# AI Team Project

*Group FARCRY's Game*

## WANNACRY

| | | |
|---|---|---|
| Munirudeen Maricar | 2153375 | mxm1287@student.bham.ac.uk |
| Vlad Dimitrie Florea | 2035179 | vxf979@student.bham.ac.uk |
| Craig John James Kerr | 2002609 | cjk909@student.bham.ac.uk |
| Maria Alejandra Rivas | 2031033 | mar933@student.bham.ac.uk |
| Benjamin Norton | 2049819 | bxn919@student.bham.ac.uk |
| Wonseok Choi | 2026739 | wxc939@student.bham.ac.uk |
| Weronika Wiesiolek | 2004147 | www947@student.bham.ac.uk |

## Table of Contents

# I. Introduction

This section will provide an overview of our game *WannaCry*, outlining its purpose and its interface. We aim to provide insight into how different AI methods, integrated into a game, can display a visible semblance of intelligence when competing against a player. Our goal is to create a tense compelling experience where the computer controlled opponents require neither advantages nor additional difficulties to be worthy challenges.

## What is our game?

Put simply, WannaCry is a top-down 2D recreation of the game 'hot potato'. In this game, players will want to avoid infection from a virus. If infected, the player can pass the virus on to another participant on the map. The objective of the game is to maximise the time without the virus, whilst minimising the time in which you have the virus yourself.

There are various game mechanics implemented that intend to make playing the game more compelling; there are plenty of tools at the users' disposal that help to evade the infected player. Players are able to dash, teleport or use their terrain to evade or catch their opponents. The player can also utilize items scattered around the map for a temporary advantage.

## Game Design for AI Development

These game mechanics serve a secondary purpose as well - in order for the intelligent behaviours of our bots to be on full display, we gave the AI opponents these tools and items, in the hope that the more complicated AI architectures can see their merit and use them effectively to defeat the player.

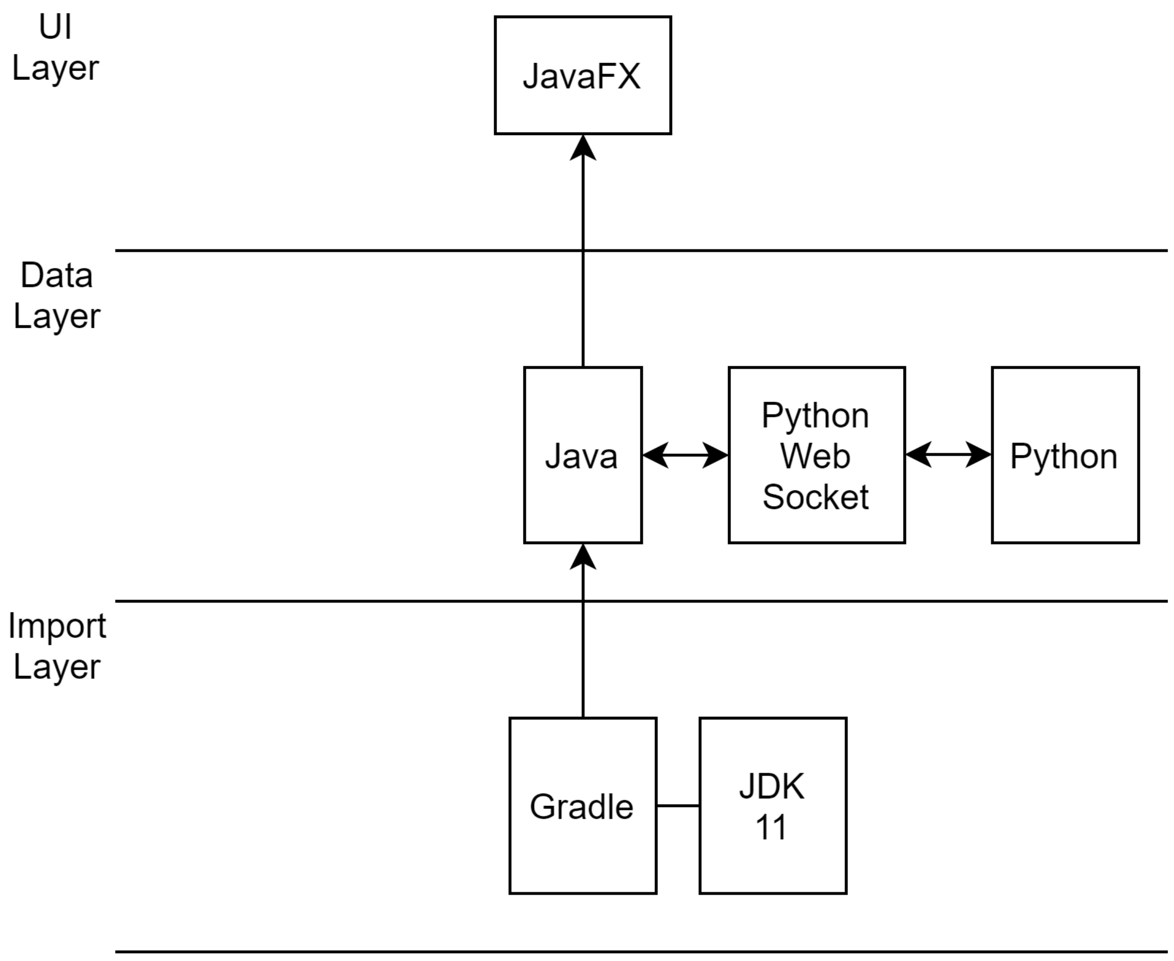
A score-based system will decide the winners. After each round, player scores will be calculated based on each individual's total infection time and the last player to be infected. The player to top the leaderboard of our game would have evaded the virus for the longest period of time.

## Tech Stack

Our application requires the integration of several technologies to function correctly. The diagram below shows:

- The game engine.
- How the game engine communicates with our AI implementation.

- How to handle installing dependencies to run the game (all of them are handled by Gradle)



## Assumptions in our Project (IEEE-829-5):

1. The user must have JDK 11+ installed in their system.
2. The user must have Python 3+ installed in their system.
3. Gradle will automatically handle the project dependencies required for the user.

# II. Requirements

We have divided the requirements of our game into functional and non-functional requirements, with each being labelled with one of **[M]**, **[S]**, **[C]** to demonstrate the importance of each requirement. We **must** implement requirements labelled with **[M]**. Those labelled with **[S] should** be implemented but are not critical. Finally, those with **[C] could** be added to the game but are low priority.

## Functional Requirements

Functional requirements are the criteria which  describe how the inputs and outputs of our game should behave . Below is a list of functional requirements, grouped into categories to clearly show their role:

### Human Players

1. The game must have at least one controllable player. **[M]**
2. The player must move on the map using the input of the W, A, S, D keys or the arrow keys (arrow keys for the first player and W, A, S, D keys for the second player). **[M]**
3. The player must be able to pass the infection onto a non-infected participant in the game. **[M]**
4. The players must not walk onto non-traversable tiles like walls. **[M]**
5. The player should utilize special items by using specifically assigned hot-keys. **[S]**
6. The player sprite could be customizable from a preselected library. **[C]**

### Bots

7. The game must implement at least two functional bots in the game that can compete against a player. **[M]**
8. All bots must operate within the bounds of the map. **[M]**
9. All bots must not walk onto walls or non-traversable tiles. **[M]**
10. All bots' behaviour should be controlled and decided by its respective AI algorithm. **[S]**
11. All bots should be able to use special tiles and abilities to the extent specified for their particular algorithms. **[S]**

## Map

12. The game map must operate with a top-down grid view. **[M]**
13. The game must provide one working map interface to play the game. **[M]**
14. This map must have collision physics, as well as set out the stage for strategic gameplay. **[M]**
15. The map must have different tiles affecting the movement and speed of actors. **[M]**
16. The map should have items appearing in particular  locations that can be utilised by the players to their advantage. **[S]**
17. The game could have different maps to allow the user to choose which map they would prefer. **[C]**

## Audio

18. The game should have background music and sound effects that can be adjusted according to the user's desires. **[S]**

## Items

19. The game must have various items in the game that can alter the behaviour of a player to act as power boosters or dampeners. **[M]**
20. The different items that must be implemented are:
    a. *Freeze*, which freezes all the enemy players close to the player who uses this item. **[M]**
    b. *Immunity*, which makes you immune from receiving the virus for a short period of time. **[M]**
    c. *Quarantine*, which creates a circle around you. While inside this circle, you will have increased movement speed while the enemy players inside the circle will move slower. **[M]**
    d. *Silence Arrow*, which shoots an arrow in the direction in which you are moving (either up, down, left or right). If the arrow hits an enemy player, that player will be unable to use any items or abilities. **[M]**
    e. *Time Extender*, which extends the time remaining in the game if you use it when you have the virus. If you don't have the virus you can only increase the time but if you have the virus then you can also increase your score. **[C]**

21. The different items must be spawned at random walkable tiles throughout the map. **[M]**
22. The game could have ultimate abilities that require a longer period of time to charge up to use to their advantage. **[C]**

## Game Basics

23. Each round must consist of the players going against the specified number of bots (the number of bots can range from 0 to 3). Each round must consist of at least two players (human players or bots) going against each other. This means there might not be bots in the game because we can play with our friend using the same keyboard. **[M]**
24. The game must allow the player to quit during any point in the game. **[M]**
25. The game must allow the player to restart during any point in the game. **[M]**
26. The game must allow the player to restart after the round has ended. **[M]**
27. The game must begin with at least one healthy player character and exactly one player infected. **[M]**
28. Once each round has concluded, the game must display a menu. **[M]**
29. The game could have a developer mode allowing AI to compete with each other without the participation of human players. **[C]**.

## Game Feedback

30. The game must show the real-time scores of all the players by pressing a key. **[M]**
31. The game must show which player is currently infected. **[M]**
32. The game must show the list of items that each player possesses. **[M]**
33. The game could show the state of the teleport and item cooldowns. **[C]**

## Score

34. During gameplay, the score of every player must be updated. **[M]**
35. Points must be calculated using the same method based on the performance of both the players and the bots. **[M]**

36. Points must be calculated based on how long the player was infected in one round. **[M]**
37. Points must be deducted if and only if at the end of the round, the player is infected. **[M]**
38. Every uninfected player should gain 60 points per second, meaning that the infected player will not be able to passively acquire points. **[S]**
39. At the start of the game, the player who starts with the virus should be given 4000 points. **[S]**
40. The player killed by the virus should lose 4000 points. **[S]**
41. Upon silencing any player with the *Silence Arrow* item, the player should gain a 1000 points. **[S]**
42. Upon freezing any player with the *Freeze* item, the player should gain a 500 points. **[S]**
43. Upon using the *Timer Extender* item as the infected player, the player could gain points equal to a fraction of the added time. **[C]**

## Socket Communication

44. The game state must have a client-network model with the client being Python and the server being Java, allowing the two languages to interact with one another. **[M]**
45. The bots chosen by must be able to send data from and receive data for the game. **[M]**
46. The client server connection must be successfully closed when the user quits the game. **[M]**

## Non-functional Requirements

Non-functional requirements are the criteria that judge the operation of our game rather than specific behaviours. These are:

1. Each round must end after 30 seconds.
2. The map must be displayed after an average of 10 seconds of starting the game.
3. The average frame rate must be around 50-60 frames per second (although this varies depending on the machine running the game). Frame rate can be seen on the top right corner of the game.
4. The average response time between a keyboard key press and its reaction must be less than 0.2 seconds.
5. The game must not gather users' personal data other than what is required for the game. As a result, the game must comply with the Data Protection Act and General Data Protection Regulation (GDPR).
6. The game must not violate any copyright laws.
7. The game should not crash whilst the user is playing it.
8. The game must require less than 512 MB of hard disk space. The RAM required to run the game must not exceed 2048 MB (RAM usage depends on the game mode).
9. The Java server should establish a connection with the Python client within 10 seconds or less.

# III. Software Design

## Class Diagrams

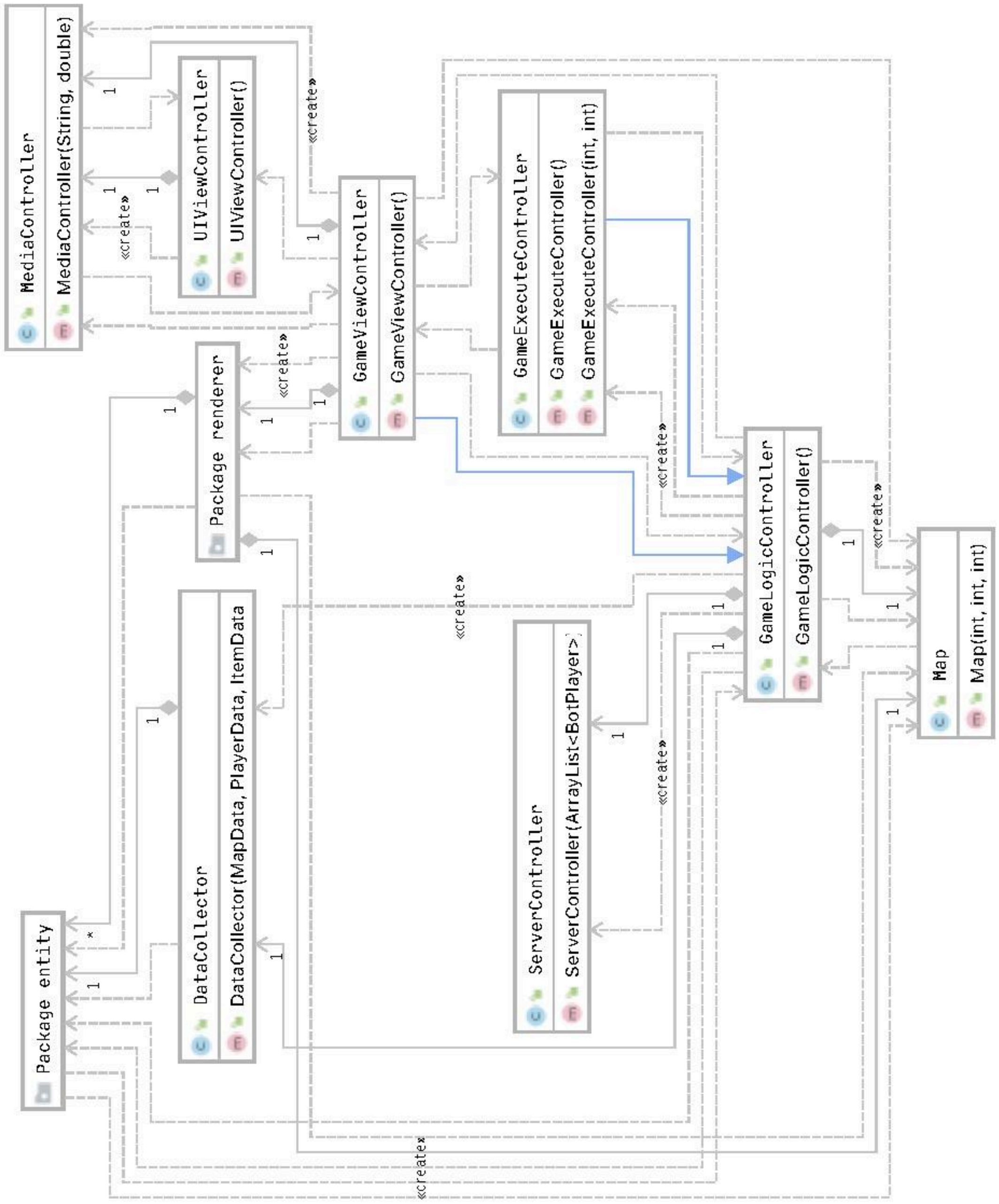


Figure 3.1 - Packaged Class Diagram 'Wanna Cry' Java Game System. See Appendix for detailed diagrams and Python Class Diagram

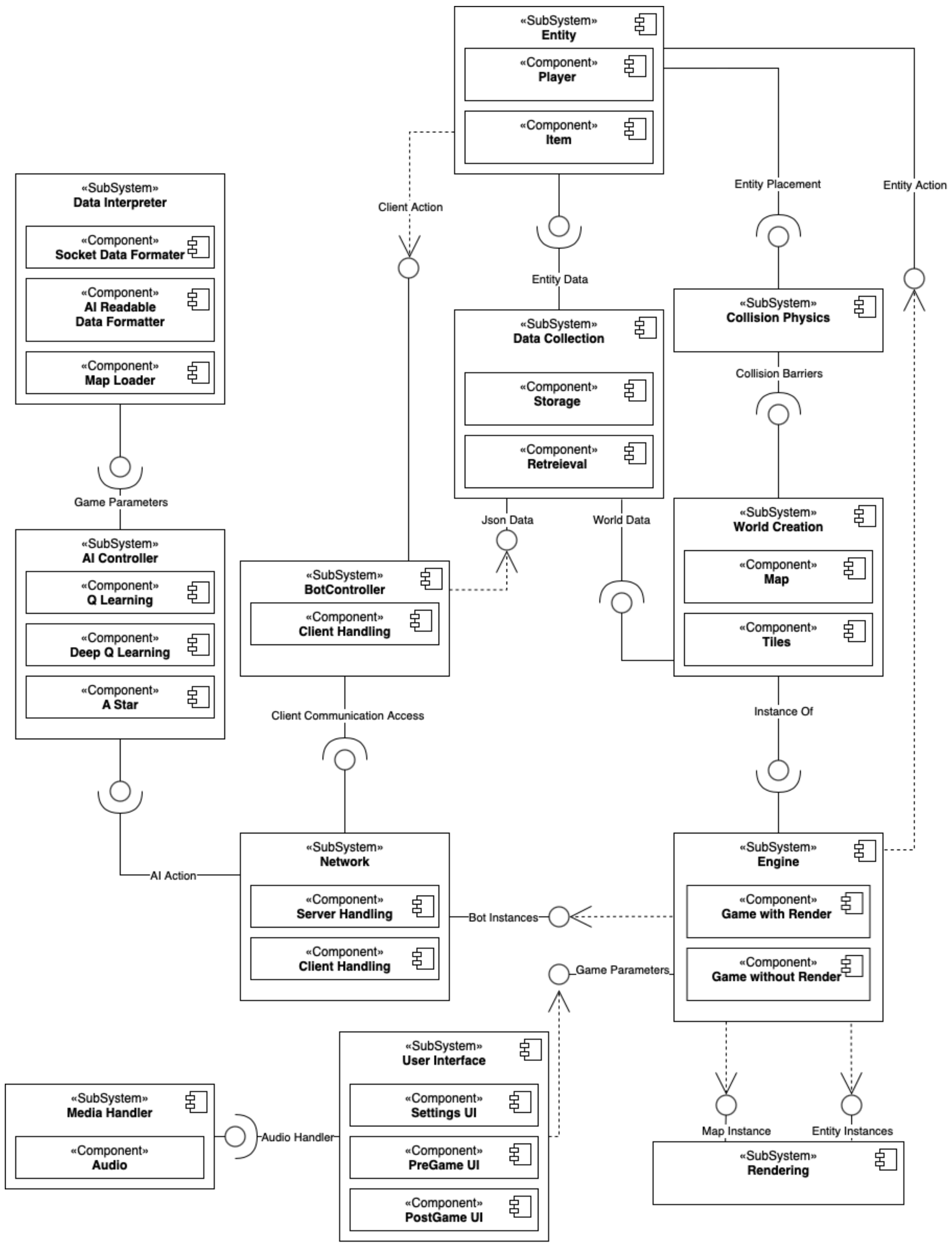


Figure 3.2 - ‘Wanna Cry’ Component Diagram - High Level View of Entire System. Python and Java Included.

## Software Architecture

'WannaCry' was adapted to fit an architecture that requires heavy calculations and ongoing network communication. For this reason, the front-end and back-end work independently. Both have been designed with different purposes in mind.

## Structure

The back-end fulfils the main duties of Data Collection, Entity Management, Networking, AI, World Creation and Physics. To make this as efficient as possible the team adopted a FlyWeight Structural Pattern where heavy data transfer would have occurred. An Example of this is in our Data Collector Sub System. Figure 9.22 has been provided in the appendix to better illustrate this point. Due to the complexity of the system and the requirement to keep various data points on many object instances, adopting this pattern has enabled better performance and more seamless communication between AI and the game. The data is directly given to the methods that will use it, thus, making the most of a costly procedure such as socket communication.

However, FlyWeight' benefits stem from optimization. To improve the system, having an intrinsic creation mechanism was a necessary basis.

## Creation

Our priorities listed as the following;

- Enable easy scalability of the product: mainly concerning the growing complexity of the project during its development cycle.
- Permit instances of an object to be created independent of class: Due to the numerous subsystems in the game,  it is not unusual for systems to require data points stored elsewhere in the program for initialization. To prevent unregulated static referencing  the team adopted a methodology that allowed for instantiation of said objects directly through the constructor.
- Allow for an application initialised under different configurations: The game has no predetermined values of creation; this means that the player can select a game with 0 to 3 different types of bot behaviours. Any of these would require a different set of Entity arrangements as well as Networking setup.
- Support object creation based on a limited predetermined set of enums. Instances, where this is needed, would be observed when initialising tiles and items.

To begin with, addressing our first priority in the list, the game engine uses a Factory Creational Pattern. Figure 9.1 in the appendix illustrates this point. Choosing this pattern allows for the separation of construction code and actual game logic which brought about the inherent practicality when adopting new game features and scheduling behaviours.

This design also allowed for easy distinction between 2 main methods of running the game. The main executable includes a variable `runGameNormal`, which allows for toggling between the *game mode* and *developer mode*.

## Game Engine

The game engine consists of 3 components: GameLogicController, GameViewController, and GameExecuteController. They enable for game execution, component binding and main game lop creation. They allow for 2 different modes of execution.

A game executed in *game mode* operates with full UI displays. The front-end and back-end meet in the main game engine. To prevent performance issues caused by the heavy physics or rendering, and movement mechanics performed by the AI, we threaded these two communications. In principle, it is also possible to vary the tick rate between logic and graphics, thus allowing for very high quality of the graphics.

A game executed in *developer mode* is not rendered. This is in order to speed up the AI training and avoid unnecessary computation or threading exceptions. Because we adopted the Factory approach, this engine reuses the inherited logic from the creator: Game Logic.

It also allowed for easy distinction between the two main game options. A rendered game would operate as expected, having full UI displays. This would be the only instance in which the front-end and back-end would meet. To prevent performance issues caused by the heavy physics and movement mechanics performed by the AI, we threaded these two communications. Another benefit comes when adding the un-rendered version of the game, used for training the AI bots and debugging. Because we adopted the Factory approach, the engine now reuses the inherited logic from the creator: Game Logic.

Our second and third priority is addressed using Dependency Injection. Figures 9.6 and 9.8 are provided under the appendix. Networking  component is contained in a thread separate from the main process. . However, its creation relies on the number and types of bots requested by the user.

Therefore, rather  than having bots exist as a static object, which limits management of access, the ServerController uses a construction injection from the injector, GameLogic, to initialise itself. Another example of this would be found in the Data Collection subsystem. Similarly, we pass MapData, ItemData and PlayerData to the DataCollector. These three parameters require specifications stored in GameLogic. If, hypothetically, the system is to exchange these to DataCollector and back through PlayerData, for example, we would encounter a higher likelihood of cyclic referencing, which may cause errors.

Finally, the last priority is addressed by using Multiton Creational Pattern. Figure 9.5 in the appendix displays an example of this. The World component of our game is created from a grid-based system of tiles. Each tile comes with inherited behaviours and properties. When created through Map, constructing the tile from an enum of TileTyes allows for multiple objects to be created whilst avoiding the redundant passage of data.

## Actors

Actors are groups of potential user entities in our game. Below, we will describe each actor and how they will use the game:

1. Players

   Players are characters that a user can control. The player can either use the arrow keys or the WASD keys to control their respective avatar. Up to two human players can play the game simultaneously.

2. Bots

   Bots are characters controlled by an AI algorithm. The number of bots within a game varies based on the users preference, and can be removed entirely. Each new bot added to the game uses a different AI architecture to interpret the game. This could be the A* algorithm, Q learning or deep Q learning approaches, or an Actor Critic model.

## Networking

The bots in the system  required a network infrastructure that would lend itself to a MultiAgent approach. This connection is achieved throughout the use of TCP Java WebSockets. To be specific, the network is set up under a  Java Server to Python Client communication. This SubSystem is demonstrated in Figure 9.6 in the appendix.

Once java has established a safe connection with the python Client, it will thread that socket connection and assign the new client handler to a BotParent (The bot parent works like a common brain for Bots under the same AI behaviour)

Because the communication and data exchange is happening live-in game. The team took some precautions to prevent excessive memory usage. These include:

- Reducing the frequency at which data is transferred (Exchanges happen only once a goal state has been reached)
- Reducing the data points sent overall. (Reduce complex map behaviour into coded strings)

- Filtering data points sent. (Bots would only receive data appropriate to their behaviour).
- Minimizing the bits sent per flush.

These allowed for uninterrupted and a fluid trafic of information between the Bots.

This connection is maintained active during the duration of a game, only once the player closes the application or the game concludes  will the server close all communication.

## Behavioural Pattern

Our game can subdivide into three different aspects:

- **User Interface:** Pre-Game, Post-Game, In-Game menu handling, game rendering and feedback response
- **Primary game engine:** This covers game creation and management, excluding AI and training.
- **The AI game engine:** Covers creation, resource management, learning and adaptability of the AI.

Both User Interface and the Primary Game Engine adopted an Observer Behavioural Pattern.

**User Interface** class diagrams can be seen in the appendix under figures 9.9 and 9.10. The Menu part of the interface includes multiple items that can adopt many states, buttons, scroll bars, drop-down boxes. Rather than having methods that would be costly and iterative, utilising event handlers, the observer allows for perceptive feedback rather than one reliant on a roundabout communication. This allows for the many dependencies that can stem from these to be tightly coupled into a single one. Similarly, for rendering we adopted an Adjusted Behavioural Pattern, discussed further below.

**The primary game engine** can be seen in the appendix under figures 9.4.  Our team developed an adjusted Observer Pattern. Rather than having an event handler observe the change, we added an in-game observer in the game loop. This allowed for the same perks of the Observer, such as having a change in the state response by changing multiple dependencies, whilst keeping up with the multi-subsystems communication of our engine. Although not part of the primary game engine, rendering operates similarly.

**The AI game engine**. For this component of the system, we employed two distinctive design patterns often used for AII: the Flyweight Behavioural Pattern and Mediator. The Flyweight is utilized in the DeepQLearning and Actor-Critic Architecture and Mediator in the Q-learning and A-Star Architecture. It was important to select a suitable architecture for the appropriate AI, as seen below.

## Artificial Intelligence Architecture

Creation Requirements:

- Avoiding bottlenecks from incoming large data points.
- Allow for clones of similar behaviour to be called and created in the appropriate event.

Training Requirements:

- Saving and loading methods that would return data once stored throughout the episodes in the application's heap. For example: after every round, and after every game.
- Updating the existing data and recognizing similar states/nodes during training.
- Sustaining a multi-agent platform and hypothesis update.

