COMP3003 Software Engineering Concepts
Assignment 1 Report

Shashen Thenuja Gannile Gedara | 20534534

Introduction

- This project is a single-player grid-based game designed using multi-threading principles in Java FX.
- Below are the classes responsible for starting threads
    - **App.java** class is the main thread of the application which is also the GUI thread. It is also responsible for starting the Score thread which updates the player's score throughout the game. A score object will be shared between the threads to update the score.
    - **Fortress.java** is another class responsible for starting a new thread where the player will be able to place walls on the grid during the game. Here the wall requests are created by triggering a click event where the user clicks on any grid square. These requests will then be put into a blocking queue and the new thread will poll the blocking queue and execute the wall requests made by the user. This uses Platform.runLater to invoke the GUI thread to update the wall images.
    - **Movement.java** class is responsible for starting the movement threads of the robots, by using the thread pool ExecutorService, it creates new movement threads for each of the robots that are being spawned. The movement thread uses a hashmap created in the *JFXArena* to update the robot coordinates. The hashmap is created as a concurrent hashmap to make it thread-safe.

- **Spawn.java** is another class responsible for starting the spawning threads of the robots using the thread pool ExecutorService where a thread creates new robot requests by checking the available spawn coordinates which are calculated in the *JFXArena* and puts them into the blocking queue and another thread takes the requests from the blocking queue and executes them. The ExecutorService is shared with the Movement class

- All the threads will be ended by interrupting them using the endGame() method in the App.java class. Also during the game when a robot gets destroyed by an impact, the movement thread responsible for that robot will also end.

Scalability Issues

Non-functional requirements

- Currently, when the game is launched it runs smoothly but as the robots are spawned the application sometimes tends to get laggy since there will be many game objects being drawn to the screen. Considering this game on a large scale there should be definitely a way to optimize the game objects to make the game playable. There should also be a proper way to optimize the performance of the game with faster response time where in this case the wall placing and the robot movement should be faster and smoother. There should also be a proper way to manage the resources that the game uses such as effective and efficient memory usage, optimized data storage and garbage collection to prevent unnecessary memory leaks. Since the game uses multi-threading concurrency should be an important factor.

Improvements

- There are several ways to optimize game objects by using techniques such as object pooling where in this case we could re-use the same game objects for robots instead of creating and destroying images to increase the performance. We could also use optimized multi-threading and parallel processing to distribute the workload across multiple CPU cores which could increase the response times. Dynamic memory allocation can be implemented to store game objects so we can minimise memory wastage.