Java Individual Coursework

CS2JA16: Designing a Android Game using Android Studio

University of Reading

Shavin Croos

Module Code:	CS2JA16
Assignment report Title:	Android Game
Student Number:	27015244
Date (when the work completed):	20/04/2021
Actual hrs spent for the assignment:	50+ Hours

Assignment evaluation (3 key points):

1. Grasped the basics of Bitmap functionality and storage.
2. Grasped the understanding of moving objects.
3. Building and improving upon a somewhat playable game.

Space Invaders

Abstract

Space Invaders is a scrolling platformer game constructed with the game framework I had created. The player controls a spaceship hero through a scrolling world, defeating enemies, avoiding obstacles and collecting gems along the way for as long as they can.

1. Introduction and Showcase

Upon launching the application the game shows a very basic splash screen to the user for a few seconds, as shown in **Figure 1**. It contains a made up company name “TheBanditProductions” that created the game, along with an image of the Gem that was used later in the game. After this, the game details load, along with a start screen as demonstrated in **Figure 2**. The start screen contains the instructions that tells the user how to play the game before they enter. When the user taps on the screen, the level is run and the game plays out as shown in **Figure 3**. During the game, if the user hits the enemy or obstacle, the Game Over (Final Score) screen is shown (**Figure 4**), where the number of gems collected and the final score are told to the user. The user has the option to also play the game again to better their previous score.

To add to this, the enemies and obstacles in the game come towards the player at random speeds, which means that the enemies are not exactly the same. This means that some may move faster than others towards the player, which adds a bit of randomness and challenge to the game.

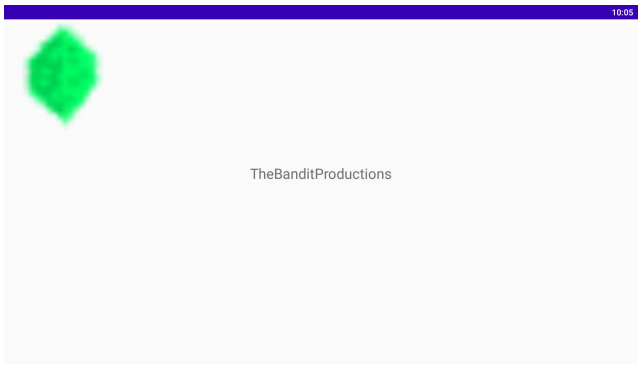

***Figure 1**: Splash page shown upon launch.*

***Figure 2**: Start screen.*

***Figure 3**: Gameplay of the game.*

***Figure 4**: Game Over screen.*

OOP design

By utilising “Code Iris”, a plugin used in Android Studio to create UML diagrams, I was able to generate a basic UML diagram that shows the classes that I have contained within the “com.shavin.spaceinvaders” package, as provided in **Figure 5**. This package is stored within my project.

