3/9/2018

Mojtaba Hafezi

N0771021 – SpaceShooter2d

Report

Mobile Interactive Systems

Table of Contents

[Introduction and Background 2](#_Toc508330606)

[Design: 3](#_Toc508330607)

[Implementation: 6](#_Toc508330608)

[Testing: 8](#_Toc508330609)

[Critique: 10](#_Toc508330610)

[References 11](#_Toc508330611)

[Demo Sheet: 12](#_Toc508330612)

# Introduction and Background

Space Shooter 2D is a simple game implemented using the IDE Android Studio and the programming language Java. Nowadays the popularity of games in the google play store is undeniable as the top paid applications consist mainly of games (Google LLC, 2018). Thus, this coursework aims to create a video game for android from scratch. The idea is to build a simple two-dimensional game like space invader with a simple menu to navigate through the different screens. There is no win situation in the game, the player will lose the game when his shields are reduced to zero and the scores will be saved. The player has unlimited “bullets” to shoot at the enemies and there are unlimited enemies being spawned, but only six at a time are going to be visible to keep the balance of the game.

Native android game programming is uncommon compared to the creation of applications using popular game engines. Nonetheless, the android features are perfect for any developer to start learning game and android application development. Google provides many benefits like API’s, virtual machines for testing, graphic libraries and media support (Zechner, 2016). Another key factor is that, mobile gaming has become popular due to the mobile devices being everywhere and their prices being so cheap (Zechner, 2016). This seems to be the perfect situation to learn how to write game object classes, detect and handle collision, draw graphics and to make a complete game respond to the player’s input (Cho, 2014).

Additionally, another important aspect this coursework aims to cover is to make use of android specific features and learn more about the native development process. Key features as Bluetooth controller handling and the use of accelerometer can be adopted for any other application as well. The simple game development framework created at the end of this work needs be altered and can then be reused as well. As Cho (2014, pp. 96-107) states, reusability is an important matter in software development, but it was neglected for this game framework to make the work feasible.

Android is provided on many different devices and hence many different screen resolutions need to be covered (Steele & To, 2010). If not handled, this issue will lead to different gameplay experiences on different devices. Horton (2015, pp. 89-97) explains that this issue can be solved by drawing the game objects at the same coordinates and scale appropriately regardless to the resolution. This solution would have gone beyond the scope of a simple game framework and was therefore omitted.

# Design:

The application aims to create a fully working simple two-dimensional game for android. The essential features of a simple game would consist of audio and graphics. To produce a better software, a game engine can be added which includes a better code structure and reusability.

The key features of the application are:

* Graphics: Using bitmaps the graphics are drawn on the screen and animated.
* Audio: To produce sound effects and the looping background music.
* Simple game engine: A game engine with reusable classes for handling the game loop.

The following are optional features:

* Persistence:
  + Shared preferences: Save the Boolean values for options like enabling sound or accelerometer.
  + Database: Connection to and population of a database is included for the high-scores.
* Sensor Input: The accelerometer can be enabled in the options for gameplay. Additionally, vibration is used to represent the damage when being hit by an enemy ship.
* Wireless Connectivity: Gamepads and joysticks to play the game
* Handling audio and video: In the options menu a tutorial button is available which opens a video player that shows the tutorial for the game.

Special code/technical features:

The application will need a single Audio manager instance to take care of specific events. For this matter, the singleton design pattern seems to be a good approach and the disadvantages of it are acceptable for this small project. The Audio manager will have one instance across all activities and handle the sound output and the ambient music. As Horton (2015, pp. 423-427) mentions, for creating an instance of a class that is available throughout the whole application, the singleton design pattern is useful and straightforward.

For drawing the graphics onto the screen there are some viable options. For simplicity, the standard “View” can be used to draw bitmaps on the canvas. In case more animation and graphics are going to be used further views are available. In this project the “SurfaceView” is going to be implemented, since the drawing can be affected by the desired number of objects. Although the “SurfaceView” is more complicated, the draw method will be done in a dedicated thread and therefor leads to better gameplay experience for the users (Derek, 2013).