Gameplay Requirements :

- Quick decision making.
- Smart judgment from the bot: Item usage, infection and enemy player tenacity. (Behaviour varies based on AI complexity)
- Distinct 'fleeing' and 'chasing' behaviour.

## A* and Q-Learning

The first AI to be developed was an A*-based bot. Initially, the only thing we have implemented was the pathfinding component, with a deterministic algorithm whose heuristic function was the distance on the map, accounting only for own position and enemy positions.

As the game progressed, we have spotted 2 key challenges:

1. The speed of the fleeing bot if iterating through all possible tiles made it impossible to play the game at a reasonable speed.
2. The further challenges posed by Q-learning even after improving it inspired us to implement some of said enhancements in A* to improve the bot's playability.

We addressed that by respectively:

1. Performing extensive out-of-game algorithmic testing of our environment, and excluding the points which will *never* be an optimal position from any other point on the map.

2. Enhancing the complexity of A* bot to account for more human-like behaviour:
    a. Modelling the map environment as a non-Euclidean space - Tiles such as conveyor belts with speed multipliers we modelled as *smaller* than regular tiles, forming a kind of reciprocal space with respect to the velocity multiplication. We also scrapped the symmetry assumption (getting from point A to B isn't necessarily as fast as getting from B to A).
    b. Randomisation of movement when fleeing bots are sufficiently far from the chaser to cause more opportunities for the chaser to catch them.
    c. Enabling teleport usage and item usage when it's optimal.
    d. Setting a preference for them to chase bots of other kinds before A* bots - this was to mitigate the effect of 2 A* bots looping in a cycle of mutually infecting one another in a confined space
    e. Creating sliding off the wall physics to mitigate the effect of bots being stuck in corners (otherwise they often had a preference for diagonal movement due to its efficiency, which caused them to fail to move in 3 out of 4 possibilities)

The second part, Q-learning, posed a lot of challenges. Firstly, we implemented a regular Q-table based on the Bellman equation. However, upon calculating the overall number of states, we saw that treating all of them as distinct states will not be feasible. Therefore, we adapted the algorithms to account for similar states by reducing the resolution of the location, and calculating whether a similar state has been achieved before as read in literature. We also applied logarithmic scaling and normalisation on the rewards, and compared the results.

However, the performance was still far from ideal, rarely exhibiting any visible patterns of behaviour. This is why we have taken a two-pronged approach of developing another AI in the meantime, and refining the A* bot to ensure a satisfying gameplay alongside any research-focused conclusions.

## Architecture Design

For Q learning, the states themselves would provide the difference between agents, (i.e. infected states correspond to the chasing agent and vice-versa). And as for A Star, this check could be performed as soon as infect status was found. It was decided that a head-on approach would be the best for both AIs. To summarise, this consisted of visualising Q-table/A Star Algorithm as the object priority. Agents were to be decided by a simple check method. This was regarded as essential since, taking a different technique, would have added some unplanned days to the development.

As for the AI execution, both Q-Learning and AStar, take in a Mediator design. Because of the above structure, the AI required for a mediator. In this case, both AStarSolver and

QNetwork fulfil this purpose respectively. The services provided come from the Network and Environment building components. Even though it would be ideal for the AI to perform on the optimum data updated, these can be lost/stall on their way. Ergo, 'loose' coupling of the classes was preferred. Although a conclusion based on partially updated data is not ideal, a deadlock/thread stall or thread latency was seen as a bigger issue. For this reason, the mediators can still carry out their task and send Java the necessary steps even in these circumstances.

The loose coupling of these components also allows for reusability. In specific this reusability is performed by DQN, which uses A StarSolver as an aid for reaching the optimum concurrent path to/away from the player.

Reference Figures 9.18 and 9.19 for the illustrated example on A star and Q-Learning respectively.

## Complex AI: Deep-Q-Network(DQN) and Actor Critic (A2C)

As stated above, development of Machine Learning algorithms in the Tensorflow library was done to research alternatives to the Q-learning bot.  The overall goal was to try out state-of-the-art architectures, adjust them to our games, and spot any emergent behaviours occurring. As for the DQN and A2C we were able to observe the following patterns when training:

1. Mimicking A* when running away - the algorithms often hid in the same “safe spots” as used by the A* algorithm.
2. Using teleporters - sometimes we could spot the bots turning back when chasing or fleeing, and use a teleporter which was the action closer to optimal.
3. Blocking each other at 2 opposite sides of an obstacle - the bots then moved left or right, hesitant to decide which way to go.
4. Preferring the conveyor belt over normal tiles - they are quicker, and only accessible from one side.

However, we also observed the following flaws:

1. Being stuck in corners - this is difficult to address as the bots did not have *a priori* knowledge of the environment.
2. Insensitivity to crossing paths with enemies - this could be mitigated by a more complicated architecture,which also took into consideration players’ velocities.
3. Slow convergence - this was mainly due to our machines being slow, and the need to constantly improve the model.
4. Looping - after convergence, the bots often looped in a fixed path. This could be mitigated by playing with human opponents to increase feedback randomization - however, this would be very time consuming.

## Architecture Design

The DQN engine follows a Flyweight pattern. However, to support a multithreaded platform, we also needed to provide:

- a coherent player to bot communication.
- a distinction between Game State AI and Training State AI.
- a creation that would lend itself to easy recyclability amongst agents

Following the general requirements and specialised focus stated above, the team decided on taking an Adjusted Prototype creational pattern. This means that the AI can inherit the benefits of the default 'original' agent, and still have an adjusted independent behaviour based on specific rewards and goals.

When it came to creating the actual models for DQN and A2C to behave in, because python was available, the team incorporated AI libraries that would help us process the data and model the behaviour after. These include MatPlotLib, TensorFlow and NumPy.

Using these libraries lead to the implementation of Flyweight for our behaviour. The benefits to flyweight have already been stated, however, one advantage that is specially catered to AI is the bottleneck avoidance caused by the large amounts of data processing and decoding at any given time. For that reason, we too saw fit to give each AI their own LogInterpreter class, as the original one could no longer lend its purpose to DQN and A2C in a timely and efficient manner. In addition an adjusted Python Client script was also provided to each in order to utilise the virtual environments specific criteria.

With all of the requirements met, this means that DQN and A2C can operate their own Environment, Agent behaviour and data extrapolation. This also renders them an efficient use of screenshots to fit weights and goals as specified by the programmers. Which, as for gameplay and training relevance, means that DQN and A2C have had:

- A rapid training stage
- Natural and precise in-game behaviour
- Receptive feedback when competing against a human player.

A foreseen drawback was the redundancy of data exchange from Java to the multiple AIs, nonetheless, it was one the team was willing to take in exchange for the benefits.

Figures 9.20.1 and Figure 9.20.2 provide the architecture implemented for DQN and Figure 9.21.1 and Figure 9.21.2 for A2C.

# IV. Interface Design

In creating the user interface, HCI (Human Computer Interaction) principles were carefully followed to provide the user with the best possible experience while interacting with the UI components. We will take a look at some of the principles below:

## 1. Fitts' Law

Fitts' Law is a crucial principle which helps enhance the user's ability to navigate through the interface. Following this principle:

- We made all the buttons and sliders big enough so that the user can quickly point to any button or slider, resulting in the time spent for aiming to be as low as possible,
- We used familiar designs, where the buttons are strategically placed where the user would expect them to be. For instance, in the main menu, the play button is the first from top to bottom, and the quit button is the last, which is the same as in many other games.. Another good example is the fact that the game can be paused by pressing on the *Esc* key, which is the case in almost every other game that has a pause function.
- We made all the screens the same size which gives the feeling of organisation to  our game.

## 2. Poka-Yoke

The Japanese principle known as Poka-Yoke (translated as "error proofing") is when we show a confirmation message when the user tries to restart or quit while the game is running. Not only does this allow us to prevent any irreversible action from unwantedly taking place, but it also makes the user not worry about miss-clicking on a button that will cause the game to completely end.

Furthermore, there are several constraints with the username that must be satisfied for the user to move to the next stage. The username chosen should be valid and and have the correct length for the *continue* button to be enabled.

Moreover, the button keys that the player uses to move/use abilities/use items are close to each other, so one player can easily play the game with one hand, without needing to reach to the other corner of their keyboard to perform an action in the game. For example, the player who uses WASD to move also uses Q and R for his abilities and Shift for using an item. Since all of those keys are close to each other, this gives the player a positive experience while playing the game. However, since we also have a two-player game-mode (using the same computer), we needed to separate the keys that the two players will use and thus, one player will interact with the keys located on the right side of the keyboard, while the other will use the keys on the left side.

Additionally, while the game is loading, the player isn't simply left with a non-responding UI to look at, but he is presented with a loading screen which contains a progress bar that updates the user about how much of the loading has been completed, as well as a progress text which provides more detailed information about what is going on behind the scenes.

Furthermore, we are displaying crucial information by using the information button, which will present the user with  necessary contextual information when hovered over. For instance, when choosing a username, the user has the option to input two different usernames, separated by a comma, in order to access the two-player game-mode. We have placed this button right next to the username text field so the user can hover over it to see the relevant information.

Also, the buttons become brighter whenever the user clicks over them and they also have a sound effect associated with this action. This makes sure that the user's cursor is on the right position and it gives a clearer impression whenever they want to make choices. Similarly, we also differentiate the font size according to the importance of the information in the screen like the column names for the leaderboard.

## 3. Hick's Law

Hick's Law dictates that the greater the number of choices, the longer it takes to make a decision. By using this law, we decided to present a few crucial options in the main menu followed by a subsequent game options menu rather than showing everything in one single screen. This design reduces the user's decision time by dividing the options into multiple screens).  We also made a helpful design by displaying the cooldowns in the game, as well as the inventory of each human player so that players can know which item they picked at any time during the game. For example, if the player can't use an ability, it's corresponding image will be grayscaled, which helps them in identifying whether or not he can use that specific ability.

# V. Software Engineering

After carefully considering the different software engineering principles we encountered from our module last year, we can conclude that we implemented some (or, to be more precise, a mixture of several) into our project, as shown below.

## 1. Agile Development

We have implemented agile development in our project by iteratively adding features to our game (*Incremental and Iterative Development*). Our game's requirements and solutions to those requirements evolve continuously after our weekly meetings with the entire team. We use feedback from one another to continually work on the project in order to refine it and deliver a final working product. We also implemented *Pair Programming*, wherein two members of our team work simultaneously on one task where one would write the code and the other would be proofreading the code as it is typed in.

## 2. Continuous Integration

We have implemented continuous integration in our project by integrating the work done by each member of the team approximately once a week. Our code was fully integrated before every meeting with our supervisor. The integrated code base resides in the *master* branch of our GitLab repository.

## 3. Rapid Application Development

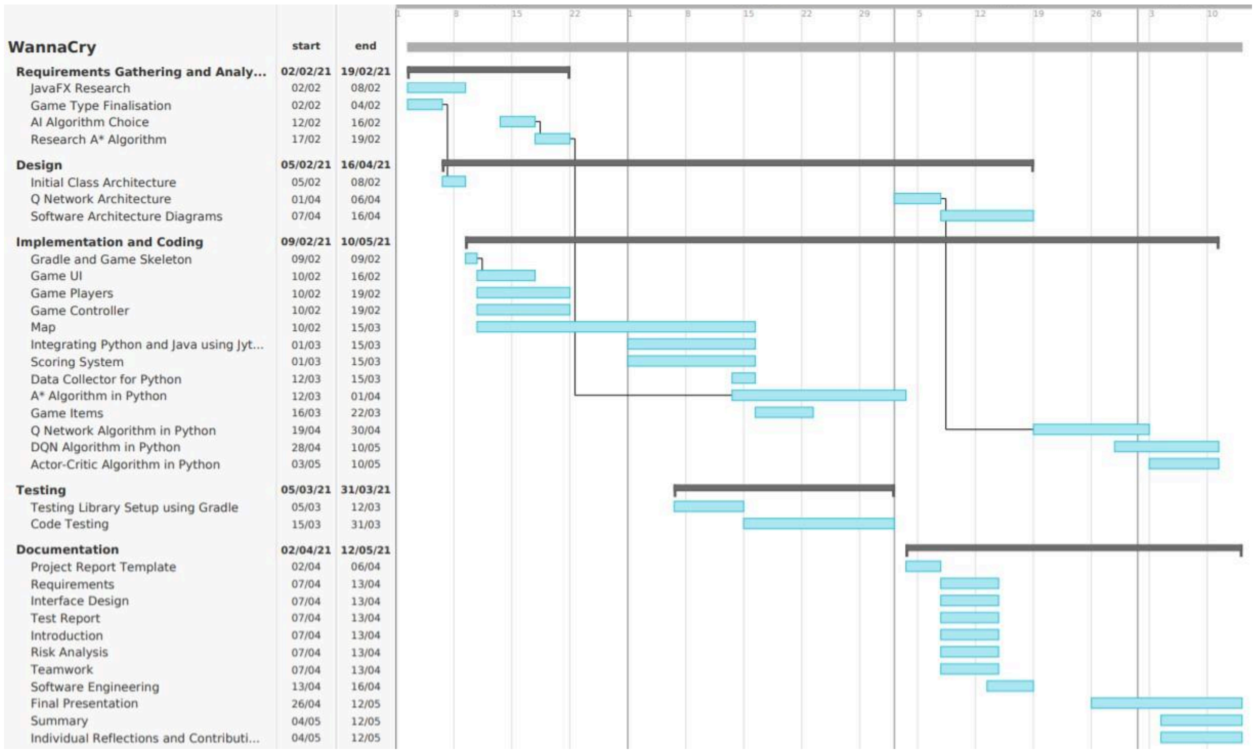
Since we were given a limited amount of time to complete our project to begin with, we implemented a rapid application development in our project by interleaving our initial planning with writing the software itself. We reduced the inherent project risk by breaking the project up into smaller segments such as UI, audio, AI, map, and more to provide more ease-of-change during the development process.

Our project incorporated a mixture of many different software engineering processes, mostly derived from the Agile Development approach. We did not have much time available for the planning aspect of the project. On top of that, since we started the coding quite quickly, we had to continually integrate all of our codebases. Hand in hand with this, our project also grew iteratively and incrementally by slowly adding one feature at a time.

Pair programming also came in handy. From this, it is clear that all three software engineering processes were pivotal to our game's creation.

## Timeline

The timeline for our project had to be constantly revised and reevaluated if necessary, as we took on the Rapid Application Development methodology. Our final timeline can be seen on the following Gantt chart:



| WannaCry | start | end |
|---|---|---|
| **Requirements Gathering and Analy...** | **02/02/21** | **19/02/21** |
| JavaFX Research | 02/02 | 08/02 |
| Game Type Finalisation | 02/02 | 04/02 |
| AI Algorithm Choice | 12/02 | 16/02 |
| Research A* Algorithm | 17/02 | 19/02 |
| **Design** | **05/02/21** | **16/04/21** |
| Initial Class Architecture | 05/02 | 08/02 |
| Q Network Architecture | 01/04 | 06/04 |
| Software Architecture Diagrams | 07/04 | 16/04 |
| **Implementation and Coding** | **09/02/21** | **10/05/21** |
| Gradle and Game Skeleton | 09/02 | 09/02 |
| Game UI | 10/02 | 16/02 |
| Game Players | 10/02 | 19/02 |
| Game Controller | 10/02 | 19/02 |
| Map | 10/02 | 15/03 |
| Integrating Python and Java using Jyt... | 01/03 | 15/03 |
| Scoring System | 01/03 | 15/03 |
| Data Collector for Python | 12/03 | 15/03 |
| A* Algorithm in Python | 12/03 | 01/04 |
| Game Items | 16/03 | 22/03 |
| Q Network Algorithm in Python | 19/04 | 30/04 |
| DQN Algorithm in Python | 28/04 | 10/05 |
| Actor-Critic Algorithm in Python | 03/05 | 10/05 |
| **Testing** | **05/03/21** | **31/03/21** |
| Testing Library Setup using Gradle | 05/03 | 12/03 |
| Code Testing | 15/03 | 31/03 |
| **Documentation** | **02/04/21** | **12/05/21** |
| Project Report Template | 02/04 | 06/04 |
| Requirements | 07/04 | 13/04 |
| Interface Design | 07/04 | 13/04 |
| Test Report | 07/04 | 13/04 |
| Introduction | 07/04 | 13/04 |
| Risk Analysis | 07/04 | 13/04 |
| Teamwork | 07/04 | 13/04 |
| Software Engineering | 13/04 | 16/04 |
| Final Presentation | 26/04 | 12/05 |
| Summary | 04/05 | 12/05 |
| Individual Reflections and Contributi... | 04/05 | 12/05 |

# VI. Risk Analysis

Understanding the risks associated with our game is vital in identifying the components that might present a threat to the game's functionality or performance. We shall now discuss the main categories of risks we have identified.

## 1. Software Development Approach

Our team wanted to make sure that we deliver a robust, playable game, and learn a lot about the AI architectures. This posed a significant challenge, especially as we wanted to establish network communication for AI training, which essentially added an additional component to our game. We have addressed that by:

- Focusing on class diagrams and high-level, rapid architecture design at first.
- Dividing the game into components from the initial meetings to ensure that tasks are not redundant.
- Fast MVP delivery.

We addressed most organisation difficulties by establishing a Communication Strategy Framework, which included three key components:

1. Weekly meetings with minutes to mitigate the risk related to absence.
2. Channels and pinned messages to mitigate the risk of losing information.
3. Personal chats with teammates to ensure everybody feels comfortable and mitigate the risk of suppressing crucial communication.

We also needed to make sure that no one works on something that they aren't comfortable with. For example, some team members preferred not to work with the most experimental parts of the AI, in which case, they would work on other parts in order to increase their productivity, including AI training or testing.

## 4. Errors

One such problem is an error occuring while playing the game. This could be caused by a change in the code that might break other parts of the system, without us even noticing. If the game is deployed and this error remains unidentified, it can have negative effects on the user's experience. In order to make sure all the components work as expected and there aren't any game-breaking bugs, an extensive test plan was created. There are automatic tests which test many crucial components such as the UI, items or the player abilities, which greatly reduces the probability of an error not being discovered by manually playing the game.

The most common types of errors we have addressed are:

- Critical faults leading to game crashing,
- Faults which did not lead to game crashing, but impacted the game mechanics or options,
- System-specific execution bugs,
- AI errors which resulted in biased training.

Each of them were addressed in our extensive testing plan. The test table, as seen in the Appendix, contains various tests corresponding mainly to the 1st and 2nd category of game faults. These are automatic tests which examine many crucial components such as the UI, items or the player abilities, which greatly reduces the probability of an error not being discovered by manually playing the game.

The system-specific bugs were discovered by us running the game as well as the testing suite on 3 most common operating systems: Windows, Linux, and Mac. In fact, we have discovered bugs that only happened on Windows, but never happened on Mac. However, those bugs were immediately addressed.

The modular architecture was also an advantage to the testing plan. A possible risk that we have considered is the possibility of a rendering error influencing the game logic or the AI bots. For instance, if the cooldown for Player 1, ability 1 is not properly displayed on the screen due to a potential bug, the propagation of such an error to the logic/AI component must be avoided at all costs. In order to achieve this goal, we have separated the game logic from the rendering component. Thus, if an error happened inside the rendering component, it was caught there, and the game logic will not be affected in any way.

## 3. Tech Stack

The Python and Java communication was a risky part of our project. Having researched multiple options, we have decided for a websocket connection and automatic Python file execution by Jython. However, Jython's version is different from the newest Python versions, and does not support many advanced features. In order to mitigate the impact of  that, we have decided to devise a plan for external execution of Python scripts. Our Java code can wait for an external agent to connect rather than crash. This has proven useful when we found out in the middle of the project that Jython does not support Tensorflow, which we did not predict initially.

In addition to the above mentioned risks, the user might not have Python installed on their computer or their version might be too old, which is likely to cause issues with AI bots. For this reason, in the first page of the Help screen which can be accessed via the main menu, the user is told that he must have at least Python version 3 installed on their system in order for the AI bots to work smoothly. As the ML bots require execution of an additional script, various errors connected to installation of libraries and script execution were addressed by adding a guide in a .txt file.

## 4. Artificial Intelligence

Our AI was a considerably risky part of the project as well for the reasons below:

1. We did not have a substantial background in game AI before.
2. Our game was novel, so we were not able to mimic any architecture 1-to-1.
3. We wanted to make multiple bots, which was time-consuming.
4. We did not have a guarantee of the effectiveness of the algorithms in our particular game.

Our plan for approaching it carefully was:

1. Researching many architectures, and selecting a redundant number of potential candidates.
2. Developing the Minimum Viable Product without bots very rapidly to allow time for AI experimentation.
3. Starting from the easiest, established bot - AStar with additional deterministic mechanic enhancements (e.g. randomisation when stuck, teleport usage ability, enhanced speed by *a priori* exclusion of stochastically improbable results). That way our game was guaranteed to have at least one smart, playable bot.
4. Parallel development of 2 further architectures to avoid the risk of starting with a less promising one.
5. Splitting training between team members, and training multiple instances to compare results.
6. Treating the further architectures as a research task, and focusing on improvements and potential justifications of their behaviour.

This approach mitigated most of the aforementioned risks. Many problems did in fact occur, for instance:

- The A* development took longer than expected because of data parsing difficulties -> the timeline allowed for that thanks to Rapid Application Development.
- The Q-learning bot was very slow to train even after simplifying the game and applying State Similarity Measure -> we have trained it with the help of many

people at once, and the team was working on another architecture in the meantime.
- There was a need for other architecture investigation -> a uniform architecture in Python enabled to rapidly develop an Actor-Critic architecture, with almost only the actual Keras model changes.
- Bots had a strong preference to use the teleports -> we had changed the reward scaling from linear to rational (scaled by distance to enemy).

Overall, despite many challenges which we have faced throughout this project, we have managed to fulfil all our key requirements thanks to adapting our plan to unexpected scenarios.

# VII. Evaluation

## 1. General evaluation

Our group can state that the project has been completed successfully. Throughout the game's creation, we have usually had a status of 'GREEN' with our teaching assistant, indicating that our project is running smoothly. However, we also encountered situations where we had to rearrange our project's timeline to keep it realistic, or to adjust the expectations based on the performance of its different versions. We will now take a closer look at the strengths and weaknesses of our project.

## 2. Strengths

Our team showed exceptional passion for the subject, and shared a common goal of learning as much as we can as well as delivering the best results possible.

We usually exhibited excellent communication skills. We used Discord for all our communication, and we had dedicated text channels for announcements, general conversations, external resources, and AI discussion. In our first meeting, we discussed the desired game type and in the subsequent meeting, we finalised the game idea.

We had weekly meetings during which discussed the current status of our tasks alongside the tasks for next week.  After the meeting, minutes were available in the channel. Our team was also flexible in rearranging the meetings as necessary due to time differences or any other inconveniences.

Furthermore, our group managed to divide tasks to utilise each member's skills. We had a dedicated team for the user interface, the audio, the map, the graphics, and other standalone components. We also managed to communicate with one another often to offer assistance, require information, or check up on their progress. We also switched up the pairs for *Pair Programming* to allow everyone to work on various aspects of the project.

Additionally, some team members were determined to rejoin the project after setbacks, which means that even periods of lesser performance did not exclude them from the project entirely.

We had some dedicated members who exhibited exceptional knowledge in the subject they were mainly working on, which allowed us to aim for the best when it comes to the realisation and design of certain components. For instance, the AI theory we have implemented has required extensive extracurricular knowledge, which was constantly improved upon. The UI had a coherent, user-friendly design utilising numerous features of JavaFX to the fullest, and the networking architecture was only possible thanks to strong programming skills of our core team members and their background knowledge about the topic.

## 3. Weaknesses

The biggest weakness of our team was our ambition. We planned on creating four different bots with four different AI algorithms, respectively. We have planned alternative approaches to this, however, when one of the algorithms had unsatisfactory performance, we kept pursuing other options to deliver a top quality bot while still learning more about reinforcement learning-based algorithms. Hence, our project's intensity, especially during the initial and final weeks, was immense.

Additionally, some of our team members fail to attend some biweekly meetings due to unforeseen circumstances, causing a communication hold-up regarding the parts of the project that the person was working on.

Another topic of concern for our group was the planning aspect. Since our project was ambitious, as mentioned earlier, we had minimal time to plan the project and went with a Rapid Application Development approach. The RAD approach is more complex to manage than other software engineering models, but it was necessary provided we wanted to maintain a robust game architecture and gain time for AI development.

Moreover, the AI architectures, although researched carefully, required a lot of experimentation due to their statistical nature and novel character of our game. This meant that the bots, although exhibiting noticeable patterns such as hiding behind obstacles, using teleports when necessary, or hiding in the same position as A*, still did not exceed the performance of an A*-based bot or a good-to-average human player. This was however somewhat expected as the A* is the optimal algorithm for pathfinding-based games.

Another desirable feature for improving our project would be performing a more extensive quantitative analysis of the different AI architectures, comparing different numbers of layers, reward function scalings, and Keras optimisers.

## 4. What have we learnt?

Importantly, we learnt that teamwork is one of the fundamental aspects of working on a big project. We were lucky enough to be part of a passionate  team whose members  communicated and were willing to help one another. We learnt  that communication is vital, even if it means admitting to slacking off or not knowing how to do an assigned task.

We also learnt how to utilise other people's skills, and found out that the only way to alleviate too much workload in case of unexpected issues is to ask another team member for help.

We have also learnt the importance of thoughtful design of game components, as each part of the game was directly dependent on previously developed components. Through this realisation, we have gained significant experience in software design, and technical knowledge about the components.

We have gained significant theoretical and practical knowledge of many technologies and topics. Among the elements, some we have worked with for the first time are:

- AI architectures and Tensorflow
- Python testing suites
- JavaFX UI and media
- Game physics
- Jython and interfacing between languages

This enabled us to improve our technical skills, and prove our adaptability in solution design.

Lastly, we have made significant knowledge progress when refining the components, and realised the importance of backwards-compatibility and incremental knowledge gain. As the topics we have worked on and the technologies we used were not easy, we valued small insights that came up during every meeting. We also gained the skills required to evaluate a project and decide its direction rather than just fulfil a predetermined task.

## 5. What could we have done better?

- Usage of Software Design Tools

  Since most of our communication and planning was through Discord, it might have been helpful to utilise planning software such as Trello for our project. Although Discord is an excellent communication tool, visualising tasks assigned to a specific person can be difficult. Therefore, planning and assigning tasks using Trello could have been beneficial for our project.

- Meeting time and communication efficiency

  During our initial stages of the project, our weekly meetings sometimes ran for more than an hour, and key points were not mentioned until towards the end of the meeting. Luckily, we improved this by focusing more on immediately getting to the points that need discussion. This could have been done earlier to avoid wasting time.

- <u>Responsibility Disproportion</u>

  At some stages, the workload of some team members was larger than the others. This was not only because of different levels of meticulousness and ambition, but also due to insufficient communication of unexpected flaws and a reluctance to give more work to other people. This was not ill-intentioned, but at times, even though some people were requesting more work, others believed it is their responsibility to finish their part, perhaps due to fear of burdening others or lack of trust in their abilities. This got better in the later stages of the project as well, but could still be improved upon.