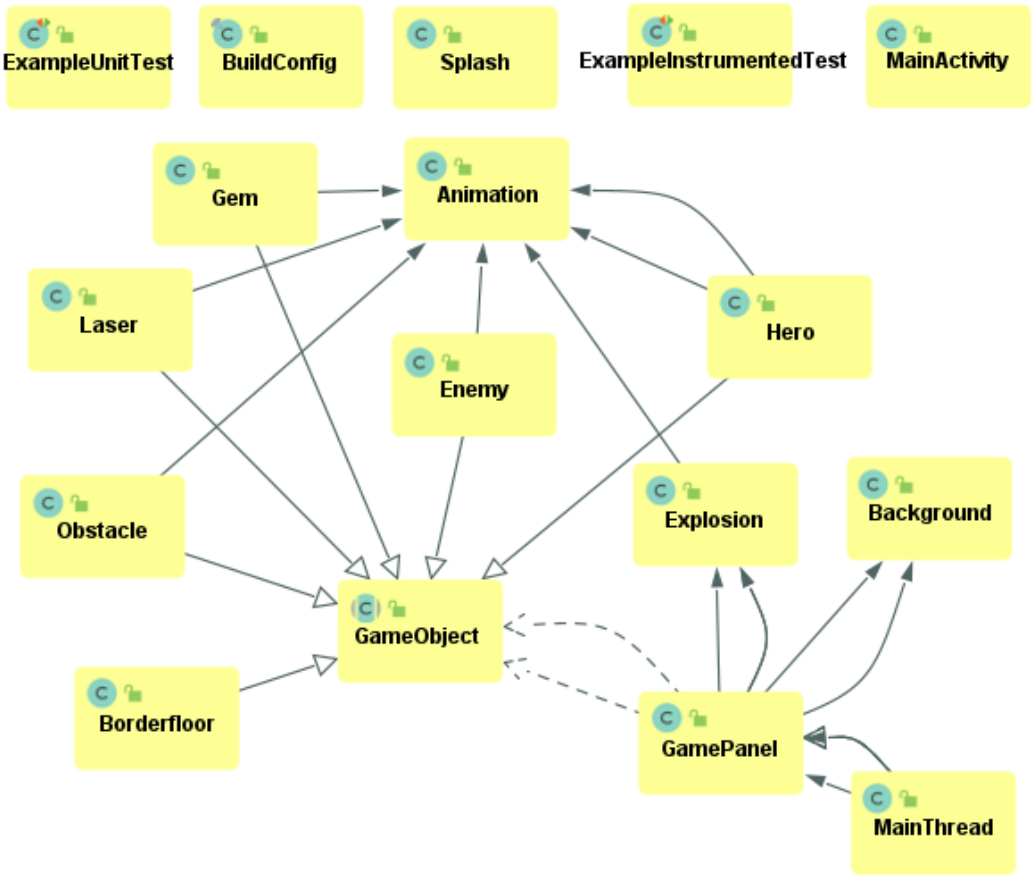

Figure 5: OOP Design Diagram

By observing each part of the diagram, the objects that are required for the game to work are all handled by one package, “com.shavin.spaceinvaders”. Such objects include the Hero, the enemies and border tiles that are used in the game. Each object has specific attributes that detail the action(s) that each object will do in the game. For instance. Gem contains a tag named ‘collectable’ to imply that the item is collectable to the player.

The majority of the classes that are used in the project are inherit attributes from the GameObject class, since this class contains the necessary private variables needed for the objects to run in the game correctly, such as the height, the width, speed, the position of these objects, the starting position of the player and which direction objects face in the game. Most of these objects also call upon the Animation class, which contains the key variables needed to control the speed of the game in terms of frames per second (FPS). This means that the smoothness of how the objects move in the game are determined by this class. However, if this game was to run the way it was without another key class, the game would lag severely, which would disrupt the game experience of the user. This is why the MainThread class is there, as it creates multiple threads to handle several processes at the same time whilst the game is running, allowing for a smooth experience for the user when they play the game. This means that the FPS is also impacted by this class as well.

2. Memory usage and Speed improvements

In my project, in order to make the game run more smoothly on many devices, I created a thread class called MainThread. This class will be integrated each time an object gets updated or drawn on the screen for each and every frame. This will allow for the game to flow more smoothly than with the class, decreasing the CPU and memory usage on the device used to play the game.

The draw methods in the GamePanel class will not necessarily draw everything at once. Only when they are called will they be drawn onto the screen. This further reduces the amount of memory used by the device and results in a smoother gameplay.

Utilising Android Studio's inbuilt profiler, the software will demonstrate how much memory is used currently and the CPU load. I will use my emulator which is the Galaxy Nexus 5 with 1GB of memory. **Figure 6** shows the CPU and RAM usage whilst playing the game and as you can see, the game uses 22.5MB of memory.

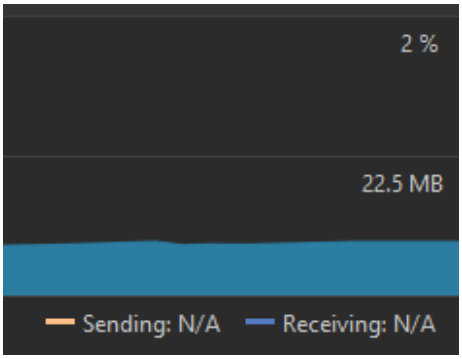

Figure 6: Game with Speed Improvements

3. Improvements/extensions

In my application, I have added a final score system that appears after the player crashes in the game. The scores of the game will be stored until the user quits the application and tries to launch it again. At this point, all the scores will be reset to 0. **Figure 7** shows the final score screen.

Figure 7: Final Score screen

I wanted to add levels to the game to make the game more interesting to play. I was, however, unsuccessful in that task in implementing this feature into the game. This was partly due to the way the code was written. To further improve, I would have liked to add a level builder feature to the game in which the user could build their levels with whatever features they would like to add.

4. Conclusions

Link to GitLab

https://csgitlab.reading.ac.uk/at015244/android-studio-coursework-space-invaders.git

Demo self-assessment 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 frequency [✓] Javadoc comments for every function (except getters/setters)	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 - Enemy.java, Gem.java, Hero.java, etc... (in ‘com.shavin.spaceinvaders’ Package)

- Appropriate use of Abstract classes - GameObject.java (in ‘com.shavin.spaceinvaders’ Package)

- Use of Android API classes/methods - GamePanel.java (in ‘com.shavin.spaceinvaders’ Package), setting MotionEvent and SurfaceHolder API classes

- Use of standardised data structures - GamePanel.java (in ‘com.shavin.spaceinvaders’ Package), setting Arraylist data structures for the Gem, Laser, Enemy, Obstacle and Borderfloor classes

- Randomly/procedurally generated features - GamePanel.java in ‘com.shavin.spaceinvaders’ Package), by calling both the asset classes (e.g. Gem.java, Obstacle.java, etc…) and the Random class together (e.g. private Random rand)

- Example of AI Opponents - in ‘com.shavin.spaceinvaders’ Package, which is the Enemy.java class

- Example of Online Scoring - GamePanel.java (in ‘com.shavin.spaceinvaders’ Package) for displaying the score and storing the score once the player crashes.

- Multithreading improvements - MainThread.java (in ‘com.shavin.spaceinvaders’ Package)