Most of the game engine relevant functions are in the “GameView” class. This is bad programming style but had to be done, since the focus of this project was android application development. All game objects need to inherit from the “GameObject” class with all the essential attributes and methods included. The collisions between objects will be calculated by creating a rectangle around the game graphic and then checking for intersection between the rectangles. This kind of collision detection is a very expensive process for the device but can be used to simplify the application as there are not more than 6 enemy ships on screen at the same time (Horton, 2015).

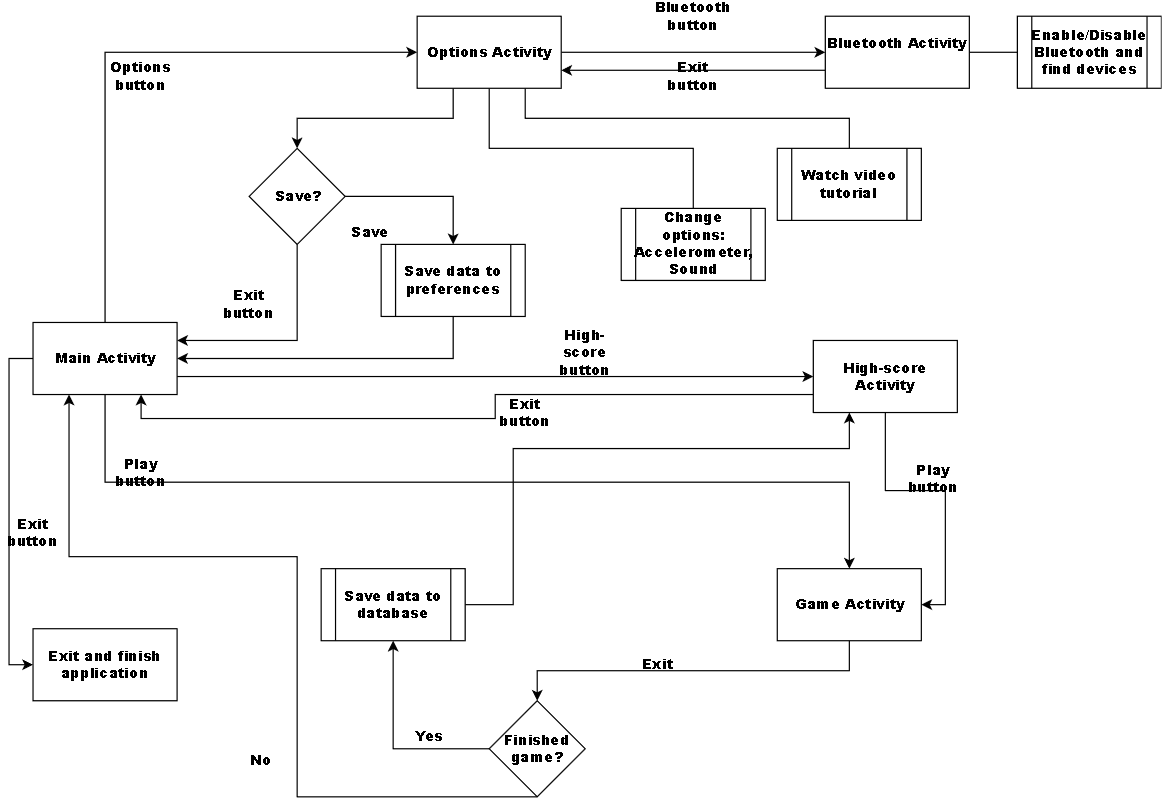


Figure 1: Chart representing the flow

Testing

The testing will be done on different android devices per user testing. Unit, integration and functional tests are recommended for any software but due to the size of this project that kind of tests were left out. The layout and the content scaling were tested on different screen resolutions using the emulators provided by the android studio IDE.

Essential test cases:

* Menu testing: The transition from one activity to another should work. The buttons should change their colour and if the sound is enabled the menu sound should be played before starting the next activity.
* Gameplay testing: A balanced gameplay. The user should not die immediately, and the instance needs to be instantiated without issues and graphic errors. Sound and music should be playing correspondingly. The player’s ship must move according to the input.
* Input: The game needs to be able to accept any Bluetooth controller, joystick or D-Pad. The touch and accelerometer motions have to work accordingly to the input.
* Bluetooth: The Bluetooth enabling/disabling and search for devices need to be tested.
* Tutorial video: The imbedded video player is required to open and play the desired tutorial video. It should be closed upon user request and the application needs to continue.
* Persistence tests: Need to check if the shared preferences and the database work correctly in different situations.
* For detailed tests see the testing section.