- <u>AI time dedication</u>

  Should the best 2 architectures be predetermined at the very beginning, we could have saved time for slower application and bot development, or testing out more parameter options. However, given that our knowledge was gained alongside the project and that many AI topics do not have a single, fixed solution, we believe we got very far with our outcomes and that the time dedicated to making this happen was mostly worth it. That said, ideally we would have come with established previous knowledge so that we can focus on implementation at a slightly slower pace.

# VIII. Summary

We are very proud of our outcomes, and of the knowledge or skills gained by each of our team members. The project was for us a valuable experience with respect to teamwork, programming practice, and theory learnt alike. The process was challenging but engaging, from the beginning to the end. Below we will give a brief summary of our project workflow.

When we started with our project, we first had to discuss the type of game we were going to create. Once we knew we wanted to create something similar to 'Hot-Potato', we started thinking about making the Artificial Intelligence aspect of the game fun and exciting for the players.

We tried working out the game requirements (both functional and non-functional) using the MOSCOW analysis method to establish our game's general idea. To outline our software architecture, we also created a high-level class diagram to help analyse how we will approach our coding. We then agreed that the next step would be to focus on the UI of the game, and that is where we implemented various Human-Computer Interaction principles to keep the UI aesthetic and consistent.

Since many of the above took place in the first few weeks of the project, it is understandable that many of them needed modification in the following weeks. Luckily, we implemented the Agile software development methodology, allowing us to update requirements as necessary. As indicated in *The Manifesto for Agile Software Development*, we followed several principles to allow our project to produce the best product possible. Since agile development is flexible, our initial planning was only a rough guideline to plan our game's approach. We also implemented a Rapid Application Development methodology to break down our tasks into smaller components and reduce risk. Additionally, continually integrating our code allowed our master branch to always have a working product.

More often than not, throughout the entirety of our project, it went smoothly with minor unpredictable setbacks. Even though it was difficult having to work together with people, having never met them before, our team's communication skills benefitted us, and each of our contribution to the group made the game what it is - a success. We are incredibly proud of ourselves and excited for other people to try out WannaCry. We also plan on developing the Machine Learning bots to the degree where they will be able to pose a real challenge to the A* bot, by experimentation with the network architecture after the formal end of the project.

# IX. Teamwork Analysis

To evaluate our group's teamwork throughout the duration of this project, we have decided to consider a variety of factors.

## Contributions and Attitude

Our team routinely offers constructive criticism that ping off one another's ideas. Every time we have a meeting, most of our members always show up with a positive attitude.

## Cooperation with Others

Every one of us would check to make sure everyone is doing an equal amount of work. Throughout the project, all of us were highly productive. Sometimes, if one of us is unable to work on his/her part one week, that person would always pick up their slack. We also work extremely well with each other since we work on certain parts of the project in sub-groups that constantly change.

## Team Role Fulfillment

We would have two meetings every week and most of us would participate in all group meetings. Even if, for some reason, someone is unable to make it to the meeting, they would let the others know the reason for their absence. We also segregate tasks during the weekly meetings and everyone does the work that is assigned by the group.

## Focus

During our weekly meetings, we would also check up on how well our team is working together. All of us are almost always focused on the task that we are responsible for.

## Ability to Communicate

Our team always listens to, shares with, and supports the efforts of others. That is why our weekly meetings are usually over an hour long. We also tend to provide each other with effective feedback, which relays a lot of relevant information on the direction of the project.

## Accuracy

Our game is complete, well-organized, error-free, and done on time. Our final presentation as well as documentation is also thorough.

# X. Appendix

## Coding Standards

### Naming

As a general rule the team named everything after their utility and function. This was done with the purpose of maximising the readability of the code. Abbreviations were not prioritized over legibility.

- Packages: in camel case e.g *com.packageNamePurpose*
- Classes: in camel case e.g. *className*
- Methods: in camel case e.g. *descriptiveMethodUtility()*
    - Boolean checks as: *isBooleanTrue*()
- Variables: in camel case e.g. *nameOfVariable*
- Constants: in uppercase spaced by underscores e.g. *NAME_OF_CONSTANT*
- Text Files: in camel case e.g. *fileName* OR in uppercase spaced by underscores e.g. *file_Name*
- Images and Audio: in uppercase spaced by underscores e.g. *image_name* OR in camel case e.g. *imageName*

Where possible, the team avoided using temporary/disposable names such as e.g. *temp, vari, i*... etc.

### Indentation and Spacing

- No line of code exceeds 80 spaces (approximately). Otherwise, the code is broken into a new line and indented as necessary.
- Every indentation is 4 spaces long.

### Documentation

- Every method used has JavaDocs added to it
- Complex Variables also have JavaDocs
- InLine Comment: in line comments are provided where otherwise the purpose of something is not clear.
- Block commenting: Clock commenting in provided at the beginning of the code being commented.
- TODOs: Added where appropriate with thorough description of what is to be done.
- Javadocs, where appropriate these are always provided:
    - General description of what the method does, how to use it.

  - @params name - description
  - @return name type description

## General

- If statements vs Switches: Where possible switch statements where used over simple ifs. However, where if statements are used, they follow this bracketing convention:
```
if (condition){
      code block;
}
```
- Iterations: All iterations are bracketed as follows:
```
iterator (statement){
      code block;
}
```
- The team used *Ratliff Style* for bracketing.
- Boolean methods:  as stated above statements prioritize returning a true statement.
- Declarations: same type variables were declared in different lines over inline
```
int varX; //Rather than int varX,varY;
int varY;
```
- Initialisation of Variables: the team prioritized the declaration of variables at the place in code where they are used rather than at the beginning of the method.
- Annotations when implementing or extending a class are always used. E.g. @Override
- Using try and catch statements over thrown to handle possible errors that could arise.

## UML Sub Systems

### Game Initialisation Sub System

The diagram below demonstrates the sub system to which the application initializes the game. GameExecuteController works regardless of UI and is run through the terminal.

GameViewController, on the other hand, initiates the game with a pleasant interface for the game to be played at.

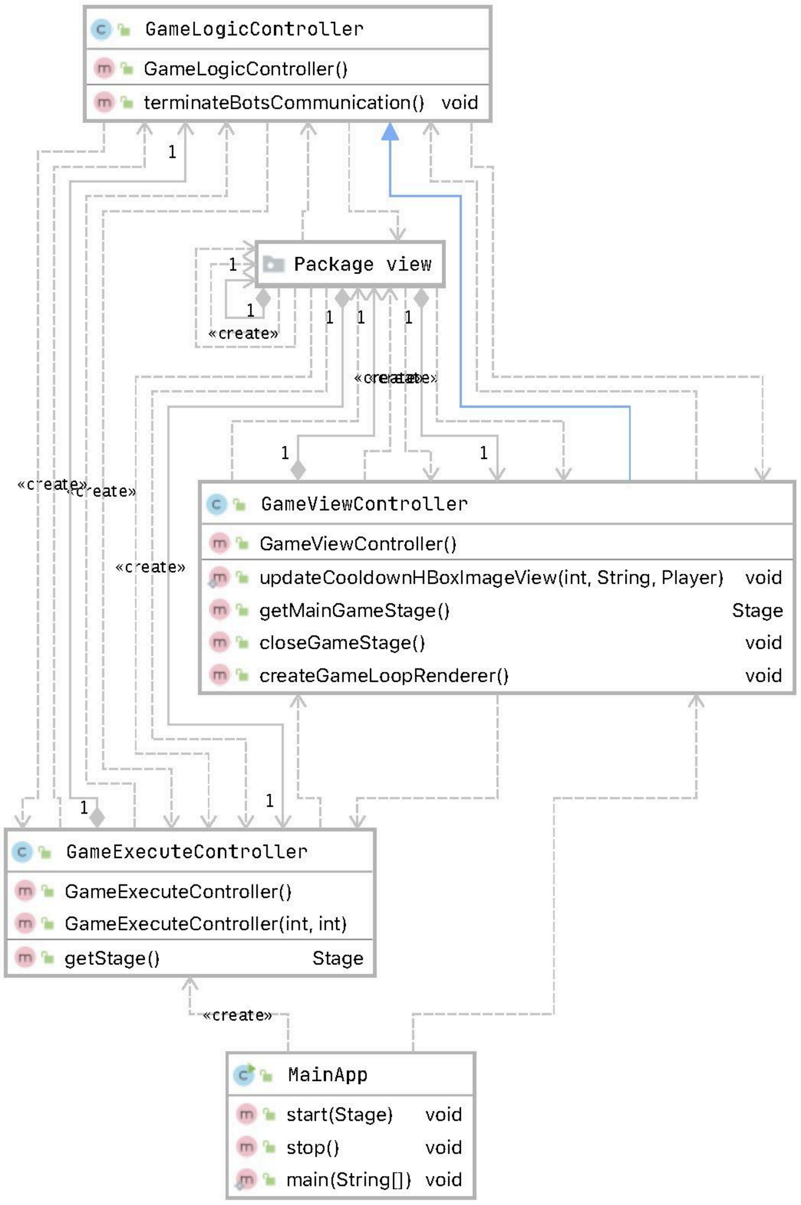
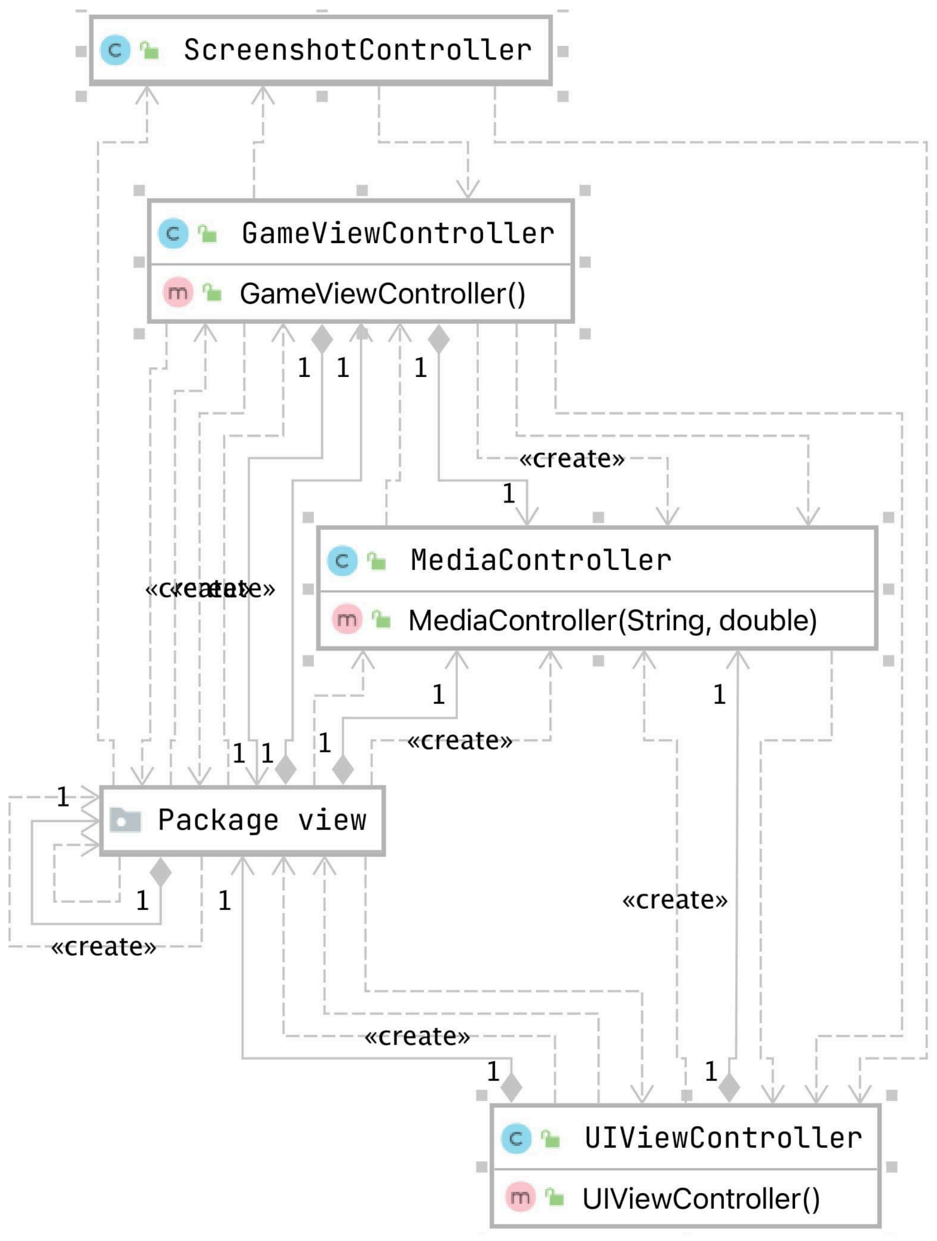


Figure 9.1 - Game Initialisation SubSystem

## User Interface Subsystem

The User Interface subsystem handles all the menus and submenus created through the life cycle of the running app.



## Game Loop Thread SubSystem

This subsystem handles the game logic. It runs a thread independent of interface and essentially creates the continuous loop of a running game session.

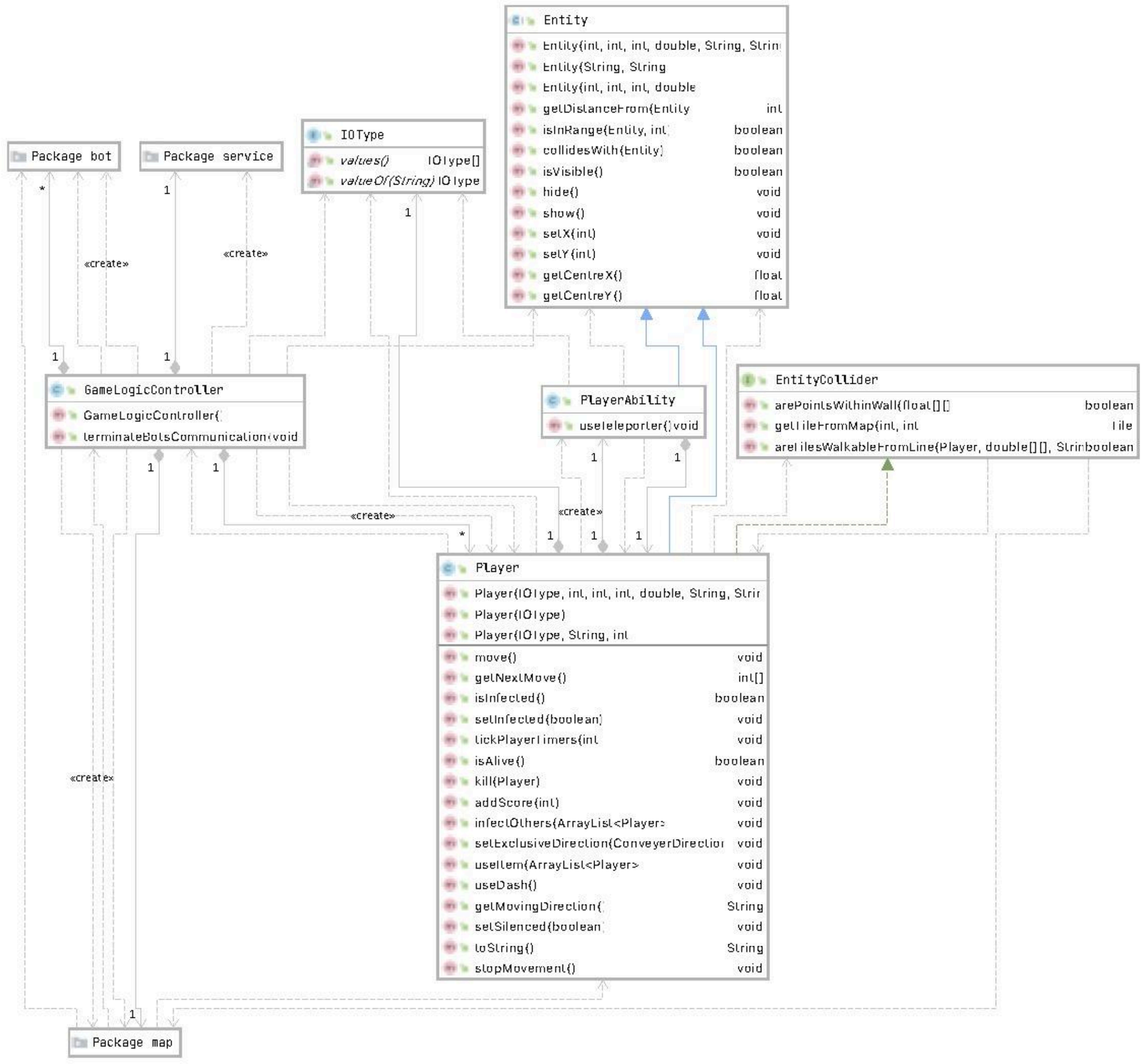

Entity
Entity(int, int, int, double, String, Strin
Entity(String, String
Entity(int, int, int, double
getDistanceFrom(Entity int
isInRange(Entity, int boolean
collidesWith(Entity) boolean
isVisible() boolean
hide() void
show() void
setX(int) void
setY(int) void
getCentreX() float
getCentreY() float
Package bot
Package service
IOType
values() IOType[]
valueOf(String) IOType
«create»
GameLogicController
GameLogicController(
terminateBotsCommunication void
PlayerAbility
useTeleporter() void
EntityCollider
arePointsWithinWall(float[][ boolean
getTileFromMap(int, int Tile
areTilesWalkableFromLine(Player, double[][], Strin boolean
Player
Player(IOType, int, int, int, double, String, Strir
Player(IOType)
Player(IOType, String, int
move() void
getNextMove() int[]
isInfected() boolean
setInfected(boolean) void
tickPlayerTimers(int void
isAlive() boolean
kill(Player) void
addScore(int) void
infectOthers(ArrayList<Player> void
setExclusiveDirection(ConveyerDirectio void
useItem(ArrayList<Player> void
useDash() void
getMovingDirection( String
setSilenced(boolean void
toString() String
stopMovement() void
Package map

Figure 9.3 - Game Loop Thread SubSystem

## Game Rendering SubSystem

Akin to the Game Loop Thread SubSystem, this System handles the rendering of the game. However it is only executed if called through the GameViewController. Either way this system has no effect on the logic loop.

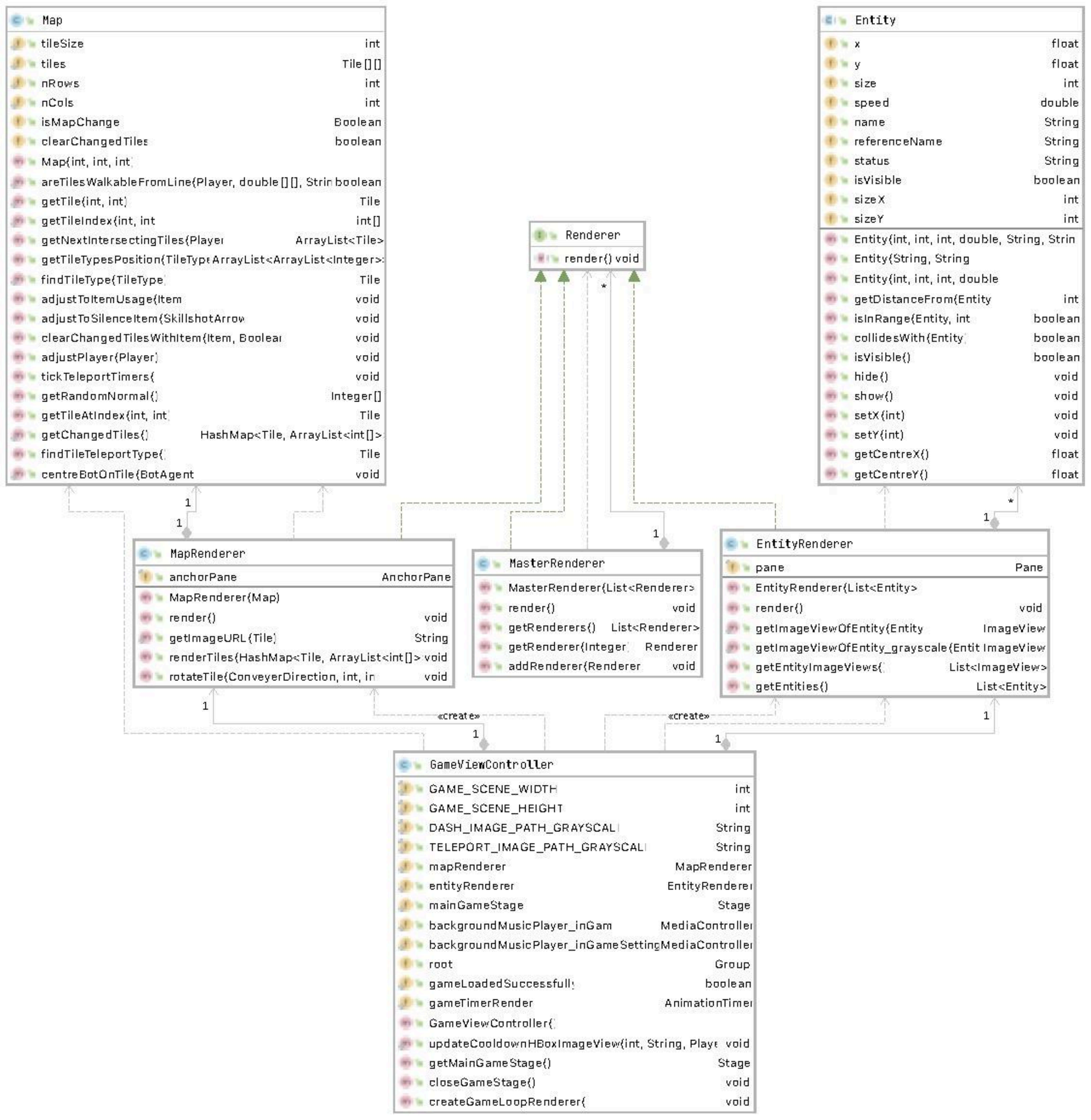

Map
tileSize int
tiles Tile[][]
nRows int
nCols int
isMapChange Boolean
clearChangedTiles boolean
Map(int, int, int
areTilesWalkableFromLine(Player, double[][], Strin boolean
getTile(int, int) Tile
getTileIndex(int, int int[]
getNextIntersectingTiles(Player ArrayList<Tile>
getTileTypesPosition(TileType ArrayList<ArrayList<Integer>
findTileType(TileType Tile
adjustToItemUsage(Item void
adjustToSilenceItem(SkillshotArrow void
clearChangedTilesWithItem(Item, Boolean void
adjustPlayer(Player) void
tickTeleportTimers( void
getRandomNormal() Integer[]
getTileAtIndex(int, int Tile
getChangedTiles() HashMap<Tile, ArrayList<int[]>
findTileTeleportType( Tile
centreBotOnTile(BotAgent void
Entity
x float
y float
size int
speed double
name String
referenceName String
status String
isVisible boolean
sizeX int
sizeY int
Entity(int, int, int, double, String, Strin
Entity(String, String
Entity(int, int, int, double
getDistanceFrom(Entity int
isInRange(Entity, int boolean
collidesWith(Entity boolean
isVisible() boolean
hide() void
show() void
setX(int) void
setY(int) void
getCentreX() float
getCentreY() float
Renderer
render() void
*
1
1
1
*
1
MapRenderer
anchorPane AnchorPane
MapRenderer(Map)
render() void
getImageURL(Tile) String
renderTiles(HashMap<Tile, ArrayList<int[]> void
rotateTile(ConveyerDirection, int, in void
MasterRenderer
MasterRenderer(List<Renderer>
render() void
getRenderers() List<Renderer>
getRenderer(Integer Renderer
addRenderer(Renderer void
EntityRenderer
pane Pane
EntityRenderer(List<Entity>
render() void
getImageViewOfEntity(Entity ImageView
getImageViewOfEntity_grayscale(Entit ImageView
getEntityImageViews( List<ImageView>
getEntities() List<Entity>
1
«create»
1
«create»
1
1
GameViewController
GAME_SCENE_WIDTH int
GAME_SCENE_HEIGHT int
DASH_IMAGE_PATH_GRAYSCAL String
TELEPORT_IMAGE_PATH_GRAYSCAL String
mapRenderer MapRenderer
entityRenderer EntityRendere
mainGameStage Stage
backgroundMusicPlayer_inGam MediaControlle
backgroundMusicPlayer_inGameSetting MediaControlle
root Group
gameLoadedSuccessfull boolean
gameTimerRender AnimationTime
GameViewController(
updateCooldownHBoxImageView(int, String, Playe void
getMainGameStage() Stage
closeGameStage() void
createGameLoopRenderer( void

Figure 9.4 - Game Rendering SubSystem

## World Subsystem

This is the system in charge of hoarding a coherent world for the players to interact in.

