) [✓] Scores (1 mark) [✓] Controllable character (1 mark) [✓] Sensor interaction (touch/ accelerometer/ etc) (1 mark) **Game has levels** [ ] Works (4 marks) [✓] Attempted (2 marks) **Use of standardised data structures (e.g., List, Vector, Collection, Date):** [✓] Works (10 marks) [ ] Attempted (5 marks) **Randomly/procedurally generated features:** [✓] Works (10 marks) [ ] Attempted (5 marks) **Opponents (AI):** [✓] Works (5 marks) [ ] Attempted (3 marks)	**Design quality** (both game and other game screens) (1 mark each): [✓] Professional looking [✓] Understandable game flow [✓] Feedback to user instead of crashing, or recover [✓] Runs smoothly without interruptions [✓] Installs without error [✓] Starts/exits without error [✓] Program does not crash [✓] Produced in video form	0-40
Improvements marks: **Online high score list** [ ] Works (10 marks) [✓] Attempted (5 marks) **Multithreading improvements** [ ] Works (10 marks) [✓] Attempted (5 marks) **Memory optimisation** [ ] Works (5 marks) [✓] Attempted (3 marks) **User Level Creation** [ ] Works (5 marks) [✓] Attempted (3 marks)	**Other extension/improvement / innovation** [ ] Works (10 marks) [✓] Attempted (5 marks) Note: *either the attempted marks or the works marks will be given (so you can only receive 5 marks per attempt max, and there is a max of 10 marks for this section but tick off any extra work done anyway!) For attempted marks to be awarded the feature must be at such a state, that the code could have worked but does not due to bugs or similar.*	0-40

Demo Self-Assessment

[cite examples where to find evidence for it in your code]

· **Appropriate use of inheritance** - *(Example of Class in your code that inherits another class*

Examples of inheritance include classes:

- GameView, GameView2, GameView3
- GameActivity, GameActivity2, GameActivity3
- Flight, Flight2, Flight3

· **Appropriate use of Abstract classes** - *(Example of Class in your code that is an abstract class)*

Examples of abstract classes include:

- Background, BackgroundLevel2, BackgroundLevel3, Bullet, Enemy

· **Use of Android API classes/methods** - *(Example (one or two would fine) of some API classes/methods)*

- AppCompactActivity Android
- SurfaceView
- Runnable

· **Use of standardised data structures** *(give some example belonging to one of these e.g., List, Vector, Collection, Date)*

- ArrayList: enemy in GameView

· **Randomly/procedurally generated features** *(e.g., randomly adding some obstacles, some sceneries, players, etc)*

- Main character sprite change to represent a plane movement
- Firing of bullets via tracers and visual effects

· **Example of AI Opponents** *(automatically acting objects, like automatically changing position, chasing other objects, opponent moving with the movement of player, etc.)*

- Enemy generation

· **Example of Online Scoring**

An example of the scoring system can be found both on the homescreen and during live gameplay where the score is saved and illustrated in the top right hand corner.

· **Multithreading improvements**

Multithreading can be found in all GameView classes as a sleep functionality is used.