Wizard Mayhem

Project is an Android game where user can control a character to survive fighting enemies and score points doing so. There are multiple enemy types and levels including a customisable level. High score of the user is tracked and saved online. High scores of all users can also be displayed.

1. Introduction and Showcase

After launching the application, user can display the high scores or start playing by pressing play and selecting a level to play. User can select to make a custom level where some attributes of the game can be selected by the user. When the game starts, the player is centred on the screen and indicated with a green health bar. There are also different kind of enemies with different amount of speed and health points that track down the player and give damage upon colliding. Enemies spawn at random locations on the map where the spawn frequency and which type of enemies can spawn depends on the current level. Hearts that heal the player upon colliding are also randomly placed on the map and their amount is also determined by the level. The map and the amount of player health points are also decided by the level played. From level 1 to 3, the game gets harder as it gets easier to die and harder to destroy enemies. The player can be controlled by the joystick on the screen, and it can cast spells when user taps on the screen. Scores are awarded for each spell hit on the enemies and the high score is updated upon dying. The game can be restarted with the same settings on the game over panel, or the user can navigate back out of the game.

Fig. 1: Main menu and high score table.

Fig. 2: Levels menu and custom level maker.

Fig. 3: Gameplays of different levels.

Fig. 4: Gameplay and the game over screen.

2. OOP design

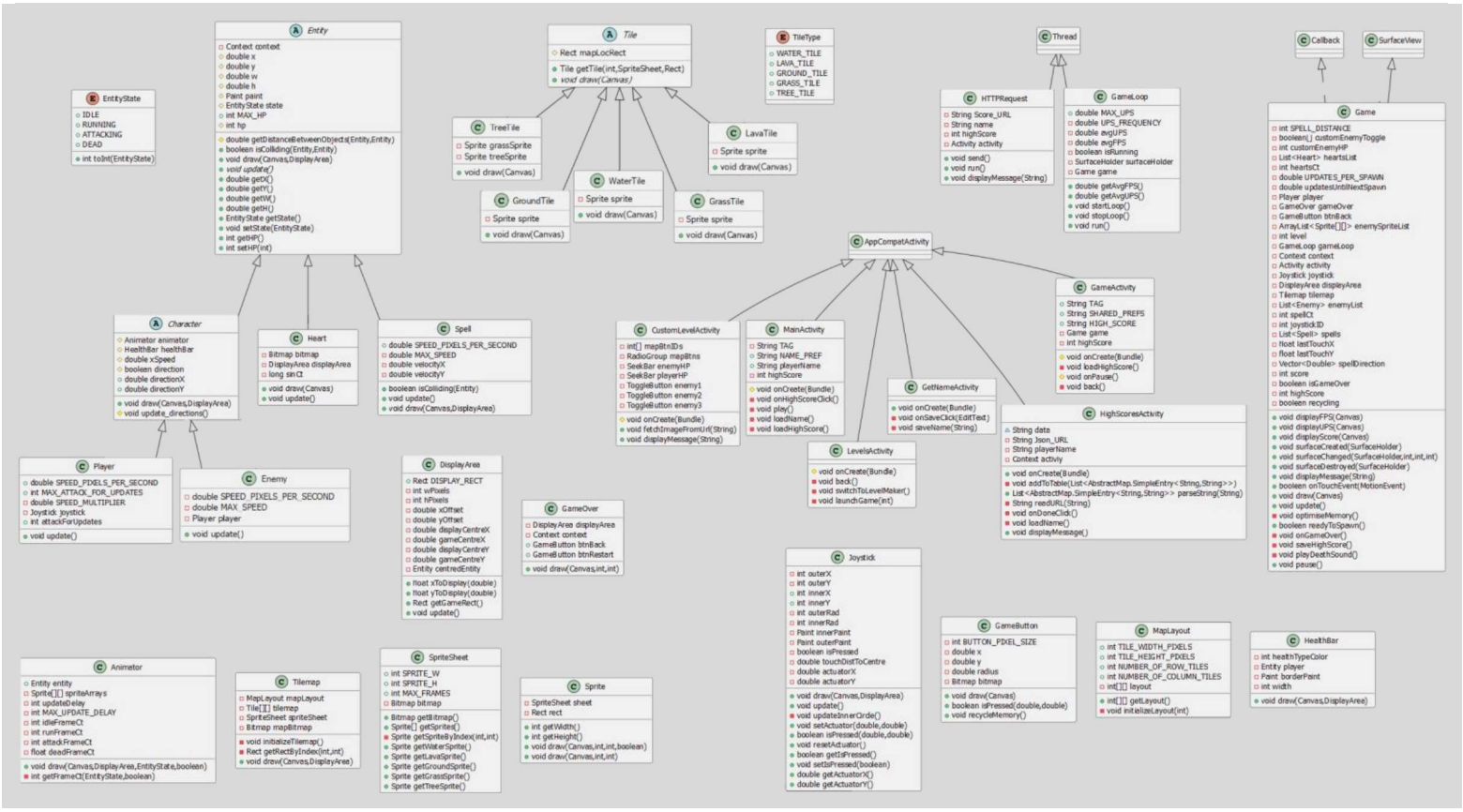

Fig 5. UML Diagram for the OOP Design

To support the game logic, a Game class was designed to manage the game, where a GameLoop that extends the Thread class is created to call the game draw function and the update function that handles the game logic on a separate thread.