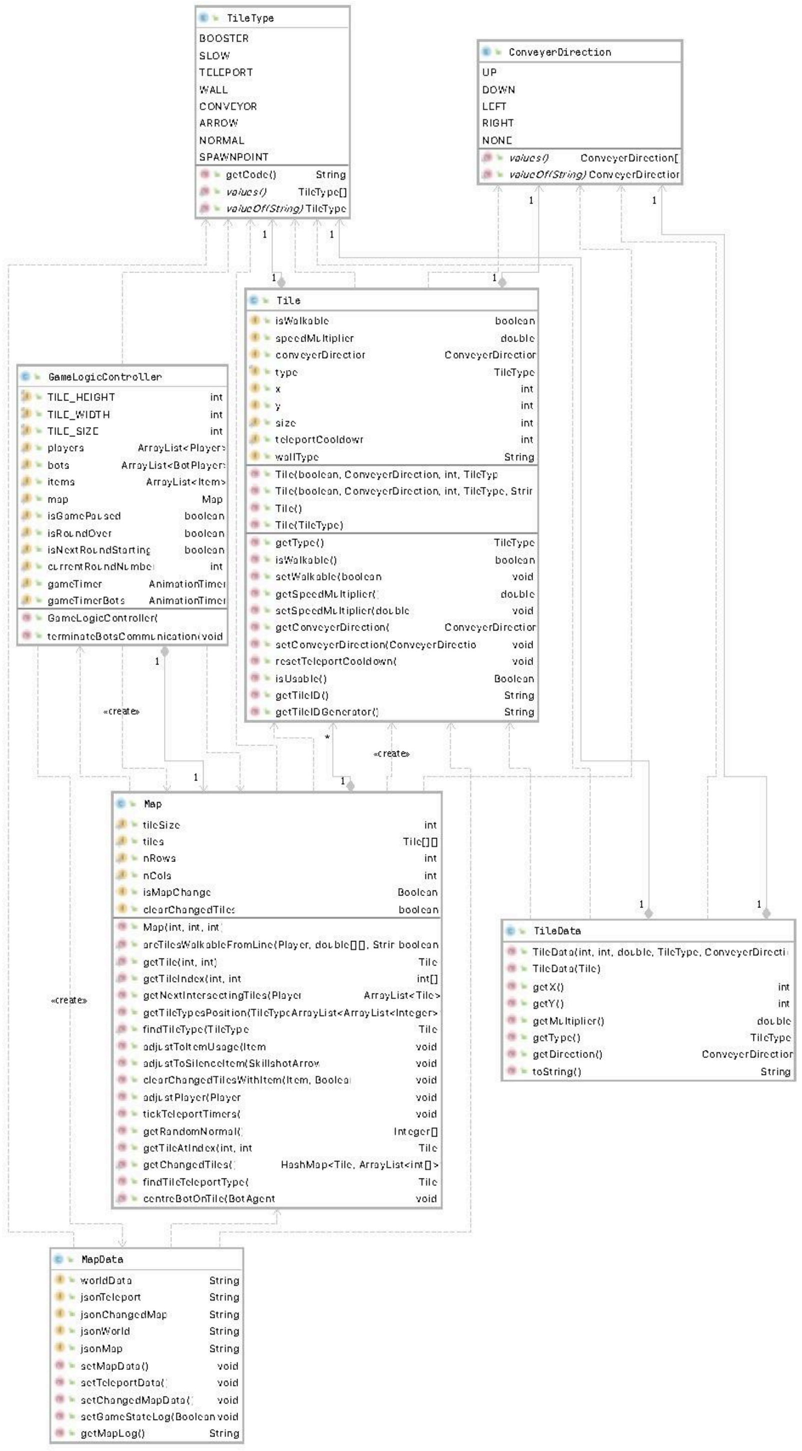

TileType
BOOSTER
SLOW
TELEPORT
WALL
CONVEYOR
ARROW
NORMAL
SPAWNPOINT
getCode() String
values() TileType[]
valueOf(String) TileType
ConveyorDirection
UP
DOWN
LEFT
RIGHT
NONE
values() ConveyorDirection[
valueOf(String) ConveyorDirection
Tile
isWalkable boolean
speedMultiplier double
conveyorDirection ConveyorDirection
type TileType
x int
y int
size int
teleportCooldown int
wallType String
Tile(boolean, ConveyorDirection, int, TileTyp
Tile(boolean, ConveyorDirection, int, TileType, Strir
Tile()
Tile(TileType)
getType() TileType
isWalkable() boolean
setWalkable(boolean void
getSpeedMultiplier( double
setSpeedMultiplier(double void
getConveyorDirection( ConveyorDirection
setConveyorDirection(ConveyorDirectio void
resetTeleportCooldown( void
isUsable() Boolean
getTileID() String
getTileIDGenerator() String
GameLogicController
TILE_HEIGHT int
TILE_WIDTH int
TILE_SIZE int
players ArrayList<Players
bots ArrayList<BotPlayers
items ArrayList<Item>
map Map
isGamePaused boolean
isRoundOver boolean
isNextRoundStarting boolean
currentRoundNumbe int
gameTimer AnimationTimer
gameTimerBots AnimationTimer
GameLogicController(
terminateBotsCommunication( void
Map
tileSize int
tiles Tile[][]
nRows int
nCols int
isMapChange Boolean
clearChangedTiles boolean
Map(int, int, int
areTilesWalkableFromLine(Player, double[][], Strir boolean
getTile(int, int) Tile
getTileIndex(int, int int[]
getNextIntersectingTiles(Player ArrayList<Tile>
getTileTypesPosition(TileType ArrayList<ArrayList<Integer>
findTileType(TileType Tile
adjustToItemUsage(Item void
adjustToSilenceItem(SkillshotArrow void
clearChangedTilesWithItem(Item, Boolea void
adjustPlayer(Player void
tickTeleportTimers( void
getRandomNormal( Integer[]
getTileAtIndex(int, int Tile
getChangedTiles( HashMap<Tile, ArrayList<int[]>
findTileTeleportType( Tile
centreBotOnTile(BotAgent void
TileData
TileData(int, int, double, TileType, ConveyorDirecti
TileData(Tile)
getX() int
getY() int
getMultiplier() double
getType() TileType
getDirection() ConveyorDirection
toString() String
MapData
worldData String
jsonTeleport String
jsonChangedMap String
jsonWorld String
jsonMap String
setMapData() void
setTeleportData( void
setChangedMapData( void
setGameStateLog(Boolean void
getMapLog() String
«create»
«create»
«create»
1
*

Figure 9.5 - World Creation and Initialisation SubSystem

## Networking SubSystem

This system handles the integrated TCP communication  between Python and Java. It obeys the Observer Pattern, with events defined as the instances the Java Server finds a Python Client. It then allocates that socket to a newly created BotCommunicationhandler

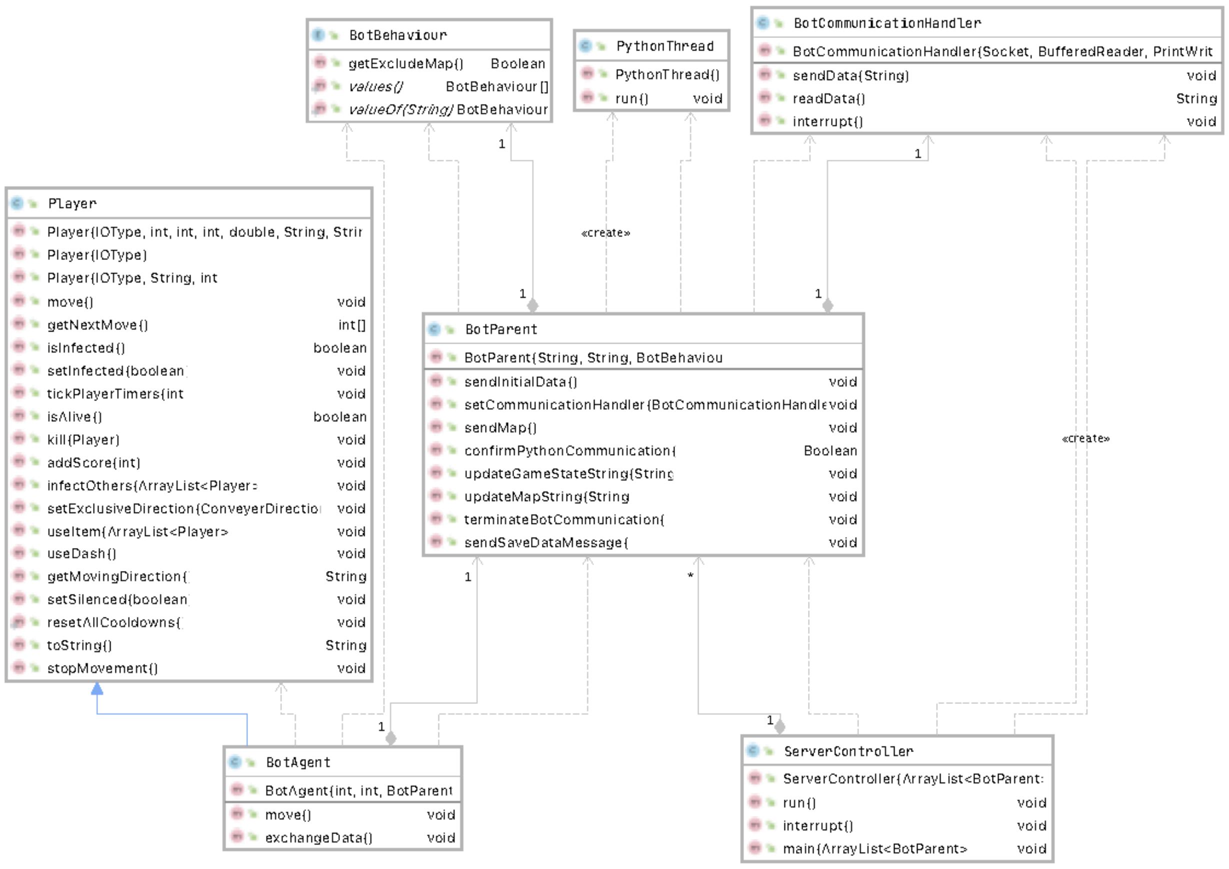

BotBehaviour
getExcludeMap() Boolean
values() BotBehaviour[]
valueOf(String) BotBehaviour
PythonThread
PythonThread()
run() void
BotCommunicationHandler
BotCommunicationHandler(Socket, BufferedReader, PrintWrit
sendData(String) void
readData() String
interrupt() void
Player
Player(IOType, int, int, int, double, String, Strir
Player(IOType)
Player(IOType, String, int
move() void
getNextMove() int[]
isInfected() boolean
setInfected(boolean void
tickPlayerTimers(int void
isAlive() boolean
kill(Player) void
addScore(int) void
infectOthers(ArrayList<Player> void
setExclusiveDirection(ConveyerDirectio void
useItem(ArrayList<Player> void
useDash() void
getMovingDirection( String
setSilenced(boolean void
resetAllCooldowns( void
toString() String
stopMovement() void
BotParent
BotParent(String, String, BotBehaviou
sendInitialData() void
setCommunicationHandler(BotCommunicationHandle void
sendMap() void
confirmPythonCommunication( Boolean
updateGameStateString(String void
updateMapString(String void
terminateBotCommunication( void
sendSaveDataMessage( void
BotAgent
BotAgent(int, int, BotParent
move() void
exchangeData() void
ServerController
ServerController(ArrayList<BotParent>
run() void
interrupt() void
main(ArrayList<BotParent> void
«create»
«create»
1
1
1
1
1
1
*
1

Figure 9.6 - Networking SubSystem

## Map Maker SubSystem

The Util SubSystem we used to aid us on map creation.

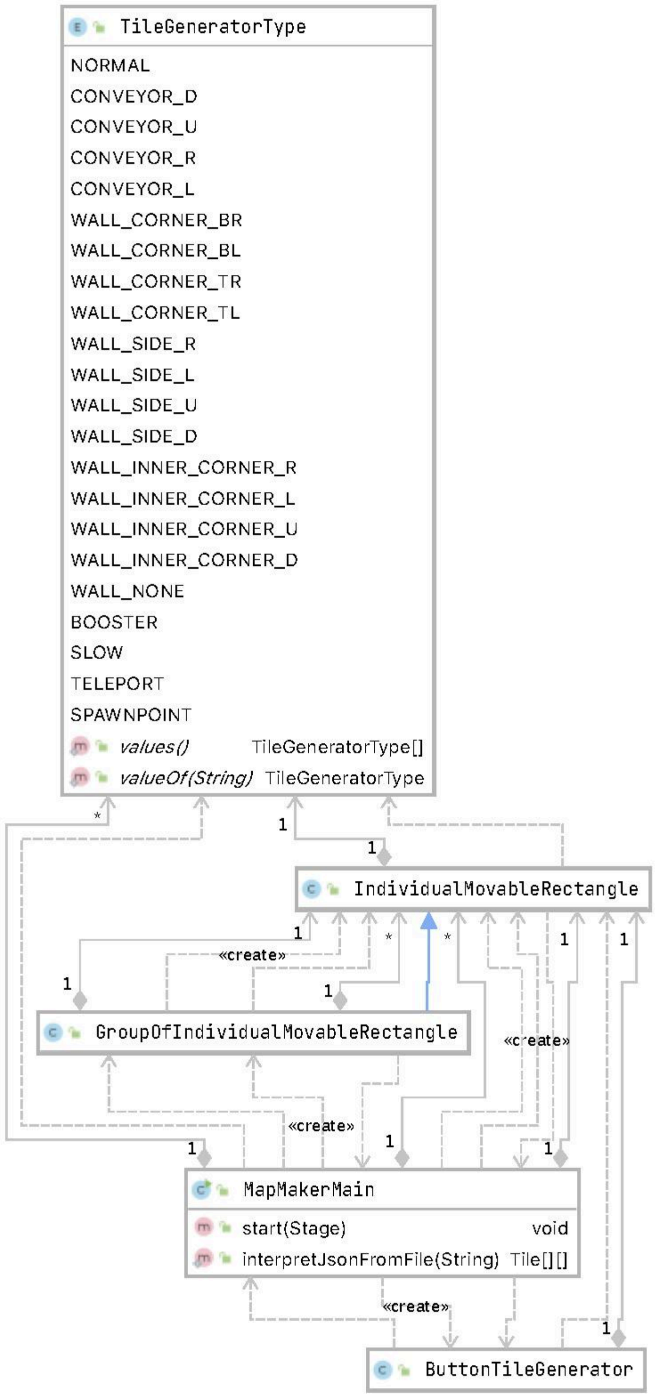


Figure 9.7 - Map Maker SubSystem

## Data Collector Sub System

The subsystem that determines the state of the game at any given time. It is used to send coherent data back to python.

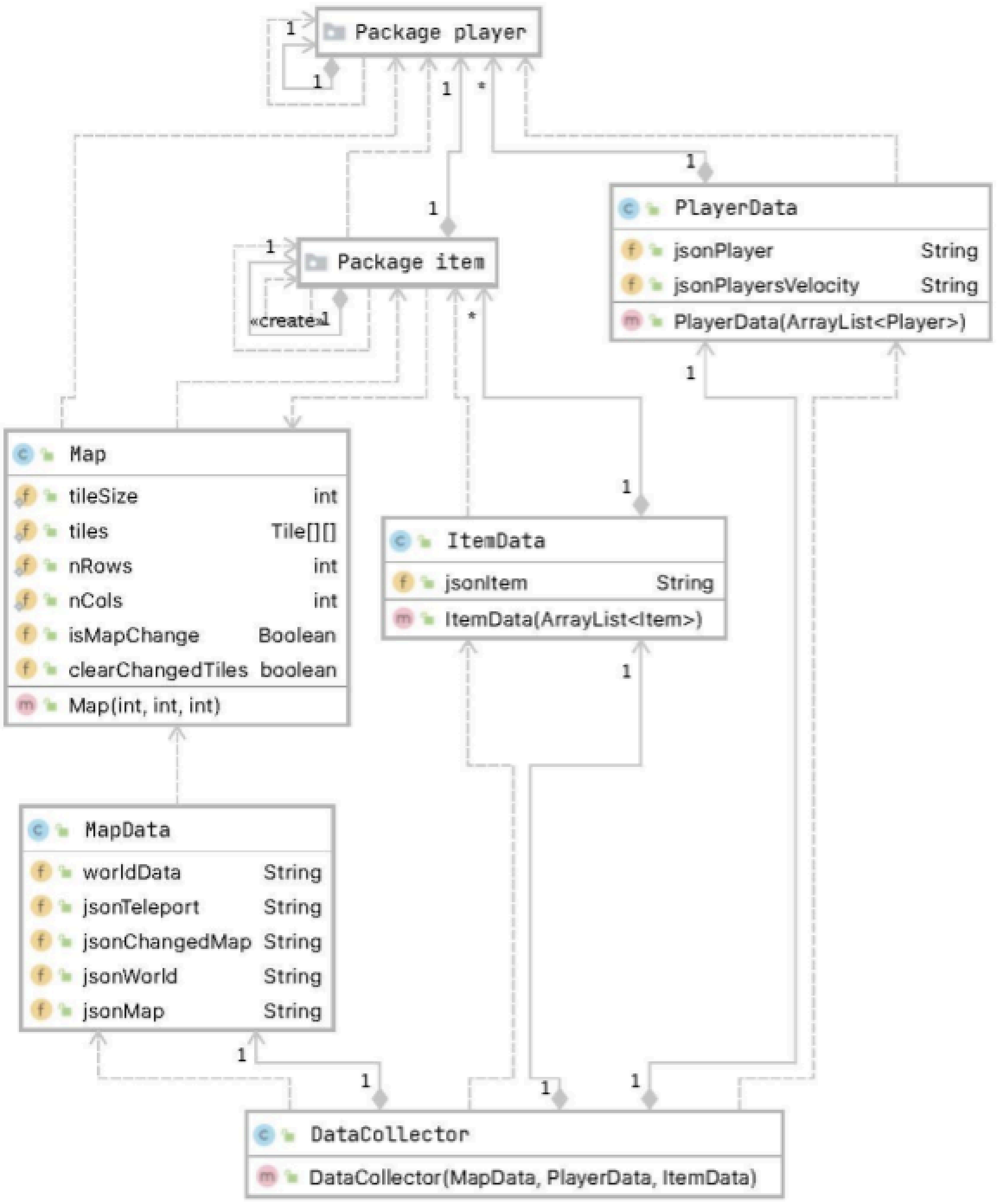


Figure 9.8 - Data Collector Sub System

## UI Subsystems

### Settings UI

The system in charge of initialising and  managing the various settings menus.

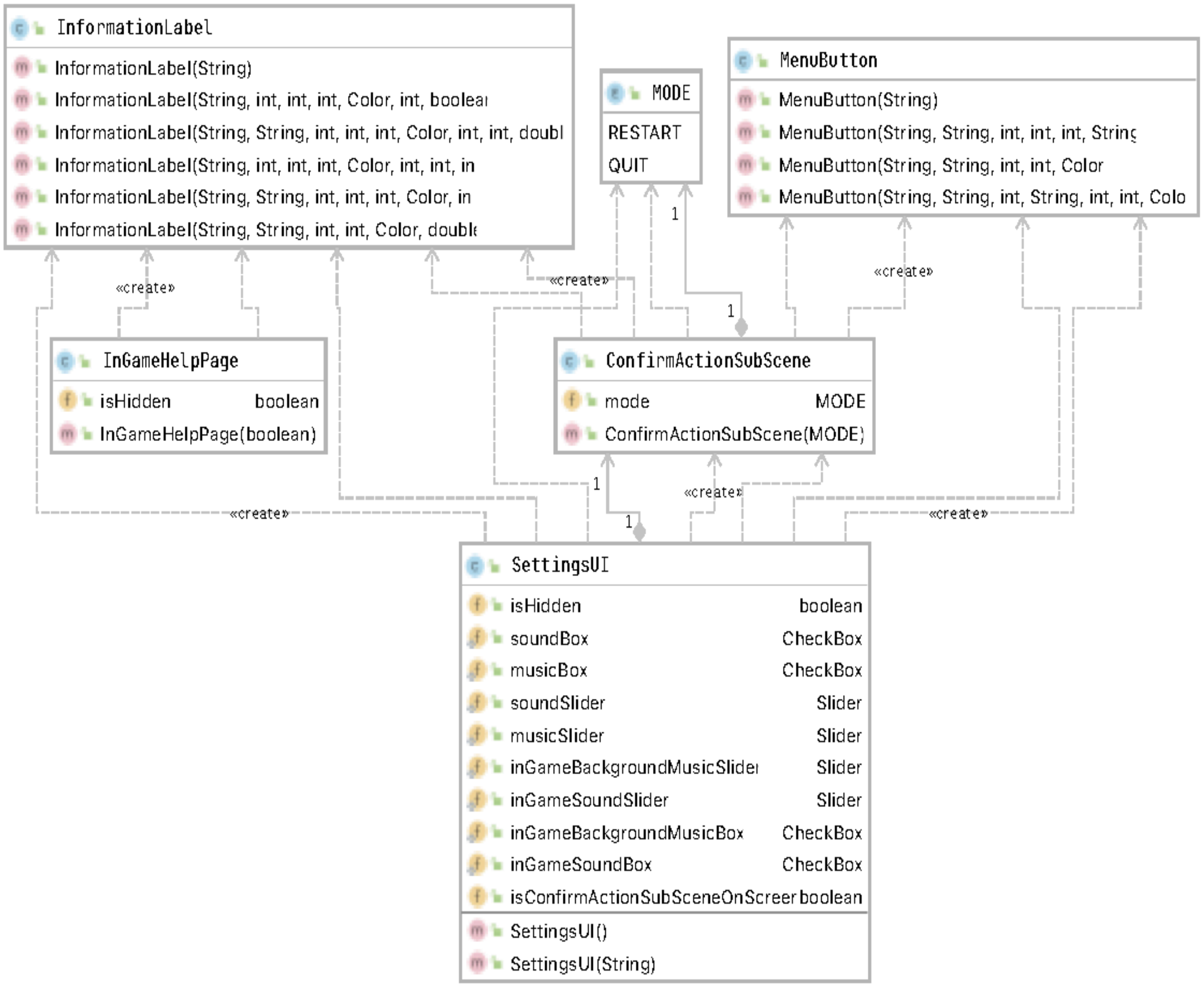

InformationLabel
InformationLabel(String)
InformationLabel(String, int, int, int, Color, int, booleai
InformationLabel(String, String, int, int, int, Color, int, int, doubl
InformationLabel(String, int, int, int, Color, int, int, in
InformationLabel(String, String, int, int, int, Color, in
InformationLabel(String, String, int, int, Color, double
MODE
RESTART
QUIT
MenuButton
MenuButton(String)
MenuButton(String, String, int, int, int, Strinç
MenuButton(String, String, int, int, Color
MenuButton(String, String, int, String, int, int, Colo
«create»
«create»
«create»
1
1
InGameHelpPage
isHidden boolean
InGameHelpPage(boolean)
ConfirmActionSubScene
mode MODE
ConfirmActionSubScene(MODE)
«create»
«create»
«create»
1
1
SettingsUI
isHidden boolean
soundBox CheckBox
musicBox CheckBox
soundSlider Slider
musicSlider Slider
inGameBackgroundMusicSlidei Slider
inGameSoundSlider Slider
inGameBackgroundMusicBox CheckBox
inGameSoundBox CheckBox
isConfirmActionSubSceneOnScreer boolean
SettingsUI()
SettingsUI(String)

Figure 9.9 - Settings UI

## General UI

These are the classes that manage all other general aspects of the UI. From providing the Loading Screen, Post Game results and recurring InGame Panels

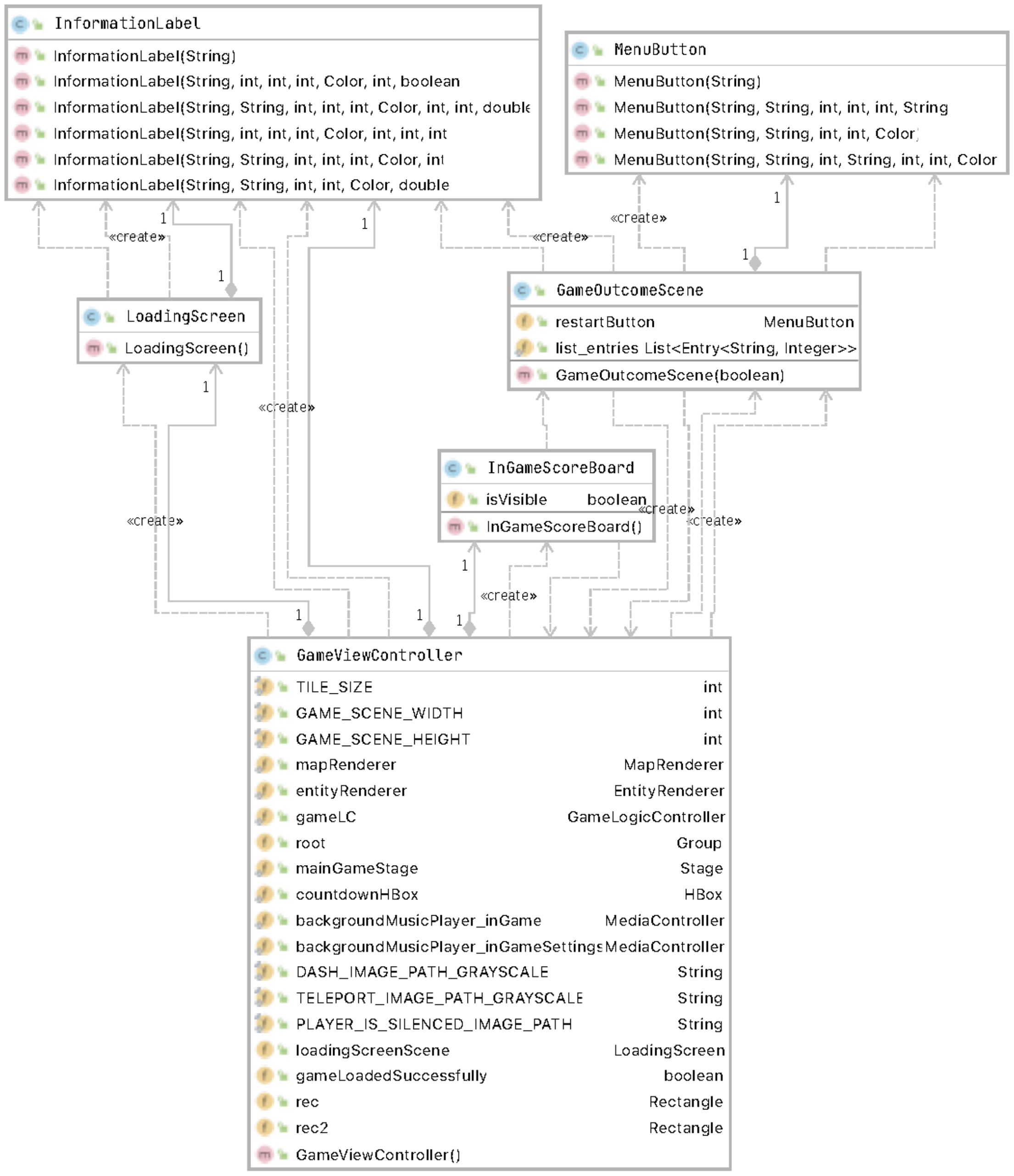

InformationLabel
InformationLabel(String)
InformationLabel(String, int, int, int, Color, int, boolean
InformationLabel(String, String, int, int, int, Color, int, int, double
InformationLabel(String, int, int, int, Color, int, int, int
InformationLabel(String, String, int, int, int, Color, int
InformationLabel(String, String, int, int, Color, double
MenuButton
MenuButton(String)
MenuButton(String, String, int, int, int, String
MenuButton(String, String, int, int, Color
MenuButton(String, String, int, String, int, int, Color
«create»
1
LoadingScreen
LoadingScreen()
GameOutcomeScene
restartButton MenuButton
list_entries List<Entry<String, Integer>>
GameOutcomeScene(boolean)
InGameScoreBoard
isVisible boolean
InGameScoreBoard()
GameViewController
TILE_SIZE int
GAME_SCENE_WIDTH int
GAME_SCENE_HEIGHT int
mapRenderer MapRenderer
entityRenderer EntityRenderer
gameLC GameLogicController
root Group
mainGameStage Stage
countdownHBox HBox
backgroundMusicPlayer_inGame MediaController
backgroundMusicPlayer_inGameSettings MediaController
DASH_IMAGE_PATH_GRAYSCALE String
TELEPORT_IMAGE_PATH_GRAYSCALE String
PLAYER_IS_SILENCED_IMAGE_PATH String
loadingScreenScene LoadingScreen
gameLoadedSuccessfully boolean
rec Rectangle
rec2 Rectangle
GameViewController()

Figure 9.10 - General UI

## Testing

Class Diagrams for the various Testing done throughout the development of the game.