# Implementation:

The application consists of different activities. Each activity implements the “onPause” and “onStart” android methods to make sure that the application behaves properly. Furthermore, all buttons in the activities manually implement the listeners for clicking events and change of focus to provide the desired result. Hence, each button’s colour changes when focused on through a game-pad. Moreover, all activities make use of the sound manager’s singleton pattern to play or pause the sound effects including the ambient music. Other common factors through them include the use of the same orientation and themes in the android manifest.

Main activity:

The main activity has four buttons and provides the user access to the other features. The title is an image view object. The Layout consists of a linear layout inside a relative layout and with all the elements as child objects. Each element has a specific layout weight and hence scales accordingly with the screen device.

Option activity:

In this activity the focus lies on the shared preferences and the implementation of the video player. The class contains two private boolean attributes to represent the use of sound and accelerometer. These are loaded at the beginning and saved accordingly upon changes.

By loading the mp4 file into the video view object and setting it as the activity’s content, one can implement an easy video player. Additionally, if the device has the appropriate android version then the controls are added as well. This presented a problem until I figured out that the controls are only available from the SKD version “Lollipop” and above. Upon cancelation the option activity is set to its default state.

Bluetooth activity:

This activity was created with the aim to enable Bluetooth, search for pairable devices and pair them. Unfortunately, I was not able to create a connection between the devices and the application and therefor the part of connectivity was omitted from this project. The Bluetooth can be enabled and disabled by sending the corresponding intent and registering a broadcast receiver. The broadcast receiver catches the different states the Bluetooth adapter is in and depending on the state different code can be executed.

Game activity:

The game activity has the class “GameView” as the content which inherits the “SurfaceView” class.

This is the reason why some of the input events like generic motion and key events for the controller need to be registered in the activity instead of the view since “SurfaceView” would require additional complexity to provide the features. After receiving the input events these are passed down its the view. The “GameView” class has a main loop that manages all game objects, their drawing to the canvas and the frames per second. All kind of input is being handled by the “InputController” class which has dedicated areas of the screen for touch input and matching code execution upon use of game controllers.

High-score activity:

In genal, this class has the connectivity with the database and upon creation will read the data from the database, create text views and add them to the linear layouts in its “ScrollView” object. Using this approach enables the creation of datasets on the fly.

Moreover, this activity receives extra parameters when called from the game activity for the scores and will insert them into the database and store the returned long for the id. Afterwards the database is queried for the scores in sorted order and retrieves the position of the latest insert which stands for the ranking and is displayed to the user through a long toast.

Database:

A simple SQLite database has been implemented. The “MyDBHelper” class inherits from “SQLiteOpenHelper” and creates the database table with the hard-coded SQL “CREATE\_TABLE” statement. Besides the create statement only a database name is required to create the database using the android library. Inside the “GameDataBase” class the read and write functions are defined. Finally, the “Constants” class contains all static strings for the database e.g. the name.

Utilities:

The “InputController” class manages all input for the application. For simplicity reasons it has high cohesion with the “GameView” class which was inevitable. The “Pref” class contains the strings for the shared preferences.

Game objects:

Each game object inherits from the “GameObject” class. It is obvious, that in a game many objects will have similar behaviour. That is why this class was created with intent for sprite, collision and movement support. The bitmap is prepared upon instantiation and the collision detection can be enabled by updating the rectangle object’s position.

# Testing:

All the following tests were done with and without the sound. The results were the same.