The Entity abstract class was designed to represent all the entities in the game, which must have x and y locations as well as size values. These entities can be Heart, Spell or Character objects. The Character class is another abstract class extending the Entity class, and it was designed to represent the characters in the game, which can be a Player or Enemy that both have similar parameters and functionalities such as the method of drawing where animators are used.

Character animations are supported by the Animator class which uses the SpriteSheet class to get Sprites (frames) from a sprite sheet and sequentially draws them for simulating animations. EntityState enum is used to represent the state of the character to determine which animation to play.

The abstract Tile class was designed to represent different types of tiles used for the map which all have map location values and similar draw methods. TileType enum was designed to make it easier to reference the tile types.

Moreover, the application Activity classes that extend the AppCompactActivity class were designed in a way to support the flow of the application where Intents are used to switch between them. The actual game is launched by the GameActivity.

HTTPRequest class was also designed to extend the Thread class to keep networking out of the main thread.

3. Memory usage and Speed improvements

In the game loop design, calculations for updates per second (UPS) and frames per second (FPS) values are implemented and these values are drawn to the screen to monitor game performance.

On initial code design, loading the frames (images) for the enemy and player animations just before drawing them on the canvas was seen as the straightforward implementation. However, this implementation meant that the images were loaded from the files every draw() call, which happens every game loop update. This implementation caused high memory usage and slowed down the game drastically as it was shown by the fall in FPS and UPS values. Therefore, the design was changed to only load the images once in the Game constructor and draw the saved images in draw() calls, optimising the memory usage and the speed.

Moreover, to recycle the memory used by the game button images that are no longer drawn after game is over, a method was implemented to go through each button and call the recycle() method of the button's bitmap (image). By doing so, the memory that is no longer being used is reclaimed (*Managing Bitmap Memory*, 2021). Additionally, releasing the memory used by the loaded audio for sound effects by calling the release() method on the MediaPlayer objects, as well as calling the finish() methods on the activities the user has switched from, optimises the memory usage.

With the initial design for the player spells, the spell would travel in the initial direction until it collided with an enemy. With this implementation, the spells that did not hit any enemies would keep moving infinitely, and never getting removed from the list of active spells, causing a build-up in the memory and wasting CPU time to calculate their position every update. Therefore, the implementation was changed to check if a spell has travelled the distance that is greater than a specified limit every update and, if so, to destroy the spell.

Furthermore, implementing multithreading improved the overall speed of the game as instead of sequentially running the game in a single thread, having multiple threads for different tasks allowed to run these tasks asynchronously at the same time. By executing tasks like sending a HTTP request or playing a sound effect on separate threads, the program does not halt until these tasks are finished, which might take long periods of time.

4. Improvements/extensions

Upon exploring similar Android games before and during the implementation of this project, some features observed in the other games were decided to be good extensions or improvement to this project. These features include character animations, on-screen joystick, hearts, tile maps and sound effects.

As character animations are widely used in Android game development, implementing a design that allows the characters in the game to be animated by utilising sprite sheets with sprites corresponding to character animation frames was decided to be a good feature to implement as an extension to the base game.

Upon researching how touch events are handled in Android, implementing multitouch where a simulated joystick button is implemented while allowing other simultaneous touch inputs to control both attacking and movement of the player was decided as it is possible to track the touch IDs and differentiate multiple touches from each other (*Handle Multi-touch Gestures*, 2022).

After observing other projects similar to this one, improving the gameplay by adding a new type of Heart entity, which the player can collect to restore lost health points, was evaluated to be a good idea as the heart shape is intuitively recognised by most users as health restoring objects in such games.

While researching a way to implement maps for different levels, some game developing tutorials were used for inspiration, and implementing a tile map with different type of tiles that reads the tile image from a sprite sheet was decided to be a good implementation to make (Alex Bükk, 2021).

Finally, seeing that having audio effects is one of the most popular ingredient in an Android game, a sound effect for destroying an enemy was decided to be implemented.

5. Conclusions

In conclusion, OOP principles were exercised in developing of the resulting game, with consideration on the performance and efficiency aspects. Learned new methods of optimisation. A working and enjoyable android game has been successfully created in Android Studio, using Java.

References

Alex Bükk. (2021, August 13). Ep. 16 Tilemap | Android Studio 2D Game Development [Video]. YouTube. https://www.youtube.com/watch?v=rPB_-tiNiik

Handle multi-touch gestures. (2022, August 25). Android Developers. https://developer.android.com/develop/ui/views/touch-and-input/gestures/multi

Managing Bitmap Memory. (2021, October 27). Android Developers. https://developer.android.com/topic/performance/graphics/manage-memory