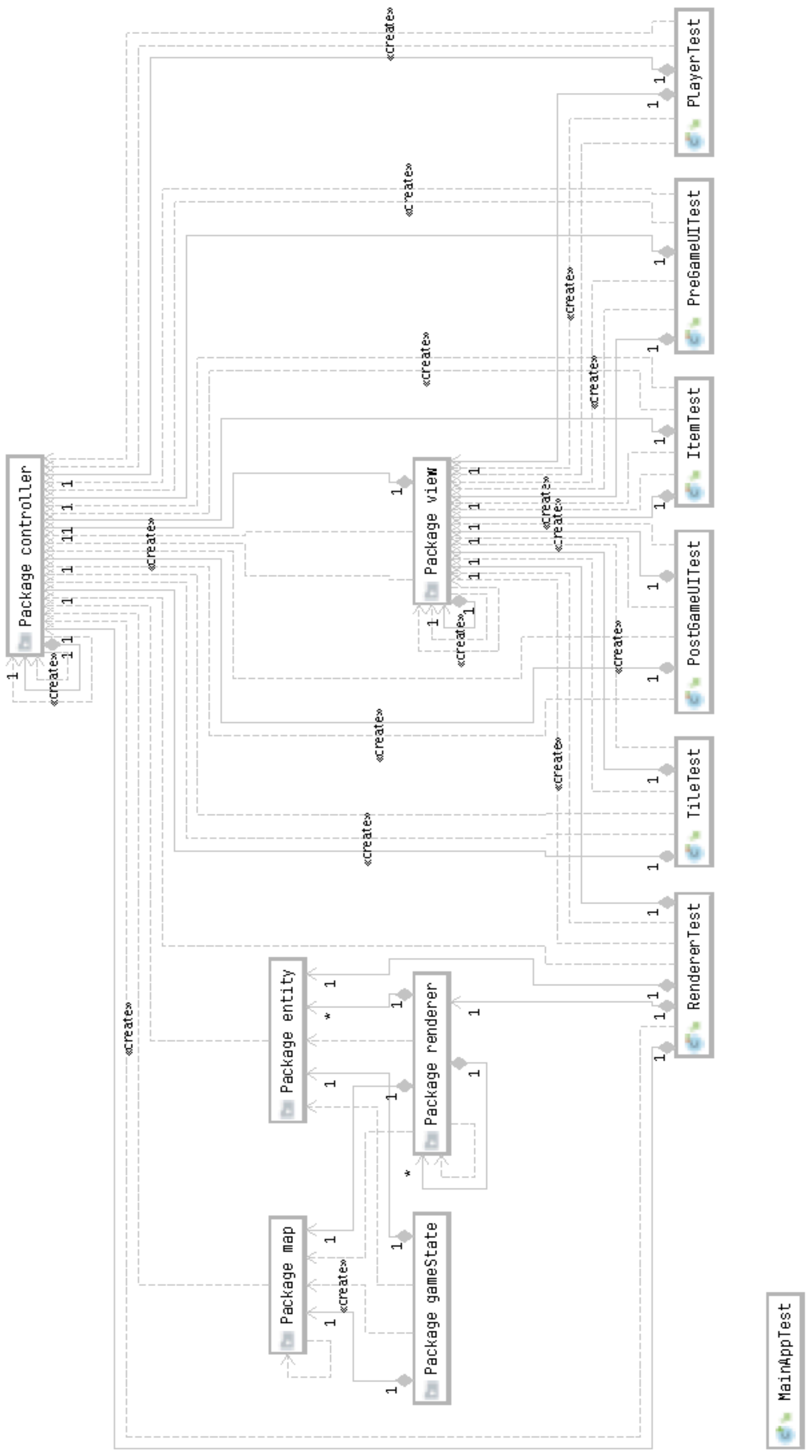


Figure 9.11 - Testing

## RenderTest SubSystem

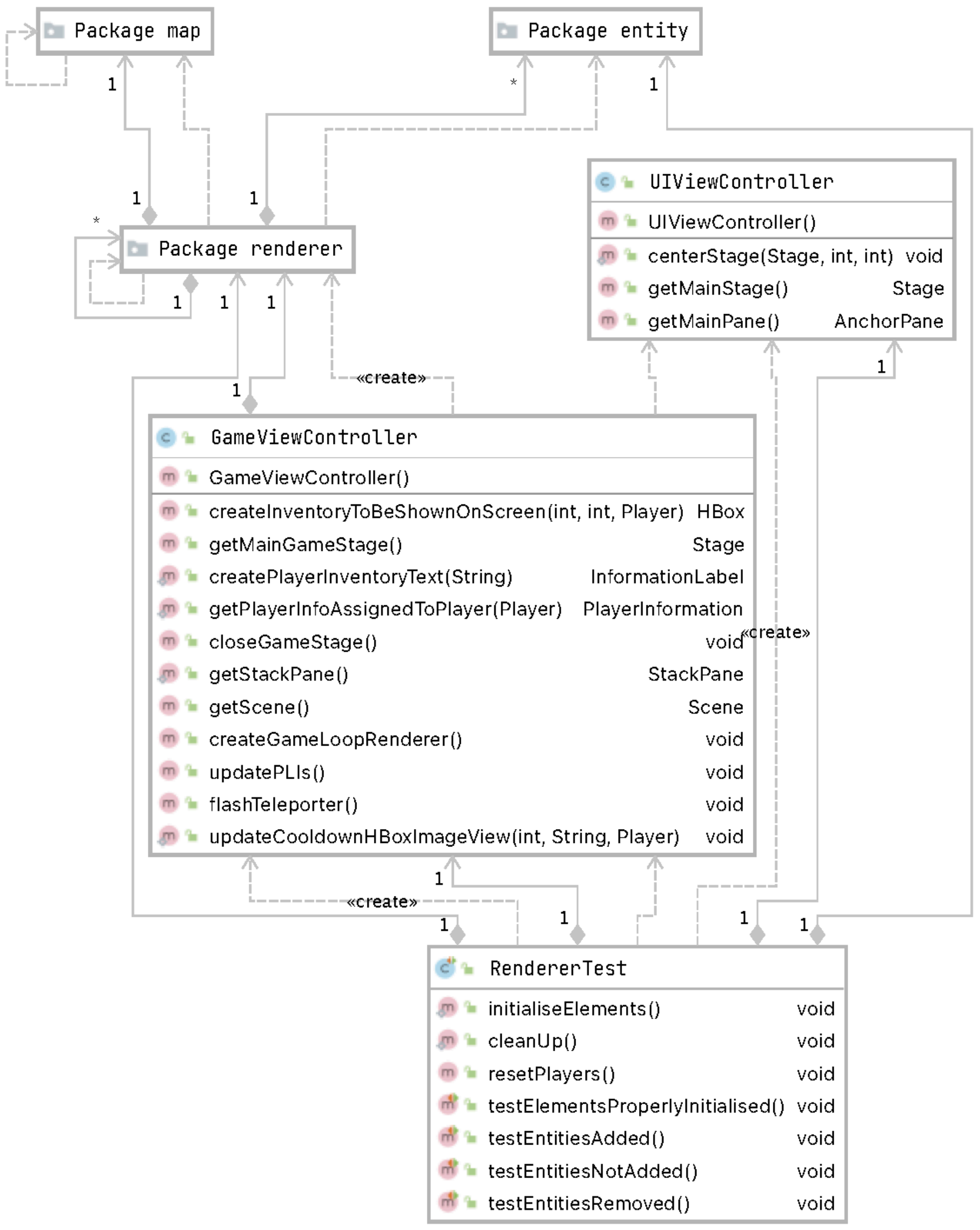

Package map
Package entity
Package renderer
UIViewController
UIViewController()
centerStage(Stage, int, int) void
getMainStage() Stage
getMainPane() AnchorPane
GameViewController
GameViewController()
createInventoryToBeShownOnScreen(int, int, Player) HBox
getMainGameStage() Stage
createPlayerInventoryText(String) InformationLabel
getPlayerInfoAssignedToPlayer(Player) PlayerInformation
closeGameStage() void
getStackPane() StackPane
getScene() Scene
createGameLoopRenderer() void
updatePLIs() void
flashTeleporter() void
updateCooldownHBoxImageView(int, String, Player) void
RendererTest
initialiseElements() void
cleanUp() void
resetPlayers() void
testElementsProperlyInitialised() void
testEntitiesAdded() void
testEntitiesNotAdded() void
testEntitiesRemoved() void
«create»
«create»
«create»
1
*

Figure 9.12 - RenderTest SubSystem

## TileTest SubSystem

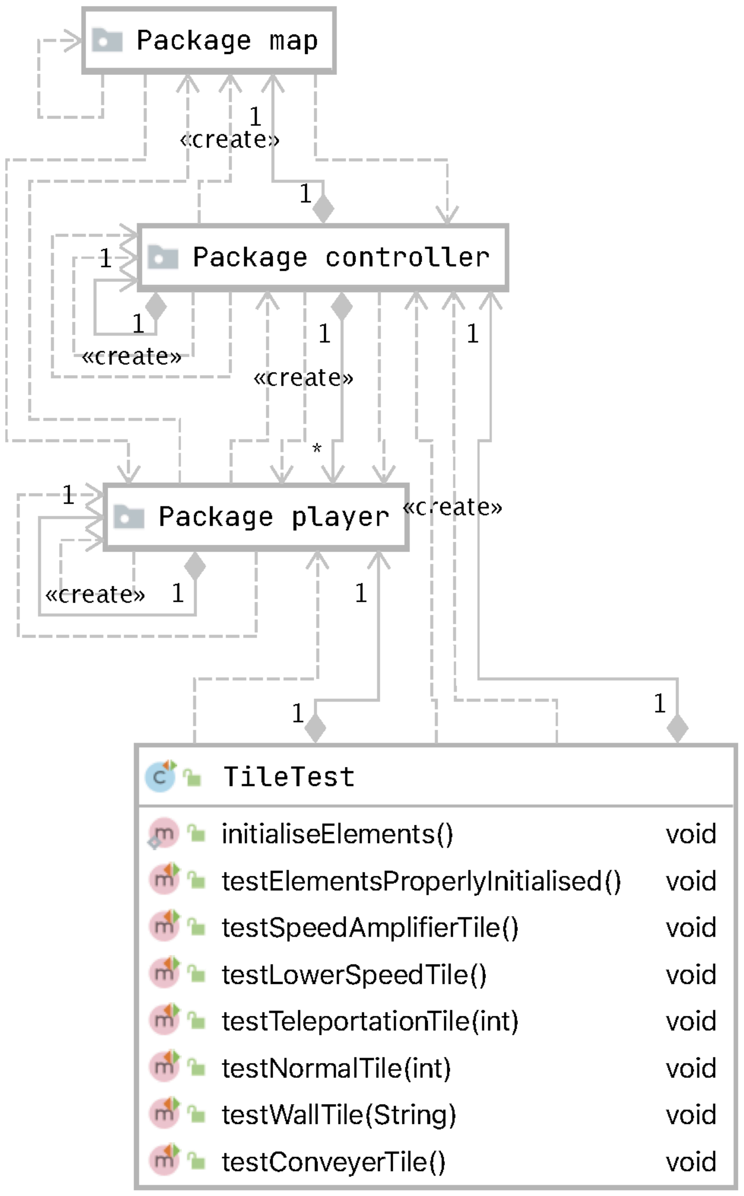

Package map
Package controller
Package player
«create»
«create»
«create»
«create»
«create»
1
1
1
1
1
1
1
1
1
1
1
*
TileTest
initialiseElements() void
testElementsProperlyInitialised() void
testSpeedAmplifierTile() void
testLowerSpeedTile() void
testTeleportationTile(int) void
testNormalTile(int) void
testWallTile(String) void
testConveyerTile() void

Figure 9.13 - TileTest SubSystem

## PostgameUITest SubSystem

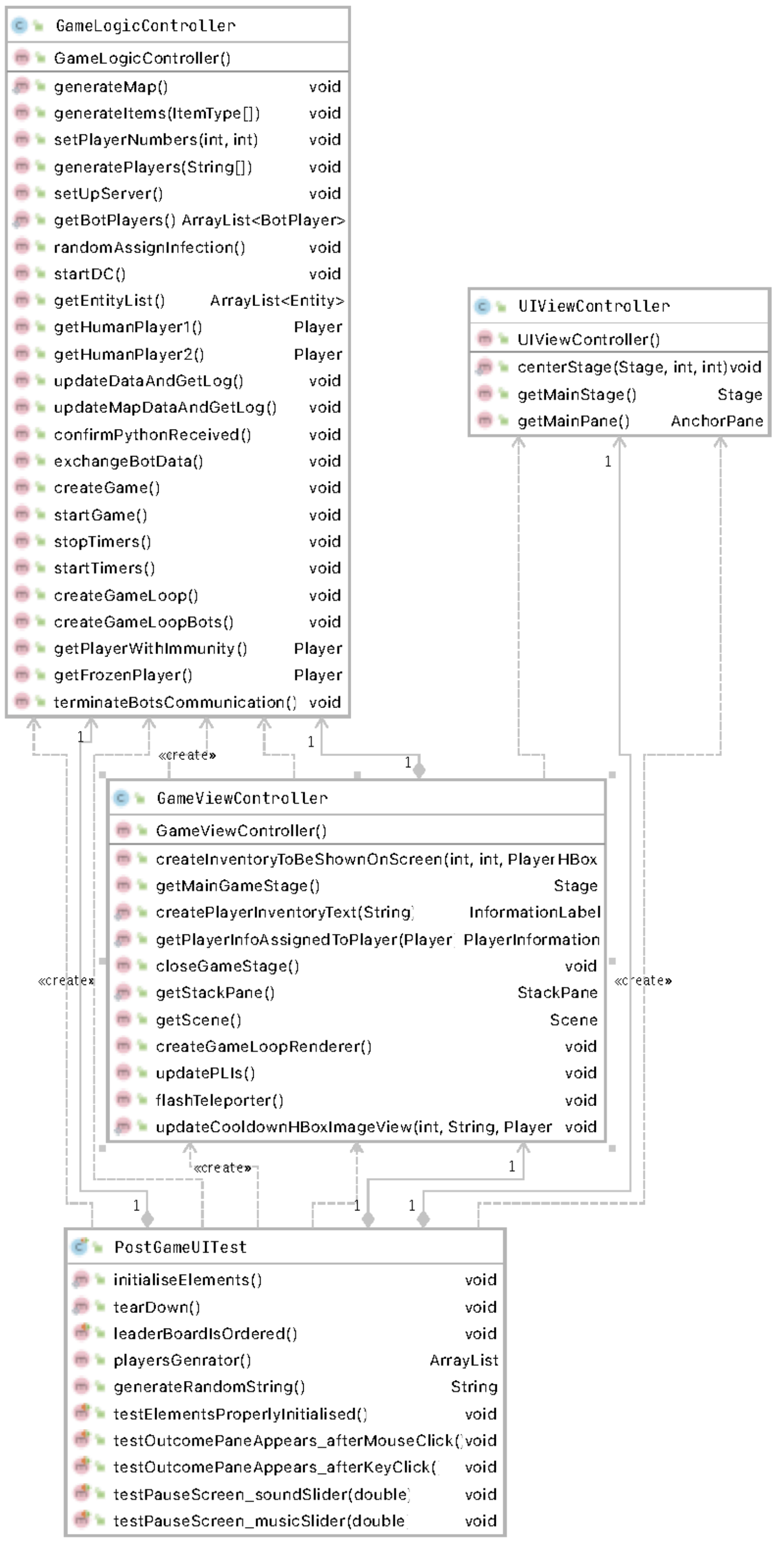
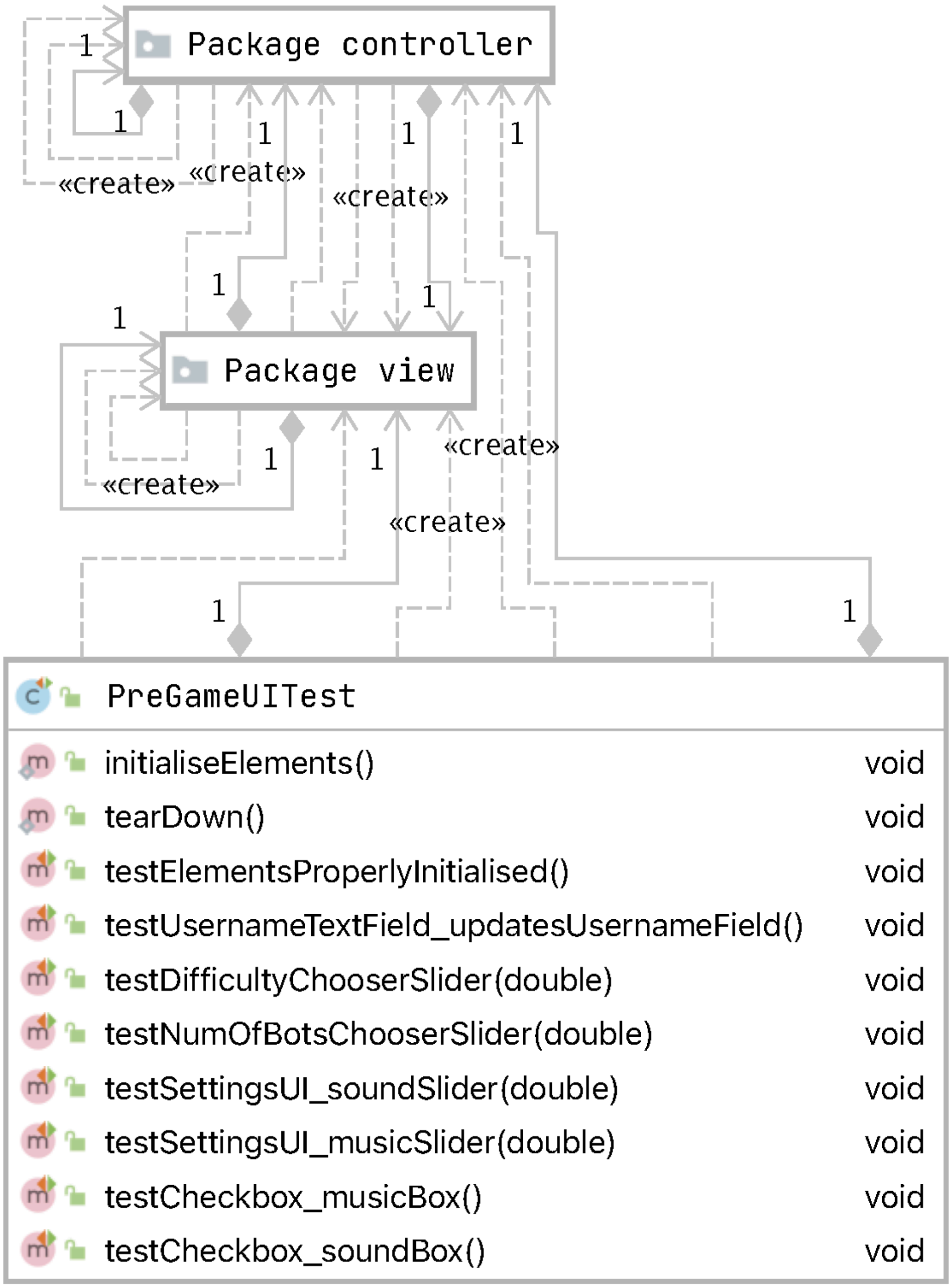

GameLogicController
GameLogicController()
generateMap() void
generateItems(ItemType[]) void
setPlayerNumbers(int, int) void
generatePlayers(String[]) void
setUpServer() void
getBotPlayers() ArrayList<BotPlayer>
randomAssignInfection() void
startDC() void
getEntityList() ArrayList<Entity>
getHumanPlayer1() Player
getHumanPlayer2() Player
updateDataAndGetLog() void
updateMapDataAndGetLog() void
confirmPythonReceived() void
exchangeBotData() void
createGame() void
startGame() void
stopTimers() void
startTimers() void
createGameLoop() void
createGameLoopBots() void
getPlayerWithImmunity() Player
getFrozenPlayer() Player
terminateBotsCommunication() void
UIViewController
UIViewController()
centerStage(Stage, int, int) void
getMainStage() Stage
getMainPane() AnchorPane
GameViewController
GameViewController()
createInventoryToBeShownOnScreen(int, int, Player HBox
getMainGameStage() Stage
createPlayerInventoryText(String) InformationLabel
getPlayerInfoAssignedToPlayer(Player PlayerInformation
closeGameStage() void
getStackPane() StackPane
getScene() Scene
createGameLoopRenderer() void
updatePLIs() void
flashTeleporter() void
updateCooldownHBoxImageView(int, String, Player void
PostGameUITest
initialiseElements() void
tearDown() void
leaderBoardIsOrdered() void
playersGenrator() ArrayList
generateRandomString() String
testElementsProperlyInitialised() void
testOutcomePaneAppears_afterMouseClick() void
testOutcomePaneAppears_afterKeyClick() void
testPauseScreen_soundSlider(double) void
testPauseScreen_musicSlider(double) void
«create»
«create»
«create»
«create»
1
1
1
1
1
1
1
1

Figure 9.14 - PostgameUITest SubSystem

## PreGameUITest SubSystem


Package controller
Package view
PreGameUITest
«create»
1
initialiseElements() void
tearDown() void
testElementsProperlyInitialised() void
testUsernameTextField_updatesUsernameField() void
testDifficultyChooserSlider(double) void
testNumOfBotsChooserSlider(double) void
testSettingsUI_soundSlider(double) void
testSettingsUI_musicSlider(double) void
testCheckbox_musicBox() void
testCheckbox_soundBox() void

## PlayerTest SubSystem

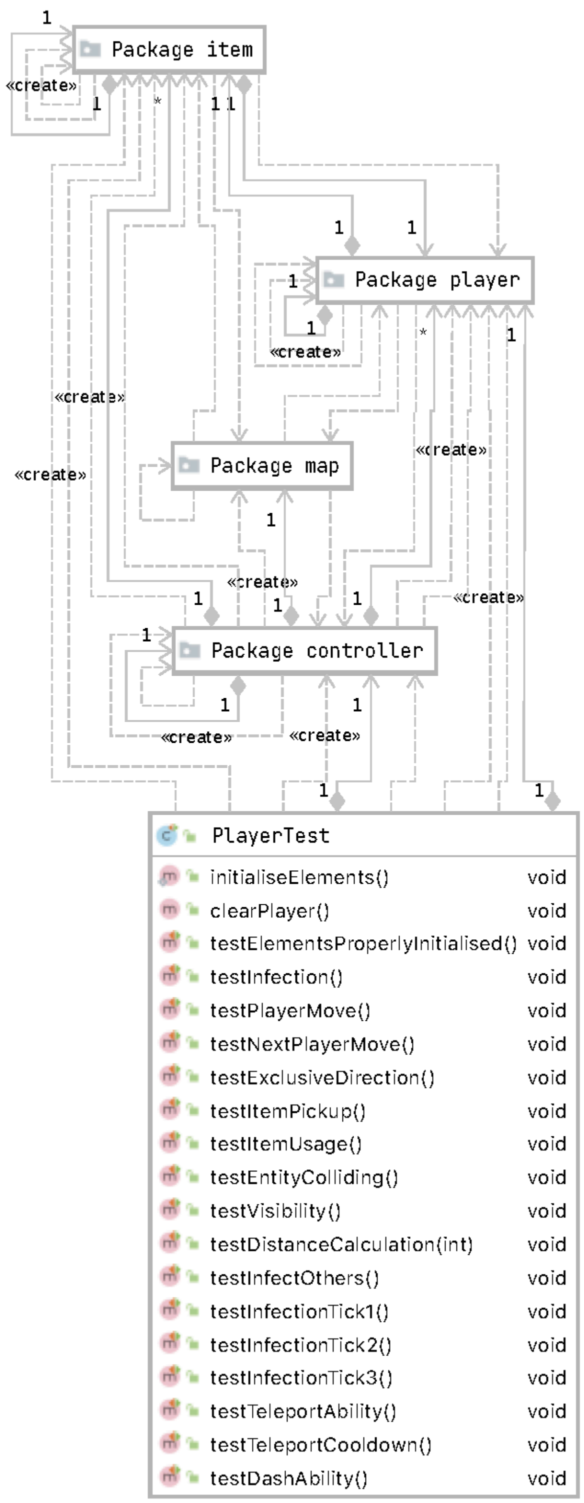

Package item
Package player
Package map
Package controller
«create»
1
*
PlayerTest
initialiseElements() void
clearPlayer() void
testElementsProperlyInitialised() void
testInfection() void
testPlayerMove() void
testNextPlayerMove() void
testExclusiveDirection() void
testItemPickup() void
testItemUsage() void
testEntityColliding() void
testVisibility() void
testDistanceCalculation(int) void
testInfectOthers() void
testInfectionTick1() void
testInfectionTick2() void
testInfectionTick3() void
testTeleportAbility() void
testTeleportCooldown() void
testDashAbility() void