|  |  |  |
| --- | --- | --- |
| #Description | Expected Result | Status |
| 1. Main to options | Opening of Options activity | Activity opened without issues |
| 2. Main to highScore | Opening of the HighScore activity | Activity opened without issues |
| 3. Main to game | Opening of the game | Activity opened without issues |
| 4. Exit from main | Exiting the application from main activity | Application closed with no processes in the background |
| 5. Pause on main activity | Pausing the application | The application was paused and put into background |
| 6. Options to bluetooth | Transition from options to bluetooth activity | Successful transition |
| 7. Bluetooth exit | Return to main activity | Successful transition back to main |
| 8. HighScore to main | Return to main activity | Successful transition back to main |
| 9. HighScore to Game | Transition and start of game activity | Successful transition and start of the game |
| 10. Test of buttons | All buttons except for the ones in the Bluetooth activity are supposed to make a sound and change background upon click | Successful for all buttons |
| 11. Test for repetitive activities by going through the activities in cycles | No multiple activities (Correct order of “finish” and “startActivity”) | No multiple activities were found. |
| 12. Enabling/disabling of option buttons | Corresponding behaviour | Accelerometer works only when enabled and in GameActivity. Sound only works over all activities when enabled. |
| 13. Option: Tutorial button pressed | Videoplayer should open and play the video | Video started playing |
| 14. Option: While video is playing pressing back button | Videoplayer should stop and the activity be shown as before | Video was closed and default state was visible |
| 15. Bluetooth: Enabling | Enabling of bluetooth | Permission request and on “yes” the bluetooth was enables |
| 16. Bluetooth: Disabling | Disabled bluetooth | Bluetooth was disabled immediately |
| 17. Bluetooth: Search for devices | Search and find devices. New Items to be shown on the list | Shows a list of already paired devices. Searches for further ones and if available adds them to the end of the ListView |
| 18. Game: Start with correct stats | Game should start every time with the correct default attributes | On at least 20 playthroughs all attributes started with the same value |
| 19. Game: No graphic errors | Game should not have graphic errors | No graphic errors. But: Due to too many draw calls some graphics were left out occasionally (explosion of enemy ships) |
| 20. Game: sound and music keep playing | Consistent sound and music throughout gameplay | Consistent music and sound throughout 20+ gameplays |
| 21. Game: Different Input devices, same movement | Same movement of the ship independent on input | Same movement of the ship were tested using the accelerometer, game controllers and touch |
| 22: Database statements and queries | Database needs to retrieve correct information and do the insert statements as planned | Correct behaviour. Different tests with and without data from game activity |
| 23. Vibration should work in the game activity | Vibration when player ship is hit or dead | Over 20+ gameplays lead to the same: vibrations on hit and when player ship was destroyed |
| 24. Pause of application on any activity | The application should go to the background and continue afterwards without issue | The application paused accordingly on every activity. Once the user returned the application continued where it was left off. |
| 25. Pause game using game controller | When “START” is pressed then the game is meant to pause and time to be halted | Upon pressing the button the game is stopped and time does continue where it was paused after pressing the buttong again |

# Critique:

Firstly, the creation of a simple game was a success. The application runs smoothly on different devices and the gameplay is easy. Secondly, are the essential features of handling graphics and audio were implemented without issues. The game engine works but the cohesion with different classes remains an issue for future undertaking. As Hunt and Thomas (1999, pp. 57-68) state, cohesion leads to less productivity, higher risks and vulnerable code. Another significant point is the sound manager class. It uses the creational design pattern known as singleton to provide access to it in every other class in the application. This is possible by letting the class take responsibility and make sure that there every only exist one instance of it in the application, which makes it very beneficial (Gemma, et al., pp. 144-153). In contrast, the disadvantages include the fact that the instance needs to be referred to from all other classes and that there may never be more than one instance, which becomes more difficult to manage in concurrent applications (Shalloway & Trott, 2001). Alternatively, other patterns could have been used but for this small application it was the correct decision.

Moreover, the persistence consists of the shared preferences and a SQLite database. Both have been implemented to acquire the experience and knowledge to handle persistence. There have been no problems with the implementation but there is a disturbing factor. The data is stored only on the device and hence can’t be accessed by other devices unless content providers are created (Lee, 2012). A better approach would have been to make use of Google’s Firebase which comes with many benefits like a minimal and simplified setup or the available massive storages (LLC, 2018).

Furthermore, the game provides only one level that only stops when the player loses all the shields. On professional level a proper game design with investigation on core mechanics, level concept and better backstories is state of the art and would have led to a better game (Zechner & Green, 2011).

Each game object’s collision detection is being detected by using a rectangle with the same position and size as its bitmap. This approach is easy to implement and fast to calculate but on the other hand needs fine tuning to find the right size, can prove to be difficult in combination with rotation and can become very expensive when too many game objects are involved (Portales, 2015).

In conclusion, the application works fine and is decently implemented for a small project but would require many improvements and code changes to be useful for further uses.