Figure 9.16 - PlayerTest SubSystem

## Artificial Intelligence SubSystems

### Overview of Python SubSystem

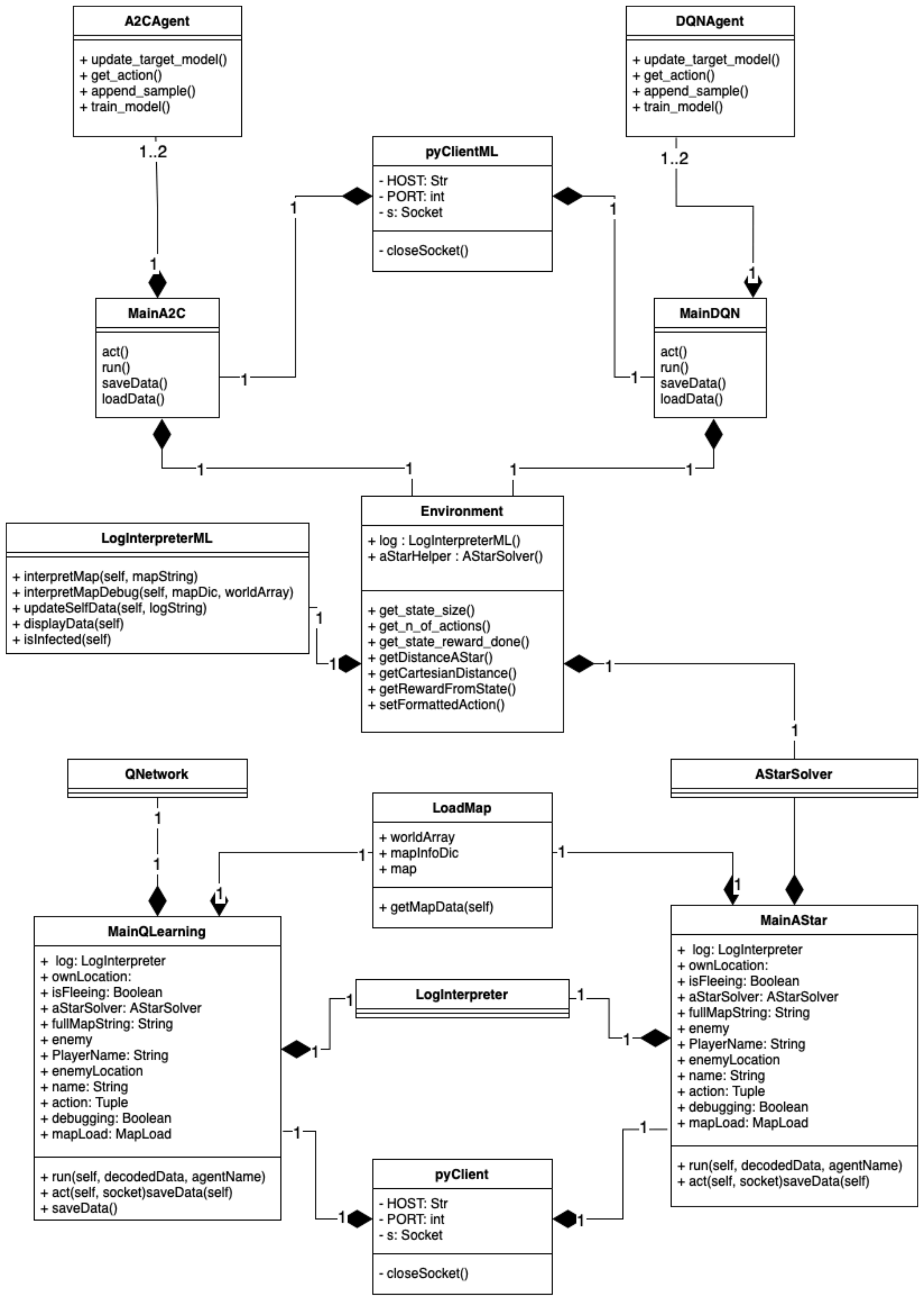


Figure 9.17 - High Level View of AI communication in Python

## AStar

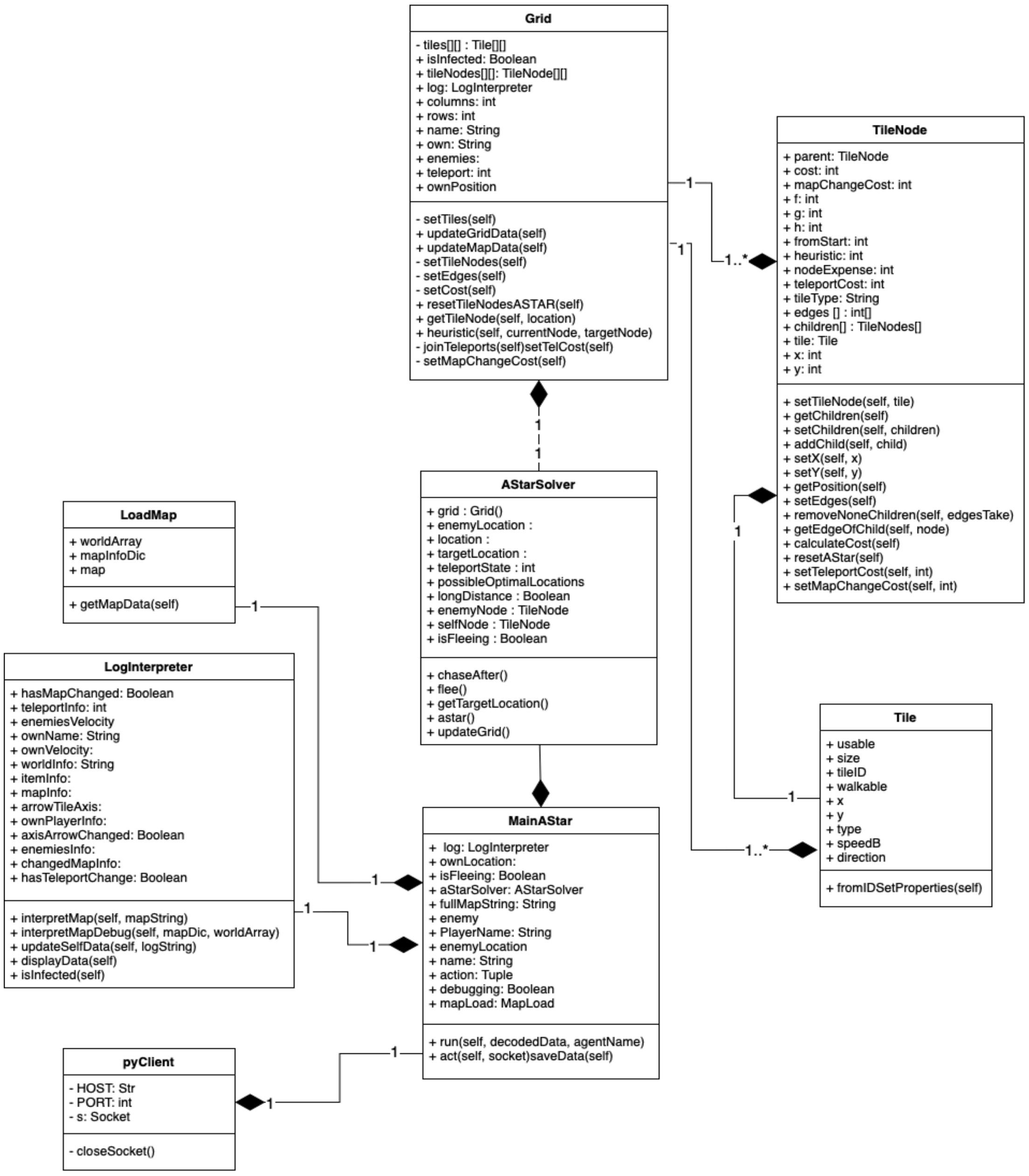


Figure 9.18 - A Star SubSystem

## QNetwork

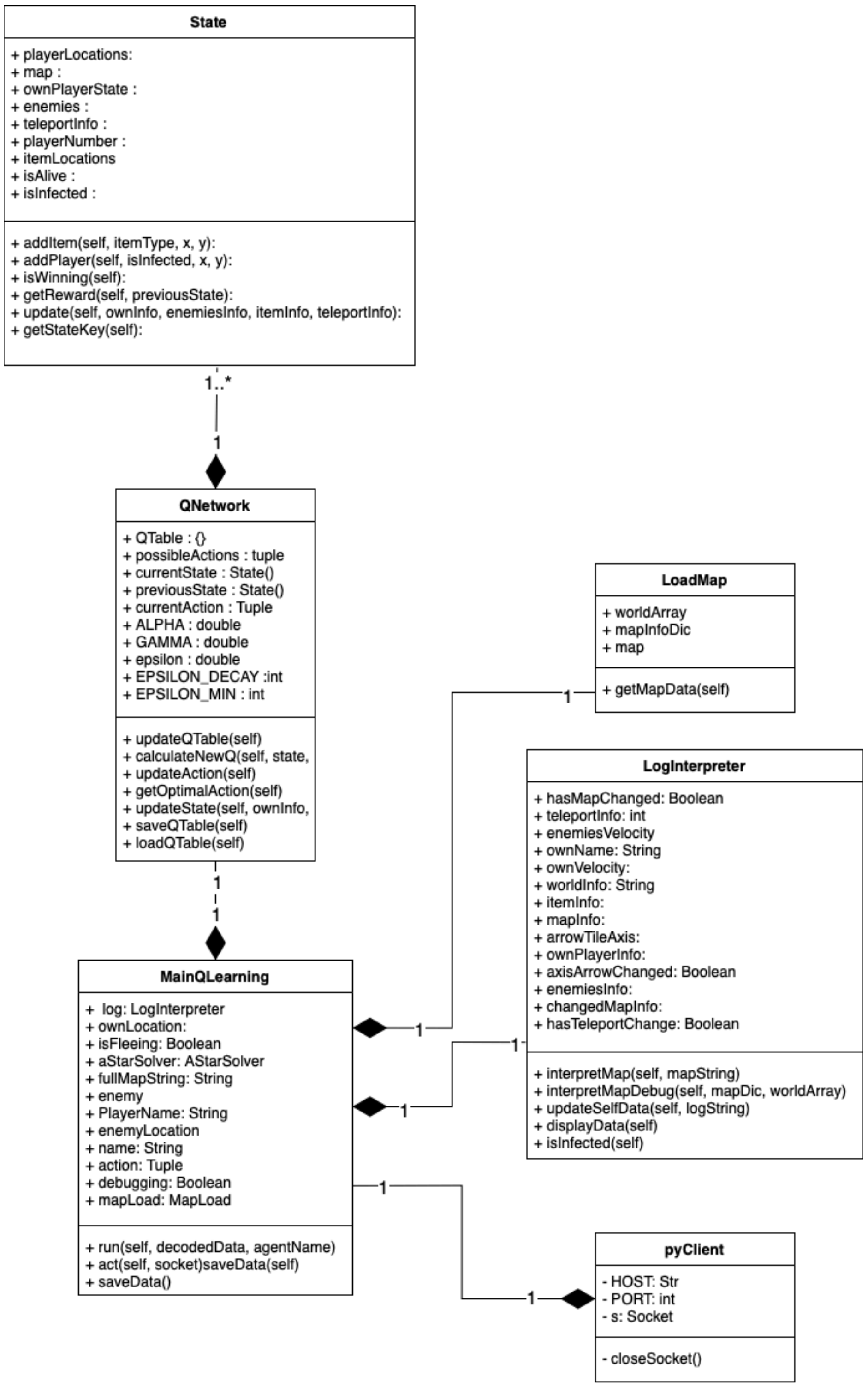


Figure 9.19 - Q Learning SubSystem

## DeepQNetwork

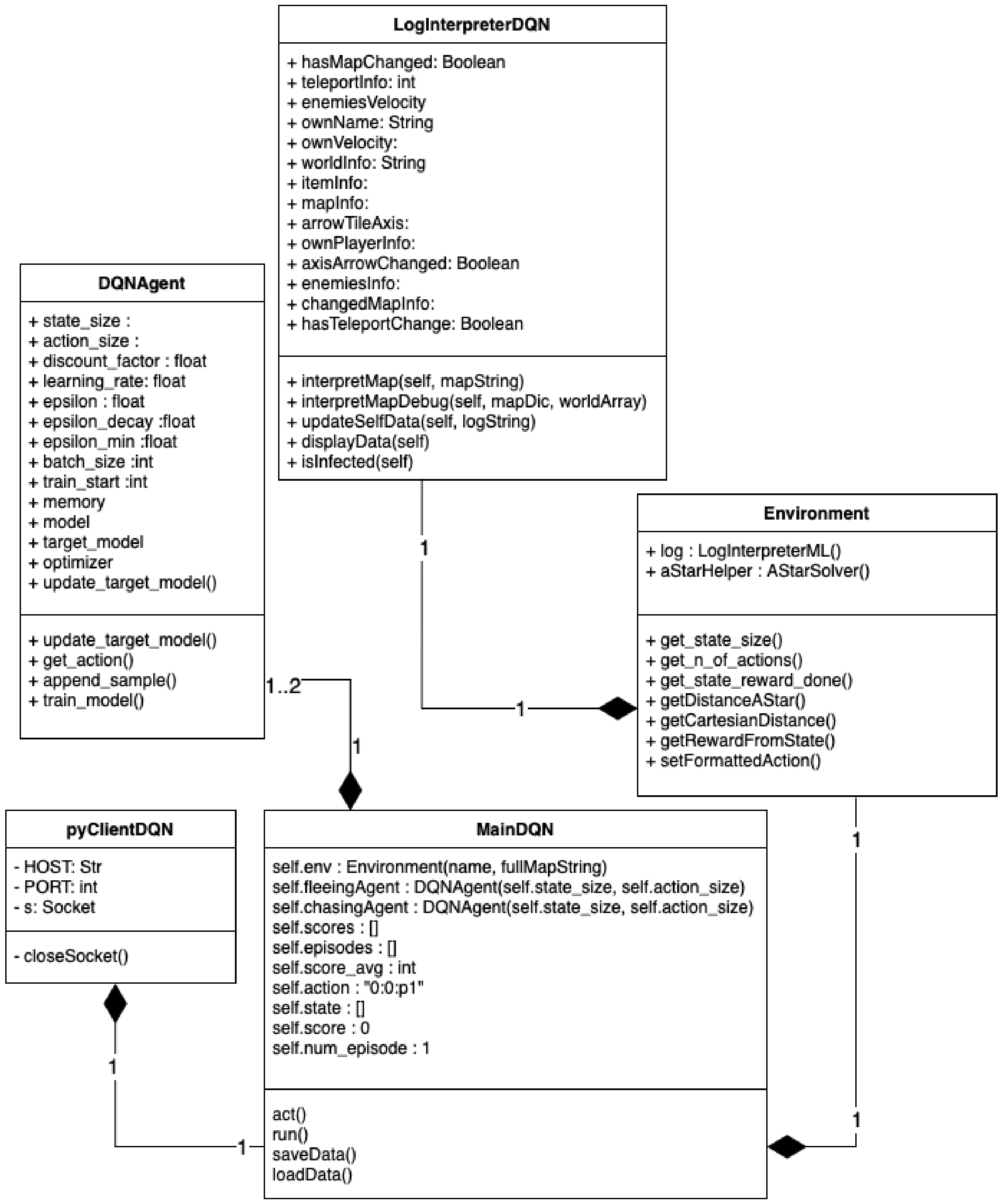


Figure 9.20.1 - Deep Q Network SubSystem - Version 1 - GamePlay Execution

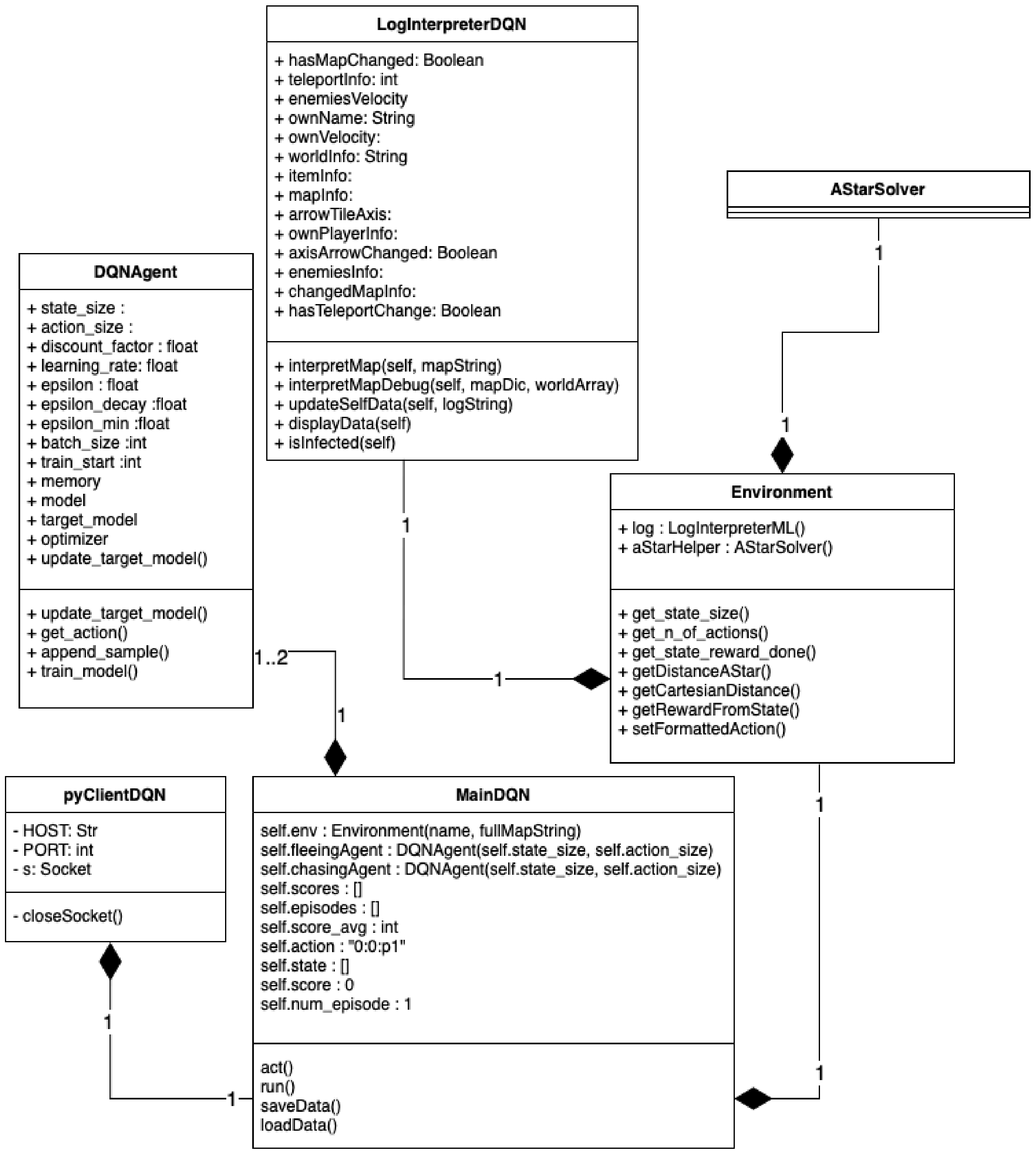


Figure 9.20.2 - Deep Q Network SubSystem - Version 2 - Training - A Star as aid.

## Actor Critic

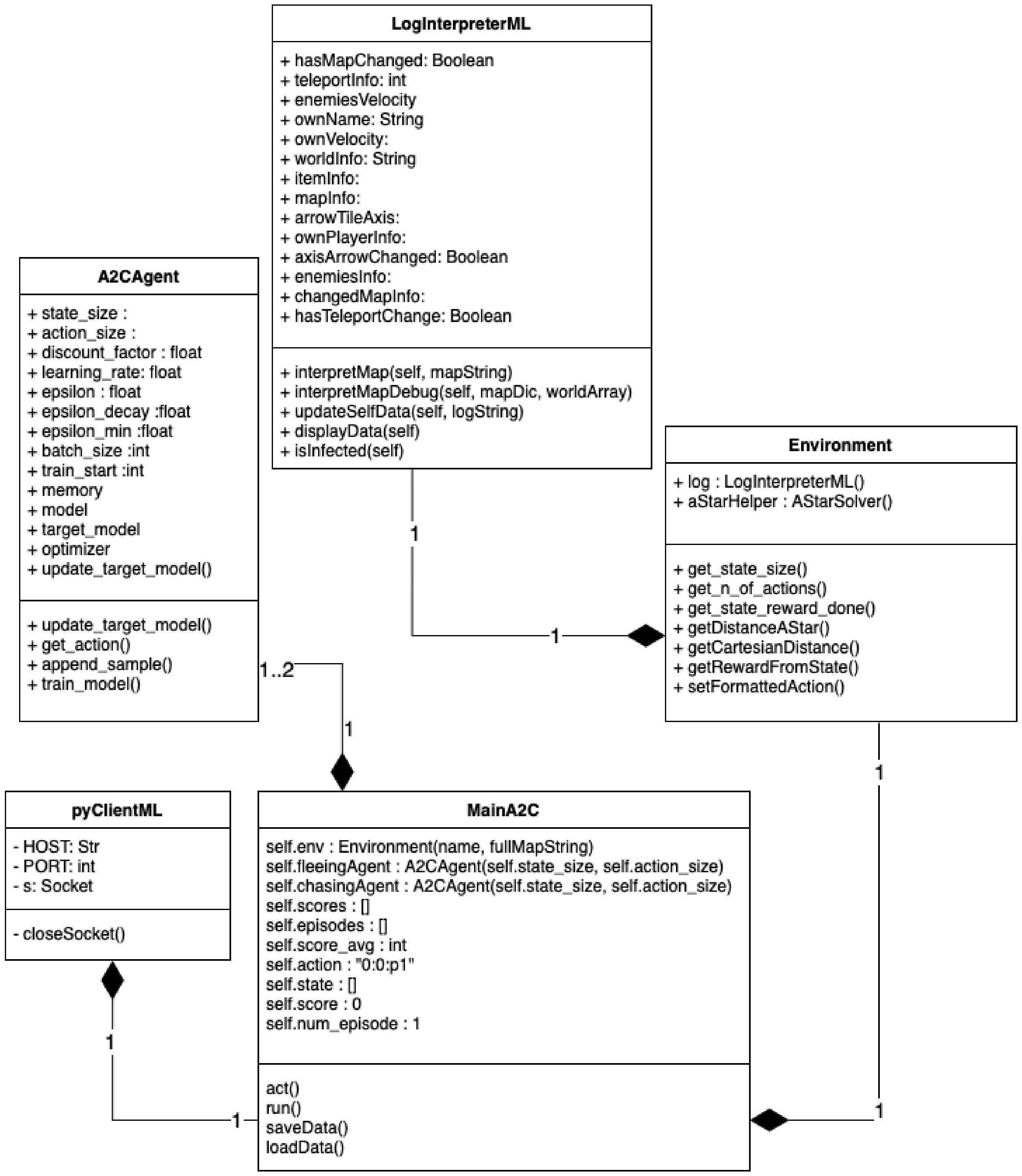


Figure 9.21.1 - Actor Critic Subsystem - Version 1 - GamePlay Execution

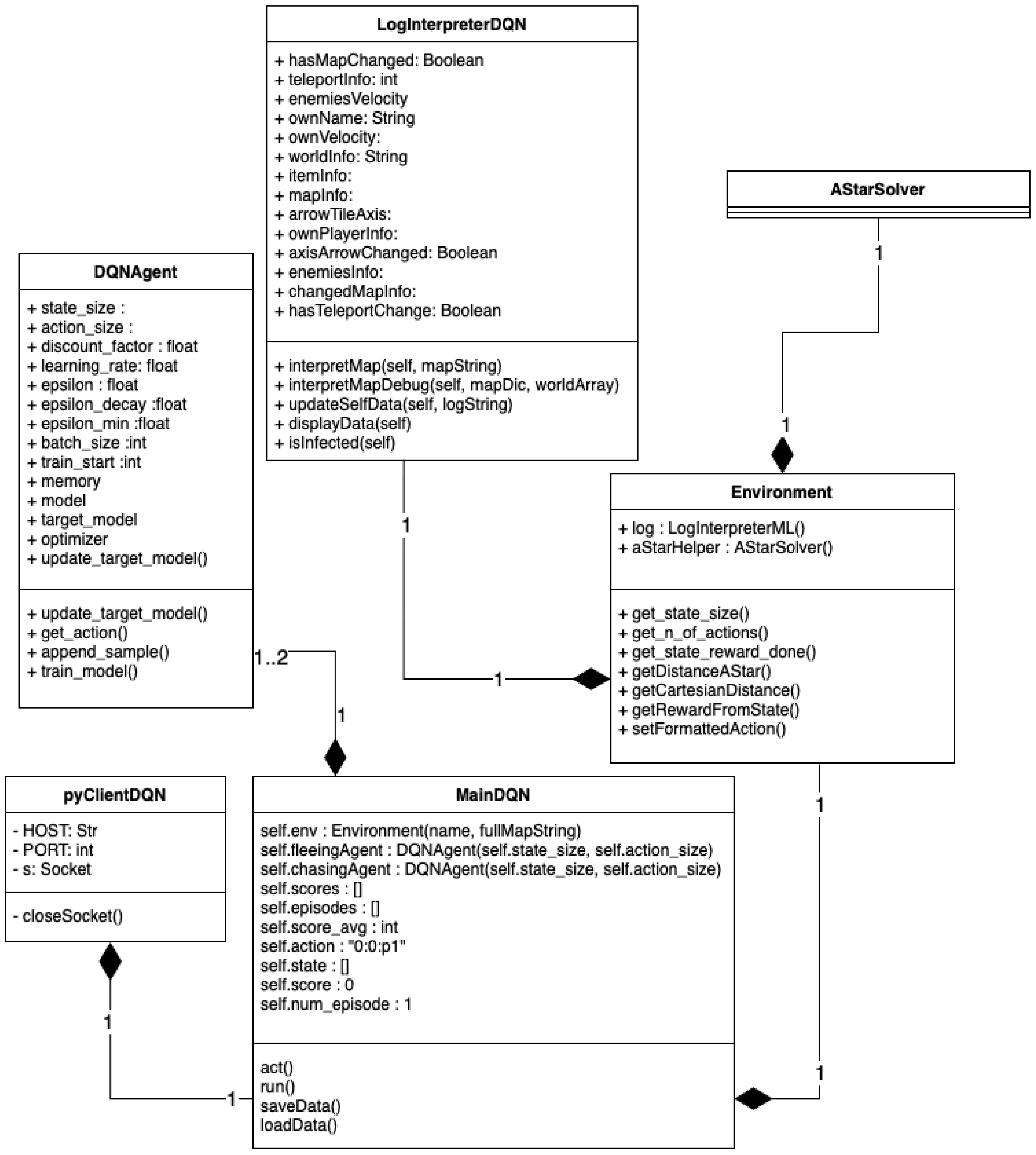


Figure 9.21.2 - Actor Critic Subsystem - Version 2 - Training - A Star as aid.

## Sample Class Diagrams

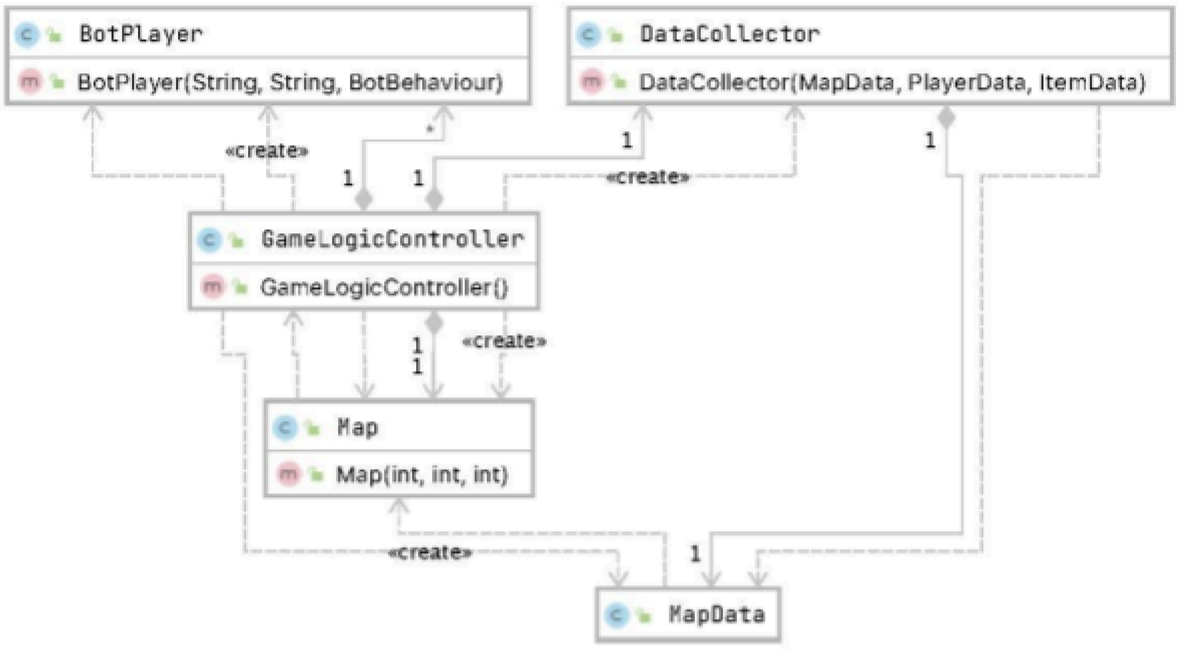


Figure 9.22 - Example of where FlyWeight Structure Pattern was used.

## External sounds and Assets

Apart from a few exceptions, all sounds and art in our game have been independently produced, in order to bring out the best of our cyberspace aesthetic.

Sound sources

Wavclick5.wav

Rollover4.wav

These files are royalty free and can be used for educational purposes.

Asset sources

NotoSansKR-Bold: https://fonts.google.com/specimen/Noto+Sans+KR

Snowflake: https://www.stickpng.com/img/nature/snow/snowflake-black

# Test Report

## 1. Introduction (IEEE-829-3)

The system consists of three main components:

- a game engine used to:
  - transition between different parts of the system
  - retrieve data from the AI system
  - control basic movement, actions and simple features like animation timers, which can then be used for other components to build different types of games, using the same basis
- a frontend user interface (UI) which allows the user to:
  - easily see what is happening in the game (render two-dimensional graphics),
  - create interactive menu screens which support changes to settings and send them to the game engine,
- an artificial intelligence (AI) system used to:
  - handle web socket communication with the Java engine
  - easily manipulate data with Python,
  - train and use AI to make decisions for bot players

Due to the large differences in the requirements and the technologies used when creating the aforementioned components, a different testing approach had to be used for each of them.

The amount of testing varied depending on the importance and feasibility of testing of each component. The testing objectives were as follows:

- Bugs which slow the game down or make it unplayable (crashes) should be identified and fixed before the game is live.
- The system must satisfy ALL functional and non-functional requirements.

## 2. Test Items (IEEE-829-4)

The most important components to test included entity movement, ability usage, UI view switching, conveyer belt movement, speed multipliers, communication with Python. Particular care was taken when investigating the boundary conditions for movement in order to prevent undesirable behaviours which violate the game mechanics. A further consideration to be made regards system performance and how different numbers of bots/players affected the performance of the game. The game was tested on a range of different operating systems, including Windows (10), MacOS and Linux distributions.

## 3. Features to be Tested (IEEE-829-6)

### Functional

1. The system should allow the user to choose a username/alias for their player before starting the game.
2. The system should allow the user to choose the number of bots/players to use.
3. The system should allow the user to choose whether to use WASD or arrow keys to control their player.
4. The system should allow the user to use a range of items they have acquired in the game.
5. The system should allow the user to catch the infection and pass it onto other non-infected participants.
6. The system should not allow the user to leave the play area or walk on non-traversable tiles.
7. The system should not prevent the user from winning, i.e. bots that are too difficult or there are no win conditions.
8. The system must allow the user to play against at least one bot.
9. The system should allow the user to play against two or three bots.
10. Each bot's AI must differ from each other in terms of AI technique.

### Non-Functional

1. The game should open within 5 seconds.
2. The settings applied by the user should occur within 2 seconds.
3. The game should start within 10 seconds (after clicking 'start game').
4. The AI should respond to the game within 1 second (before data becomes outdated).
5. The game should not drop below 60 frames per second (FPS) while playing.

## 4. Features not to be Tested (IEEE-829-7)

1. Performance on outdated operating systems (e.g. Windows XP or Windows Vista).
2. Performance on outdated hardware (e.g. Single core processors).
3. Performance of the AI against state-of-the-art, proprietary models.

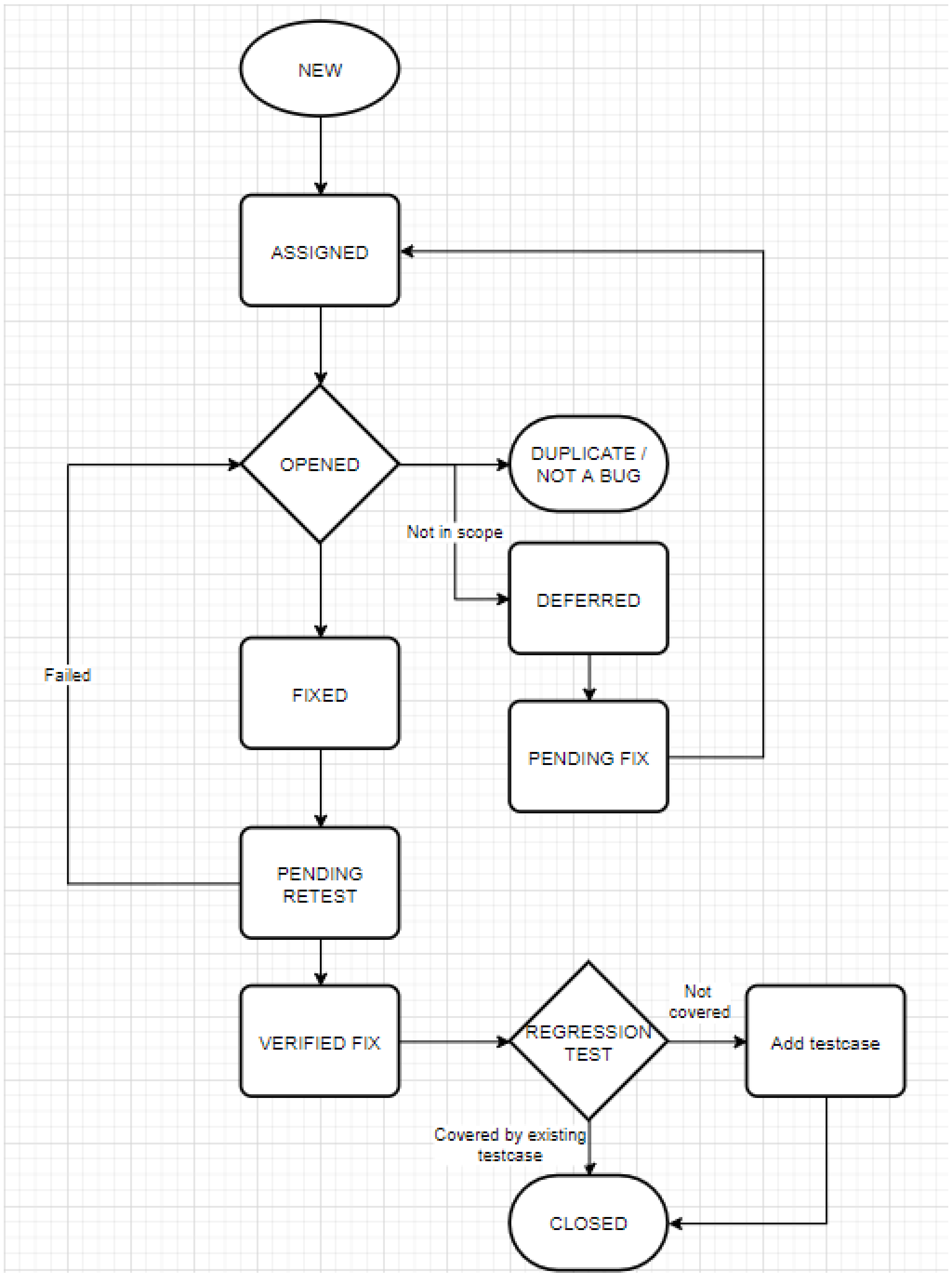
## 5. Approach (IEEE-829-8)

Since the testing will be done by the programmers too, it seems suitable that the approach to use is white-box testing. This method of testing primarily exists to test smaller components of the system where their operation is well-understood. Since the testers will also be the programmers, it seems the most suitable. Black-box testing would be more suitable for testing general functionality of the system, in which the test cases can be derived from the system requirements. The black-box tests can be used to test whether the system meets the basic system requirements and the standards outlined above.

White box tests will be delegated to team members depending on which parts of the code they have written or have extensive knowledge of; a member who has written code for items will know its intended function and so they would be in a better position to write tests for those. Criteria for tests passing or failing will be decided by the testers at the time the tests are written. Comments will be written after the tests are carried out by the tester including the produced behaviour, regardless of whether the test passed or failed. Other testers will then review the comments written by each other to check that a test has not wrongly failed or passed.

Once this has been completed, any tests which have failed will be corrected using the test fix method. These tests will be assigned to the programmer who wrote the code for the part of the code that caused the test to fail for it to be rectified. This process is then repeated until the test has passed correctly.

## 6. Test-Fix Cycle



Once all tests have been marked as a 'Pass', provided each tester gives approval, the testing process will be complete.

## 7. Levels of Testing and Testing Methods

**N.B.** The words 'testers' and 'programmers' are used interchangeably since testing will be done by programmers.

Testing methods would be a combination of:

- *Unit Testing* – Testers will use white-box testing and test coverage techniques to test modules of the code. The unit test report would produce failure/error descriptions which would then be checked by the programmers.
- *Integration Testing* – Testers will combine parts of the system together to ensure that they work together as well as working separately (unit testing). This is carried out as the system is being developed rather than at the end of the development process.
- *Acceptance Testing* – Tests created to ensure the system meets the user requirements. Done by testers.
- *System Testing* – Checks that the system provides the functionality that was promised and that it has no fault. Conjunction of all testing techniques mentioned above.

## 8. Pass/Fail Criteria (IEEE-829-9)

**N.B.** In the table below, the prefixes 'B' and 'W' in the 'Case ID' refer to Black-Box and White-Box testing respectively.

| Case ID | Description | Relevant Components | Steps | Data | Expected Result | Actual Result | Pass / Fail | Comments |
|---|---|---|---|---|---|---|---|---|
| B-WC-1-a | Navigate from main menu to 'Play' menu | - | Click play | - | Should open 'Play' menu | Opens 'Play' menu | Pass | |
| B-WC-1-b | Navigate from 'Play' menu to starting game | - | Click play, type "Player" as username, choose hard difficulty, select 0 bots, select continue | - | Should start game | Starts game | Pass | |
| B-WC-1-c | Navigate from main menu to 'Settings' menu | - | Click settings | - | Should open settings page | Opens settings page | Pass | |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| B-WC-1-d | Navigate from main menu to 'Help' menu | - | Click help | - | Should open help page | Opens help page | Pass | |
| B-WC-2-a | User's name appears on screen during the game. | - | Select play, type data (username), choose hard difficulty, select 1 bot, select continue. | 0123 4567 89 | The username '0123456789' should appear on the screen in the bottom left block. | Username '01234567 89' appeared on the screen on the bottom left block. | Pass | Text identical to the username typed in appeared in the bottom left block next to items. |
| B-WC-2-b | Very long username appears on screen during the game, without covering other components. | - | Select play, type data (username), choose hard difficulty, select 1 bot, select continue. | qwer tyuio pasdf ghjkl zxcvb nm | Username 'qwertyuiopa sdfghjklzxcvb nm' should be cut off to prevent it from covering other components | Username 'qwertyuio pasdfg' appears on the bottom left of the screen. | Pass | Text gets cut off to prevent it from covering up other components |
| B-WC-3-a | Music checkbox disables music | - | Select settings, move music slider to 100, untick music | Volu me = 100 | Music should stop playing (muted) | Music becomes muted | Pass | |
| B-WC-3-b | Sound checkbox disables sound | - | Select play, choose any username, hard difficulty, 1 bot, start game, press ESC, move sound slider to 100, untick sound | Volu me = 100 | Sound should stop playing (muted) | Sound becomes muted | Pass | Sound becomes muted but there is no sound to begin with |
| B-WC-3-c | Music slider increases volume | - | Select settings, move slider to right | Volu me = 75 | Music volume should increase noticeably, by more than double original value (since original is ~30) | Music volume increases drastically | Pass | Music volume noticeably increases, difficult to test exactly through black-box testing |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| B-WC-3-d | Music slider decreases volume | - | Select settings, move slider to left | Volume = 0 | Music should become muted | Music is muted, “Music” checkbox unticked | Pass | Music becomes non-existent and checkbox for “Music” becomes unticked |
| B-WC-3-e | Sound slider increases volume | - | Select play, choose any username, hard difficulty, 1 bot, start game, press ESC, hover over restart button to let sound play, move Sound slider to right, then hover over restart button again and check that sound volume is higher than before | Volume = 75 | Sound volume should increase noticeably | Sound volume increases noticeably | Pass | |
| B-WC-3-f | Sound slider decreases volume | - | Select play, choose any username, hard difficulty, 1 bot, start game, press ESC, hover over restart button to let sound play, move Sound slider to left, then hover over restart button again and check that no sound is played | Volume = 0 | Sound volume should become muted | Sound volume becomes muted, “Sound” checkbox is unticked | Pass | |
| B-WC-3-g | Music volume saved when disabling and re-enabling | - | Select settings, move Music slider to 100, untick Music, re-tick Music | Volume= 100 | Music volume should return to 100 | Music volume returns to default value (~30) | Fail | Music volume is saved between game restarts, but upon unticking and then re-ticking the box, volume returns to default |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| B-WC-3-h | Sound volume saved when disabling and re-enabling | - | Select play, choose any username, hard difficulty, 1 bot, start game, press ESC, move sound slider to right, untick sound, re-tick sound | Volume = 100 | Sound volume should return to 100 | Sound volume returns to default value (~30) | Fail | Sound volume is saved between game restarts, but upon unticking and then re-ticking the box, volume returns to default |
| W-WC-IT-a | Test items can be picked up | Item, Player | Loop through all spawned items, move player to the item, collect the item with the player, check that item has been collected, move the player to a random normal tile, use the item, check the item has been removed from player's inventory, move second player to item's old position to check that it isn't still there. | Spawned Items | Should pass all 4 assertions; player has no item prior to picking up, player has item after picking up, player has no item after using it, and other player cannot pick up item that has been used. | Passes all 4 assertions | Pass | Test passes but unsure whether it should, last assertion may be wrong. Needs checking |
| W-WC-IT-b | Test time extender item works | Item, Player, GameViewController | Create new time extender item, simulate infection timer increases, store timer before using item, use item, make sure player no longer has item, check that the infection timer before and | - | Should pass both assertions; player has used item (no longer in inventory) and that the infection timer before and after use of the item are different. | Passes both assertions | Pass | |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | after use are different. | | | | | |
| W-WC-IT-b | Test player cannot be infected while immunity cooldown is still active | Player, Item, GameViewController | Create immunity item and add to healthy player's inventory, move infected player to healthy player. Simulate infection tick, make healthy player use immunity item, make infected player try and infect healthy player. Check that healthy player did not get infected and that infected player stays infected. | - | Should pass both assertions; infected player stays infected and healthy player stays healthy. | Passes both assertions | Pass | |
| W-WC-IT-c | Test player is infected when immunity cooldown is no longer active | Player, Item, Map, GameViewController | Create immunity item and add to healthy player's inventory, move healthy and infected players to a random tile. Make healthy player use the immunity item, wait until cooldown expires, make infected player try and infect healthy player after cooldown expires. Check that infected player is no longer infected and that | - | Should path both assertions; infected player is no longer infected and healthy player is now infected. | Passes both assertions | Pass | |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | healthy player is now infected. | | | | | |
| W-WC-IT-d | Test that infected player is silenced when healthy player fires arrow at them | Player, Item, GameViewController | Create silence item and add to healthy player's inventory, put both players in the same coordinate on the y-axis but have different values (300 and 40) as the x position. Make healthy player aim at infected player (right) and fire arrow. Check that healthy player no longer has the item, wait for arrow to hit infected player then check if infected player becomes silenced. | - | Should pass both assertions; arrow is no longer in healthy player's inventory and infected player becomes silenced. | Passes both assertions | Pass | |
| W-WC-IT-e | Test that infected player does not get silenced when healthy player misses their arrow shot | Player, Item, GameViewController | Create silence item and add to healthy player's inventory, put players at different positions, where their x nor y coordinates are the same, i.e. (40, 250) and (400, 200). Make healthy player aim right and fire their arrow, check to make sure healthy | - | Should pass both assertions; arrow is no longer in healthy player's inventory and infected player is not silenced. | Passes both assertions. | Pass | |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | player now has no item. Wait for 1.5 seconds so the arrow has time to hit the infected player then check that the infected player is not silenced. | | | | | |
| W-WC-IT-f | Test that player movement speeds change when in quarantine area | Player, Item, GameViewController, Map | Set quarantine duration to 0, create new quarantine item and add to healthy player's inventory. Move players 5 squares apart, make healthy player use quarantine item, simulate time passing by 0.5 seconds, move players and check that the healthy player's speed increases and the infected player's speed decreases in the quarantine zone. Move the infected player outside the zone and check that their speed is now back to normal. | - | Should pass all 3 assertions; healthy player's speed is greater than the default, check that the infected player's speed is less than the default, and check that the infected player's speed is back at the default after leaving the quarantine zone. | Passes all 3 assertions | Pass | |
| W-WC-PT-a | Test that setting player infection to true actually infects player | Player | Infect player, check that their infected value is true, check that their status is "infected", heal | - | Should pass all 4 assertions; checking player is infected, checking | Passes all 4 assertions | Pass | |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | player, check that their infected value is false, check that their status is "healthy". | | status is "infected", checking player is no longer infected, and checking that the player's status is now "healthy". | | | |
| W-WC-PT-b | Test that player moves correct distance each tick | Player | Store player's current position, set player's movement direction to up and right, move the player, check that the new x value is the same as the original x value + movement speed and that the new y value is the same as the original y value - movement speed. | - | Should pass both assertions; player has moved to the correct new x position and that the player has moved to the correct new y position. | Passes both assertions | Pass | |
| W-WC-PT-c | Test that next move generated is correct | Player | Store player's current position, set player's movement direction to down and left, generate the next move. Check that the generated x value is the same as the original x value - movement speed and that the generated y value is the | - | Should pass both assertions; game generated correct next x position and also generated correct next y position. | Passes both assertions | Pass | |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | same as the original y value + movement speed. | | | | | |
| W-WC-PT-d | Test that exclusive direction moves player correct distance in correct direction | Player | Store player's current position, set the player's exclusive direction as right, check that the player's new x position is the original + movement speed. Set the player's exclusive direction to up and check that the player's new y position is the original - movement speed. | - | Should pass both assertions; player moves to correct new x position and player moves to correct new y position. | Passes both assertions | Pass | |
| W-WC-PT-e | Test that player can pick up item | Item, Player | Create new freeze item, place item in same tile as player, make player collect item, check that item is in player's inventory. | - | Should pass assertion; check freeze item is in player's inventory. | Passes assertion. | Pass | |
| W-WC-PT-f | Test that player can use item (immunity) | Item, Player | Create new immunity item, move item to player's position, store current immunity timer, make player collect item, make player use item, check that immunity | - | Should pass both assertions; check immunity timer has increased and check that player's inventory is now null. | Passes both assertions. | Pass | |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | timer has increased, check that player no longer has item. | | | | | |
| W-WC-PT-g | Test that player can detect when it is colliding with another player | Player | Create new temp player (2) at the same position as player (1), check that the collidesWith method returns true when 1 collides with 2. | - | Should pass assertion; check that 1 detects when it collides with 2. | Passes assertion. | Pass | |
| W-WC-PT-h | Test that hiding and showing methods make player invisible and visible | Player | Store player's original visibility state, hide player, check that player is invisible, show player, check that player is visible, restore player's old visibility state. | - | Should pass both assertions; player is invisible after calling hide() and player is visible after calling show(). | Passes both assertions | Pass | |
| W-WC-PT-i,j,k,l,m,n,o | Test that player calculates correct distance between other player. (Parameterized Test) | Player | Create temp player (2) at the position of player (1) + distance parameter passed into test, calculate answer using simplified equation, check that both distance calculations are the same. | Distance = ints ={-3,-2,-1,0,1,2,3} | Should pass assertion 7 times; check that both distance calculations give same result for every distance passed in. | Passes assertion all 7 times | Pass | |
| W-WC-PT-p | Test that player can infect another player | Player | Create temp player (2) at the same position as player (1), | - | Should pass both assertions; check that 1 | Passes both assertions | Pass | |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | infect 1, set cooldown immunity timers to 0 for both players, make 1 try and infect 2, check that 2 is now infected and 1 is now healthy. | | is healthy and that 2 is infected. | | | |
| W-WC-PT-q | Test that infection kills player | Player | Create temp player, infect temp player, tick infection timer while setting max infection time for temp player to 0, check that temp player is not alive and is not visible. | - | Should pass both assertions; check that temp player is no longer alive and check that temp player is invisible. | Passes both assertions | Pass | |
| W-WC-PT-r | Test that infection doesn't kill player until timer ends | Player | Create temp player, infect temp player, tick infection timer while setting infection time for temp player to a large value (e.g. 500000), check that temp player is still alive and still visible. | - | Should pass both assertions; check that temp player is still alive and is still visible. | Passes both assertions | Pass | |
| W-WC-PT-s | Test that infection does not make player alive when passed to a dead player | Player | Create temp player, get other player to kill temp player, infect temp player. Tick infection timer while setting infection time for temp player to a large value (e.g. 500000), | - | Should pass both assertions; check that temp player is still dead and still invisible. | Passes both assertions | Pass | |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | check that temp player is still dead and still invisible. | | | | | |
| W-WC-PT-t | Test that player gets teleported to teleporter when using teleporter ability | Player, PlayerAbility | Set player's ability cooldown to 0, use the teleporter (place teleporter), store teleporter's position, move player to new tile, use teleporter, set cooldown to 0 again and check that player has teleported to teleporter. | - | Should pass both assertions; check that x position is the same as teleporter and check that y position is the same as teleporter. | Passes both assertions | Pass | |
| W-WC-PT-u | Test that player cannot use teleporter again before cooldown expires | Player, PlayerAbility | Use teleporter, move the player, use the teleporter, move the player, use the teleporter, move the player, store the player's current position, then use the teleporter again and check that the player has not moved. | - | Should pass both assertions; check that player's x position has not changed and check that player's y position has not changed. | Passes both assertions | Pass | |
| W-WC-PT-v | Test that dash does not let player move out of the map | Player | Move player just to the right of the far left side of the map, set the player's movement direction to left | - | Should pass both assertions; check that player is at far left of map (not outside map) | Passes second assertion, does not pass first. Says that player is in the wall. | Fail | Far left side has coordinates (26,y), after using dash, the player has the position (25,y), needs looking into. |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | and activate the dash ability. Check that the player's new x position is the far left of the map and that their y position has not changed. | | and check that player's y position has not changed. | | | |
| W-WC-PoT-a | Test that the outcome pane appears upon clicking the mouse (repeated 5 times) | GameOutcomeScene | Create new mouse event, fire mouse event on the outcome scene, check that the outcome pane is now the current scene and check that the outcome pane is not null. | - | Should pass both assertions 5 times; check that outcome pane becomes the new current scene and check that the outcome pane is not null. | Passes both assertions all 5 times | Pass | |
| W-WC-PoT-b | Test that outcome pane appears upon pressing key (repeated 5 times) | GameOutcomeScene | Create new key event (K key), fire key event on the outcome scene, check that the outcome pane is now the current scene and check that the outcome pane is not null. | - | Should pass both assertions 5 times; check that outcome pane becomes the new current scene and check that the outcome pane is not null. | Passes both assertions all 5 times | Pass | |
| W-WC-PoT-c | Test sound slider in pause screen (Parameterized test) | SettingsUI | Set slider value to value passed in as 'volume', check that the in game sound volume is now the same as the value passed into the test, check that | Volume = {4.5, 100.0, 99.1, 32.1, 0.0, 9.8, 0.0, | Should pass all 4 assertions for each value; check that in game volume is same as the value passed in, check that | Passes all 4 assertions for each value passed in. | Pass | |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | the in game sound volume is the same as the volume when pressing a button, check that the in game sound volume is now the same as the value on the slider. If the volume is 0, check that the sound box becomes unticked, if not, check that it is ticked. | 100.0 } | the in game sound volume is the same as the button pressing volume, check that the in game sound volume is the same as the value shown by the slider. If volume = 0, check that the sound box is unticked, otherwise, check that it is ticked. | | | |
| W-WC-PoT-d | Test music slider in pause screen (Parameterized test) | SettingsUI | Create key event, fire key event on main game stage, create new media controller and assign it to correct music file, set slider value to value passed in as 'volume', check that the in music volume is now the same as the value passed into the test, check that the background music volume is now the same as the value on the slider. If the volume is 0, | Volume = {0.0, 100.0, 99.1, 32.1, 0.1, 9.8, 0.0, 100.0 } | Should pass all 3 assertions for each value; check that music volume is the same as the value passed in, check that the background music volume is the same as the value shown by the slider. If volume = 0, check that the music box is unticked, otherwise, check that it is ticked. | Passes all 3 assertions | Pass | |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | check that the music box becomes unticked, if not, check that it is ticked. | | | | | |
| W-WC-PrT -a | Test that the username text field updates the player’s username | - | Set text field to “myUsername” check that the player’s username is equal to “myUsername”, check that the username in the pregame subscene is “myUsername”, change the player’s username to something other than “myUsername” and check that the name has changed in the pregame subscene. | - | Should pass all 3 assertions; check that the player’s username is “myUsername”, check that the player’s username is “myUsername” on the pregame subscene, check that the player’s username has changed in the pregame subscene. | Passes all 3 assertions | Pass | |
| W-WC-PrT -b | Test that the bot number slider shows correct value (Parameterized test) | PreGameSubScene, UIViewController | Change value of bot slider to value passed in as ‘numOfBots’, check that slider value is equal to value passed in, check that the number of bots stored in the UI view controller is equal to the value passed in. | numOfBots = {1.0, 4.0, 3.0, 2.0} | Should pass both assertions for each value; check that number of bots is the same as the slider, check that the number of bots is the same as what is stored in the UI view manager. | Passes both assertions for each value | Pass | |
| W-WC-PrT -c | Test sound slider in settings screen | SettingsUI | Change value of sound slider to value passed | volume = {1.0, | Should pass all 3 assertions | Passes all 3 assertions | Pass | |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | (Parameterized test) | | in as ‘volume’, check that slider value is equal to value passed in, check that the sound slider volume is the same as the value stored in the media controller. If the volume=0, check that the music box is unticked, otherwise, check that it is ticked. | 43.8, 99,9, 0.0, 100.0, 0.2} | for each value; check that slider value is the same as the volume, check that the slider value is the same as the value stored in the media controller. If volume=0, check that sound box is unticked, otherwise check it is ticked. | for each value | | |
| W-WC-PrT-d | Test music slider in settings screen (Parameterized test) | SettingsUI | Change value of music slider to value passed in as ‘volume’, check that slider value is equal to value passed in, check that the music slider volume is the same as the value stored in the media controller. If the volume=0, check that the music box is unticked, otherwise, check that it is ticked. | volume = {1.0, 43.8, 99,9, 0.0, 100.0, 0.2} | Should pass all 3 assertions for each value; check that slider value is the same as the volume, check that the slider value is the same as the value stored in the media controller. If volume=0, check that music box is unticked, otherwise check it is ticked. | Passes all 3 assertions for each value | Pass | |
| W-WC-PrT-e | Test music box is ticked/unticked correctly | SettingsUI | Change value of music slider to non-zero value, check music box is ticked change | 67.5, 0.5, 0.0, 100.0, 0.0 | Should pass all 5 assertions; check music box is ticked, check music | Passes all 5 assertions | Pass | |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | value again to non-zero value and check music box is ticked. Change slider to 0 and check music box is unticked. Change slider to 100 and check music box is ticked. Change slider back to 0 and check that music box is unticked. | | box is still ticked, check music box has become unticked, check music box has become ticked again, and check the music box becomes unticked again. | | | |
| W-WC-PrT-f | Test sound box is ticked/unticked correctly | SettingsUI | Change value of sound slider to non-zero value, check soundbox is ticked change value again to non-zero value and check sound box is ticked. Change slider to 0 and check sound box is unticked. Change slider to 100 and check sound box is ticked. Change slider back to 0 and check that sound box is unticked. | 67.5, 0.5, 0.0, 100.0, 0.0 | Should pass all 5 assertions; check soundbox is ticked, check sound box is still ticked, check sound box has become unticked, check sound box has become ticked again, and check the sound box becomes unticked again. | Passes all 5 assertions | Pass | |
| W-WC-RT-a | Test visible entities are added to screen | EntityRenderer | Call render method for entity renderer, get the image views for the screen and check that entities are | - | Should pass the assertion; Check that entities are added to entity pane. | Passes assertion | Pass | |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | added to screen. | | | | | |
| W-WC-RT-b | Test hidden entities are added to screen | EntityRenderer, Player | Hide both entities, call render method for entity renderer, get the image views for the screen and check that the entities are not added to screen. | - | Should pass the assertion; Check that entities are not added to entity pane. | Passes assertion | Pass | |
| W-WC-RT-c | Test hidden entities are removed from screen | EntityRenderer, Player | Call render method for entity renderer, get image views for the screen, check that entities are added to screen, make entities invisible, call render method for entity renderer method again, check that entities are not added to screen. | - | Should pass both assertions; Check that entities are added to entity pane at start, check that entities are removed from entity pane after hiding them. | Passes both assertions | Pass | |
| W-WC-TT-a | Test that speed tiles increase player's speed | Tile, GameLogicController, Player, Map | Find a random normal tile and get the list of speed tiles, check that the normal tile is not null. Move player to the normal tile and store the speed of the player on the tile. For every speed tile, check that the | - | Should pass both assertions; check that normal tile is not null and check that the player's speed is greater on booster tiles. | Passes both assertions | Pass | |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | player's speed is greater than normal. | | | | | |
| W-WC-TT-b | Test that slow tiles decrease player's speed | Tile, GameLogicController, Player, Map | Find a random normal tile and get the list of slow tiles, check that the normal tile is not null. Move the player to the normal tile and store the speed of the player on the tile. For every slow tile, check that the player's speed is lower than normal. | - | Should pass both assertions; check that the normal tile is not null and check that the player's speed is lower than slow tiles. | | | |
| W-WC-TT-c | Test that teleportation tile teleports player (Parameterized test) | GameLogicController, Player, Map | Get the list of teleport tiles and store the nth teleport tile's coordinates (n is passed in as 'teleporterIndex'). Move player to teleport tile, set the teleport tile's cooldown to 0. Check that the x and y position of the player is not the same as the teleport tile (since player should have moved) and check that the difference between the x and y values of the player and teleport tile is greater than | teleporterIndex = {0,1,2,3,4,5,6,7} | Should pass all 4 assertions for each value; check that player's x coordinate is different to teleport tile, check that player's y coordinate is different to teleport tile, check that the difference between the x value of the player and teleport tile is greater than 300, and check that the difference between the y value of | Passes all 4 assertions for each value | Pass | |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | 300. | | the player and teleport tile is greater than 300. | | | |
| W-WC-TT-d | Test that player can move on normal tiles (Parameterized test) | GameLogicController, Map, Player | Get the list of normal tiles, for every normal tile, move the player to that tile and store that position. Move the player right for n iterations (n is passed in as 'iterations') and move player down for n iterations. If the player was able to move in the x direction (no walls) then check that the player's current x value is different to their original. If the player was able to move in the y direction then check that the player's current y value is different to their original. | Iterations = {3,50,100,200,700} | Should pass 1-2 assertions for each value (depending on whether player moved); check if player's new x position is different to their original and check that their new y position is different to their original y position. | Passes all assertions for each value. | Pass | |
| W-WC-TT-e | Test that wall tiles stop player moving (Parameterized test) | GameLogicController, Map, Player | Get the list of wall tiles, for each wall tile, check that the tile is actually a wall, check that the tile is not walkable and check that the wall tile type is not walkable. Move the | - | Should pass all 4 assertions; check that the current tile is a wall tile, check that the tile is not walkable on the map and check that all | Passes all 4 assertions | Pass | |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | player to a random normal tile, move the player in the 'direction' passed in as a parameter for 1300 iterations (player will hit a wall) and check that it will not allow the player to move again. | | wall tiles are not walkable. Check that player is prevented from moving when facing a wall. | | | |
| W-WC-TT-f | Test that player is moved on conveyor in right direction | GameLogicController, Map, Player | Get the list of conveyor tiles, for every conveyor tile, store the tile's position, move the player to the tile, store the player's position, tick the game so that the player is pushed along the conveyor. Check that the player was moved in the correct direction depending on the direction of the conveyor. | - | Should pass the assertion; check that player moved in correct direction depending on the conveyor direction. | Passes assertion for each tile | Pass | |