# References

Cho, J. (2014) *'The Beginner’s Guide to Android Game Development',* 1 edition, Dublin: Glasnevin Publishing.

Derek, J. (2013) *'Android Game Programming for Dummies',* 1st edition, New Jersey: John Wiley & Sons.

Gemma, E., Helm, R., Johnson, R. & Vlissides, J. (1994) *'Design patterns : elements of reusable object-oriented software',* 1st edition, s.l.:Addison Wesley.

Google LLC, (2018) *Top paid in category apps,*   
https://play.google.com/store/apps/collection/topselling\_paid  
(accessed 07 03 2018).

Horton, J. (2015) *'Android game programming by example',* 1st edition, Birmingham: Packt Publishing.

Horton, J. (2015) *'Android Programming for Beginners',* 1st edition, Birmingham: Packt Publishing.

Hunt, A. & Thomas, D. (1999) *'The pragmatic programmer',* s.l.:Addison Wesley.

Lee, W.-M. (2012) *'Beginning Android 4 Application Development',* 1st edition, IN: John Wiley & Sons.

LLC, G. (2018) *Firebase,*   
https://firebase.google.com/  
(accessed 08 03 2018).

Portales, R. (2015) 'Collision Detection', in 1st, (ed) *Mastering Android Game Development,* Birmingham: Packt Publishing, pp. 127-149.

Shalloway, A. & Trott, J. R. (2001) *'Design Patterns Explained: A New Perspective on Object Oriented Design',* 1st edition, s.l.: Addison Wesley.

Steele, J. & To, N. (2010) *'The Android Developer's Cookbook',* 1st edition, s.l.:Addison-Wesley Professional.

Zechner, M. (2016) *'Beginning Android Games',* 1st edition, Portland: Appress.

Zechner, M. & Green, R. (2011) 'Game development 101', in *Beginning Android 4 Games Development,* s.l.:Appress, pp. 62-66.

Table of figures:

[Figure 1: Chart representing the flow 4](#_Toc508234519)

# Demo Sheet:

Mobile Interactive Systems Demo Sheet

Name: Mojtaba Hafezi N-Number: 0771021

App Title: SpaceShooter2D

App Description:

The application is a simple two-dimensional space shooting game. The user can play the game indefinitely and create high-scores which will be saved in the shared preferences.

Gameplay: Use your fingers and touch the upper or lower part of the screen’s left half to move the ship. Shoot by pressing anywhere on the right half of the screen. Destroy as many enemy ships as possible and survive long enough to beat the high-score. Initially, the player starts off with two lives and after getting hit the first time has a two second time frame in which the ship is immune to hits.

Key Features:

* Graphics
* Audio
* Simple game engine

Other Features

* Persistence: Database and shared preferences
* Sensor Input: Vibration and Accelerometer
* Wireless Connectivity: Any gamepad, joystick or D-Pad via Bluetooth
* Handling audio and video

What have you done which addresses the following:

* Design: After prototyping how the individual menus should look like they were created having simplicity in mind. Many users become confused if there are too many options to choose from.
* User testing: I asked friends and neighbours to test the application. The game was tested on three different devices with success and no issues.
* Adapting to different devices: The layouts have got a “weight” attribute to determine how much of the screen space they will cover. Through this method the layout looks approximately the same on different resolutions.

Did you use any external resources such as code, multimedia, text…etc?

Yes, all multimedia used in the project are from the public domain.

List of external multimedia:

* The ships and laser sprites along with the sound effects were from the following website: <https://kenney.nl/assets/space-shooter-redux>
* All title images were created using the “spaced out” theme from the following website: <https://de.cooltext.com/Logo-Design-Spaced-Out>
* Background images were found on: <https://pixabay.com/de/>
* The application icon was created using the application “Iconion”
* Ambient music by frankum: <https://freesound.org/people/frankum/>
* Font: <https://www.1001freefonts.com/>
* Chart: <https://www.draw.io>

Is your application creative? Why and how?

Most of the games created nowadays, make use of available game engines. This application provides everything from scratch and only accesses the android utilities. Furthermore, providing support for Bluetooth game controllers improved the playability and usability of the application. Although the main concept is not new, some game features are creative e.g. the rapid fire of lasers when many ships are close to the player ship.