## Fixes as a Result of the Test Report

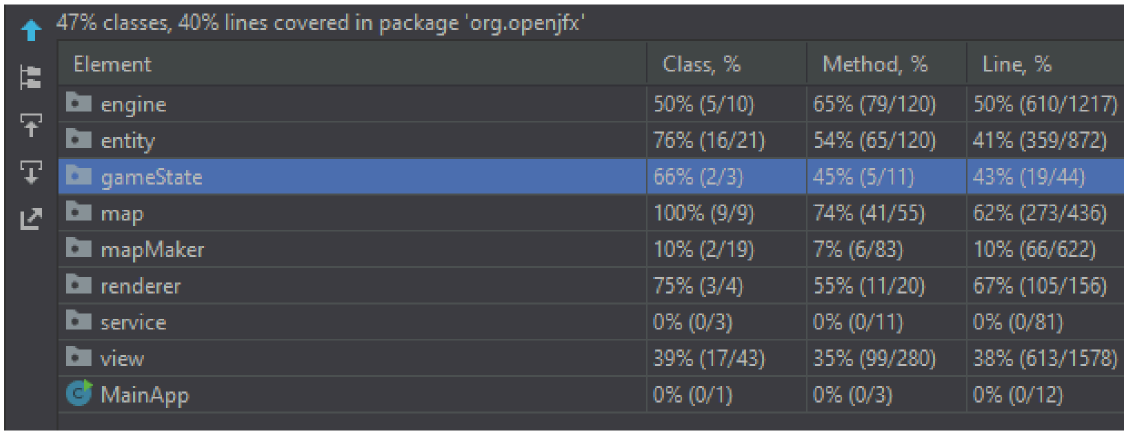
- W-WC-PT-v : This test actually failed because of a misunderstanding of the code, the unit test was checking for the wrong value and so the test failed even when it returned the correct value. Test was checking for 26 when it should have been checking for 25. Unit test is now fixed and it passes. Finding this was helpful in ensuring the game's values are better understood by the testers.

## Exit Criteria

95% of all test cases for functional and non functional features should be passed in order to meet the exit criteria, with no failed critical cases.

## Code Coverage

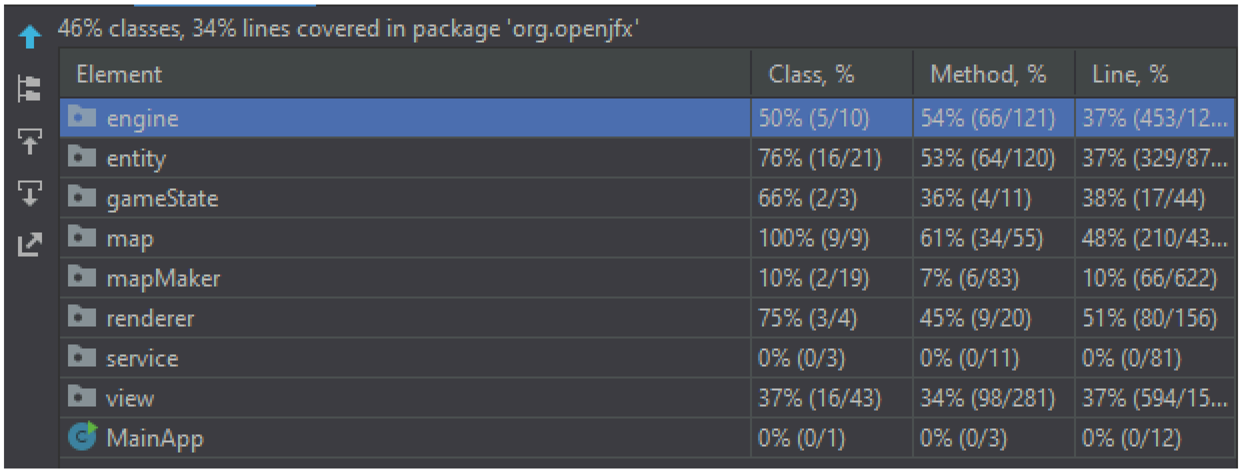
### ItemTest



47% classes, 40% lines covered in package 'org.openjfx'

| Element | Class, % | Method, % | Line, % |
|---|---|---|---|
| engine | 50% (5/10) | 65% (79/120) | 50% (610/1217) |
| entity | 76% (16/21) | 54% (65/120) | 41% (359/872) |
| gameState | 66% (2/3) | 45% (5/11) | 43% (19/44) |
| map | 100% (9/9) | 74% (41/55) | 62% (273/436) |
| mapMaker | 10% (2/19) | 7% (6/83) | 10% (66/622) |
| renderer | 75% (3/4) | 55% (11/20) | 67% (105/156) |
| service | 0% (0/3) | 0% (0/11) | 0% (0/81) |
| view | 39% (17/43) | 35% (99/280) | 38% (613/1578) |
| MainApp | 0% (0/1) | 0% (0/3) | 0% (0/12) |

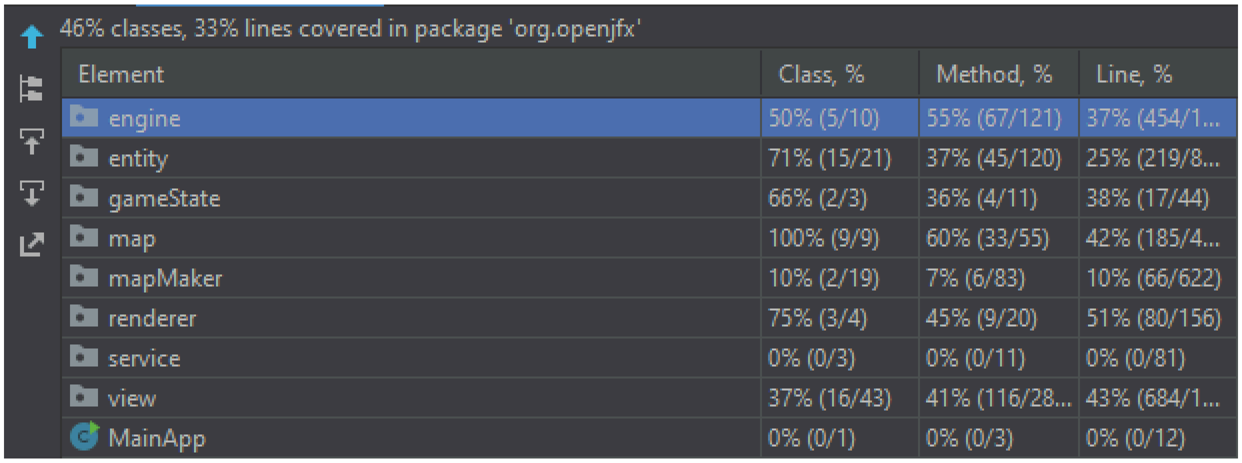
### PlayerTest



46% classes, 34% lines covered in package 'org.openjfx'

| Element | Class, % | Method, % | Line, % |
|---|---|---|---|
| engine | 50% (5/10) | 54% (66/121) | 37% (453/12... |
| entity | 76% (16/21) | 53% (64/120) | 37% (329/87... |
| gameState | 66% (2/3) | 36% (4/11) | 38% (17/44) |
| map | 100% (9/9) | 61% (34/55) | 48% (210/43... |
| mapMaker | 10% (2/19) | 7% (6/83) | 10% (66/622) |
| renderer | 75% (3/4) | 45% (9/20) | 51% (80/156) |
| service | 0% (0/3) | 0% (0/11) | 0% (0/81) |
| view | 37% (16/43) | 34% (98/281) | 37% (594/15... |
| MainApp | 0% (0/1) | 0% (0/3) | 0% (0/12) |

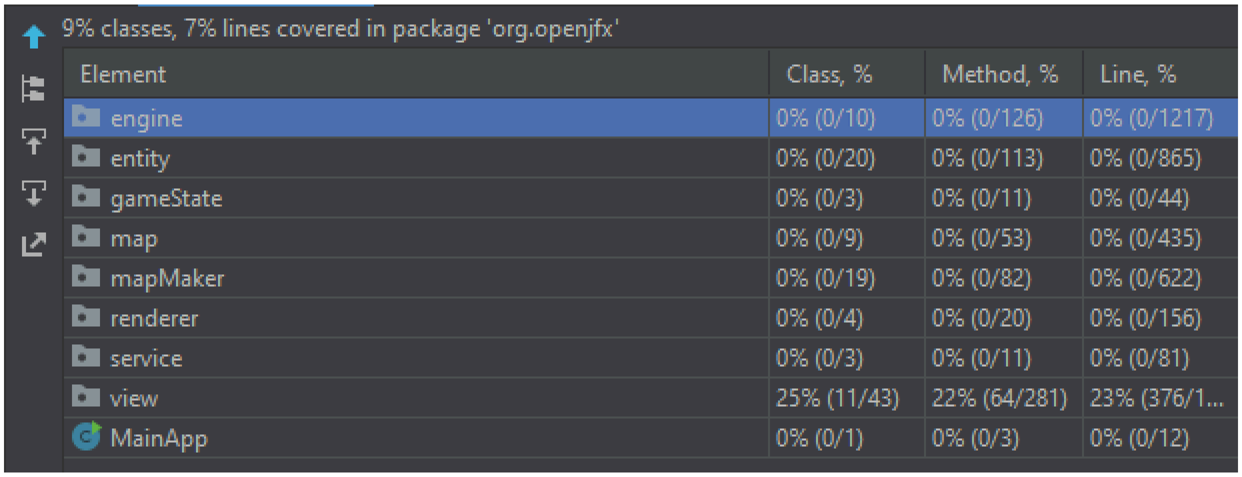
## PostGameUITest



46% classes, 33% lines covered in package 'org.openjfx'

| Element | Class, % | Method, % | Line, % |
|---|---|---|---|
| engine | 50% (5/10) | 55% (67/121) | 37% (454/1... |
| entity | 71% (15/21) | 37% (45/120) | 25% (219/8... |
| gameState | 66% (2/3) | 36% (4/11) | 38% (17/44) |
| map | 100% (9/9) | 60% (33/55) | 42% (185/4... |
| mapMaker | 10% (2/19) | 7% (6/83) | 10% (66/622) |
| renderer | 75% (3/4) | 45% (9/20) | 51% (80/156) |
| service | 0% (0/3) | 0% (0/11) | 0% (0/81) |
| view | 37% (16/43) | 41% (116/28... | 43% (684/1... |
| MainApp | 0% (0/1) | 0% (0/3) | 0% (0/12) |

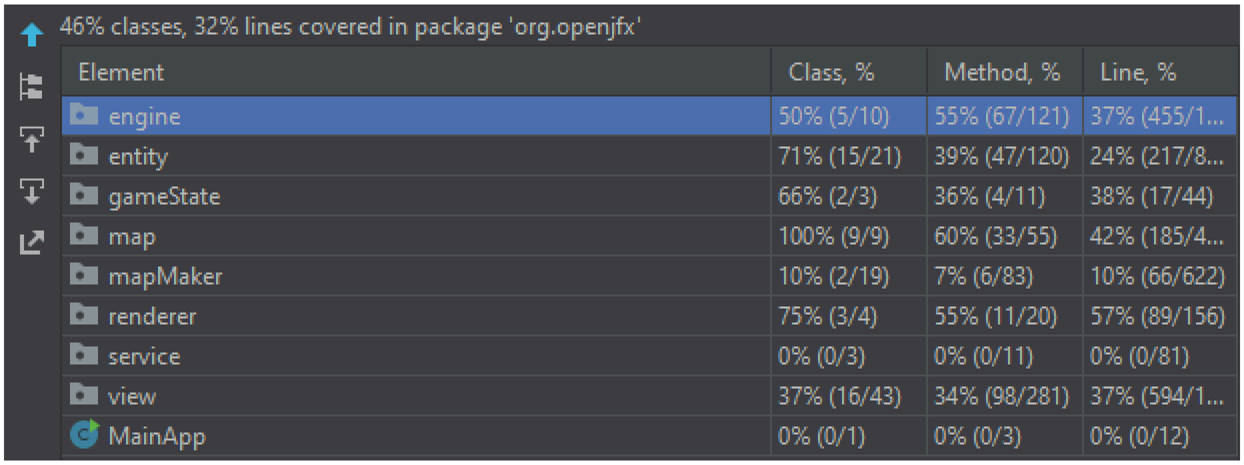
## PreGameUITest



9% classes, 7% lines covered in package 'org.openjfx'

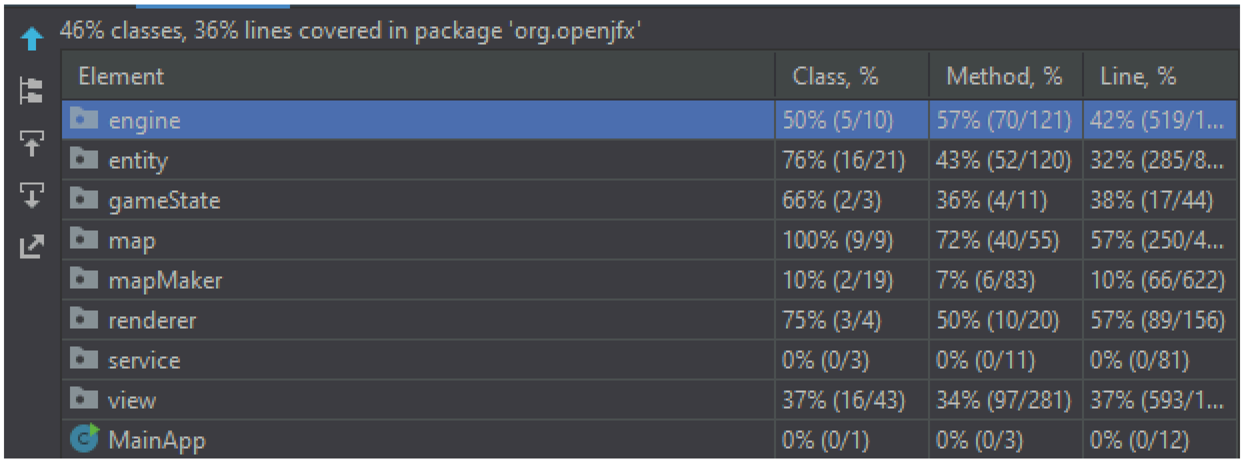
| Element | Class, % | Method, % | Line, % |
|---|---|---|---|
| engine | 0% (0/10) | 0% (0/126) | 0% (0/1217) |
| entity | 0% (0/20) | 0% (0/113) | 0% (0/865) |
| gameState | 0% (0/3) | 0% (0/11) | 0% (0/44) |
| map | 0% (0/9) | 0% (0/53) | 0% (0/435) |
| mapMaker | 0% (0/19) | 0% (0/82) | 0% (0/622) |
| renderer | 0% (0/4) | 0% (0/20) | 0% (0/156) |
| service | 0% (0/3) | 0% (0/11) | 0% (0/81) |
| view | 25% (11/43) | 22% (64/281) | 23% (376/1... |
| MainApp | 0% (0/1) | 0% (0/3) | 0% (0/12) |

## RendererTest


46% classes, 32% lines covered in package 'org.openjfx'

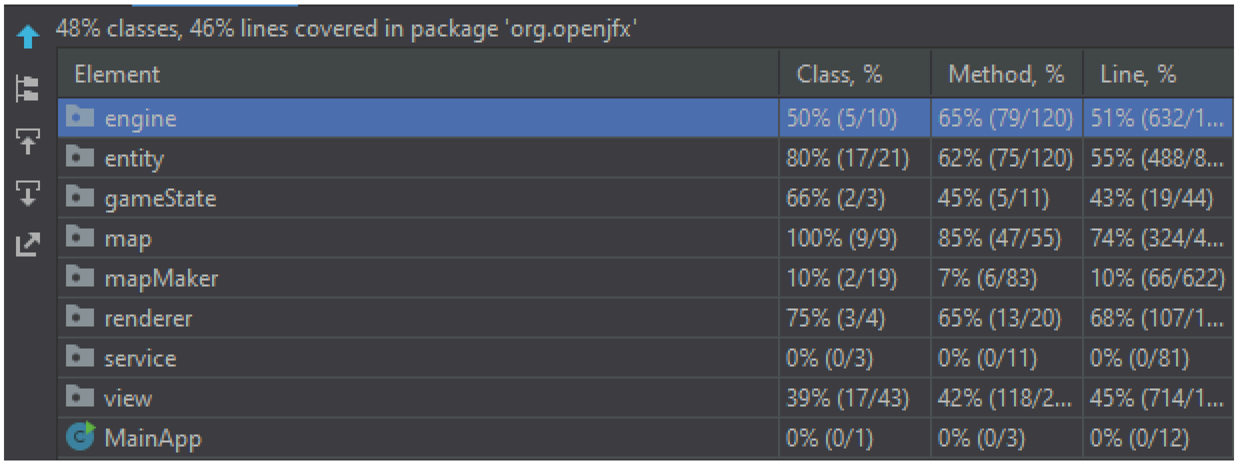
| Element | Class, % | Method, % | Line, % |
|---|---|---|---|
| engine | 50% (5/10) | 55% (67/121) | 37% (455/1... |
| entity | 71% (15/21) | 39% (47/120) | 24% (217/8... |
| gameState | 66% (2/3) | 36% (4/11) | 38% (17/44) |
| map | 100% (9/9) | 60% (33/55) | 42% (185/4... |
| mapMaker | 10% (2/19) | 7% (6/83) | 10% (66/622) |
| renderer | 75% (3/4) | 55% (11/20) | 57% (89/156) |
| service | 0% (0/3) | 0% (0/11) | 0% (0/81) |
| view | 37% (16/43) | 34% (98/281) | 37% (594/1... |
| MainApp | 0% (0/1) | 0% (0/3) | 0% (0/12) |

## TileTest


46% classes, 36% lines covered in package 'org.openjfx'

| Element | Class, % | Method, % | Line, % |
|---|---|---|---|
| engine | 50% (5/10) | 57% (70/121) | 42% (519/1... |
| entity | 76% (16/21) | 43% (52/120) | 32% (285/8... |
| gameState | 66% (2/3) | 36% (4/11) | 38% (17/44) |
| map | 100% (9/9) | 72% (40/55) | 57% (250/4... |
| mapMaker | 10% (2/19) | 7% (6/83) | 10% (66/622) |
| renderer | 75% (3/4) | 50% (10/20) | 57% (89/156) |
| service | 0% (0/3) | 0% (0/11) | 0% (0/81) |
| view | 37% (16/43) | 34% (97/281) | 37% (593/1... |
| MainApp | 0% (0/1) | 0% (0/3) | 0% (0/12) |

## Full Code Coverage



48% classes, 46% lines covered in package 'org.openjfx'

| Element | Class, % | Method, % | Line, % |
|---|---|---|---|
| engine | 50% (5/10) | 65% (79/120) | 51% (632/1... |
| entity | 80% (17/21) | 62% (75/120) | 55% (488/8... |
| gameState | 66% (2/3) | 45% (5/11) | 43% (19/44) |
| map | 100% (9/9) | 85% (47/55) | 74% (324/4... |
| mapMaker | 10% (2/19) | 7% (6/83) | 10% (66/622) |
| renderer | 75% (3/4) | 65% (13/20) | 68% (107/1... |
| service | 0% (0/3) | 0% (0/11) | 0% (0/81) |
| view | 39% (17/43) | 42% (118/2... | 45% (714/1... |
| MainApp | 0% (0/1) | 0% (0/3) | 0% (0/12) |

The reason for low code coverage in some packages is due to the fact that they often contain very simple methods which we thought would not need testing.

The testing for our project was done with the help of JUnit 5 and Hamcrest.

### Main Application:

- Main application running

### UI:

- Check the slider volume corresponds to the real volume
- Check that the checkboxes work as expected
- Check that the username is correctly bound to the username text field
- Check that the game timer stops when the game is paused

### Players:

- isValidMove()
- Passing the virus - single-target, multi-target
- Test current location is the same as graphics

### Tiles:

- Different speeds on different tiles
- Teleportation tile
- Different conveyor belt movement

## Map:

- Checking distance between two players.t
- Intersects

## DataCollector:

- Assert format of data is correct
- Assert every player location is valid (not out of bounds, not wall)
- Assert a JSON file can be processed for every game state (check on random examples)

## BotCommunicationThread:

- Handle sudden disconnections - assert exception is thrown
- Assert bot dies on disconnection
- Assert Python catches the same data (ping Python, resend the data to Java, compare)
- Assert the Python interpreter runs correctly (cross-platform testing, running test python files)

## Bot:

- Assert simple movement works as expected (compare positions before and after move)
- Assert

## MasterRenderer:

- Assert images rendered are all found.
- Assert images rendered are loaded within window bounds.

## JUnit Test Classes:

- ItemTest
- PlayerTest
- PostGameUITest
- PreGameUITest
- RendererTest
- TileTest
- MainAppTest - Intended to be used to run all tests at once

## Other Tests:

- UnitTests.py - Python unit testing

## Individual Contributions

The contributions of each member of our group regarding the tasks they contributed to can be seen from the table below. In addition to this, each team member's contribution is given as a percentage at the appendix of our report.

| Member | Tasks |
|---|---|
| Munirudeen Maricar | • Project initialisation.<br>• Prepared presentation.<br>• Gradle configuration.<br>• Introduction with Ben.<br>• Map design with Ben using Paint.<br>• occupyMap function to integrate the tiles in the map.<br>• Project requirements.<br>• Software Engineering Principles and enabling them.<br>• Project evaluation.<br>• Project summary.<br>• Project timeline maintenance (visualised using a Gantt chart)<br>• Teamwork analysis.<br>• Integrated JUnit and Hamcrest with Gradle.<br>• Code refactoring for project modularisation.<br>• MainAppTest to run all the tests in sync.<br>• Test report formatting help for Craig.<br>• Edge length calculation in A* algorithm (added on by Alex and Weronika).<br>• QTable State Saving and Updating (Pair Programming) with Ben and Weronika.<br>• Trained Q learning algorithm with support of Ben. |
| Vlad Dimitrie Florea | • CSS styling of the UI - together with Wonseok (all the background images were made and provided by Alex)<br>• UI logic<br>• Help pages in the UI<br>• PreGameSubScene - together with Wonseok<br>• Loading screen<br>• In game help page (shows when pressing F1)<br>• Audio and sounds - logic done together with Wonseok, sound files provided by Ben<br>• Round system logic + displaying the round number<br>• Items: silence arrow and quarantine item: idea + full implementation<br>• Abilities: teleporter ability: idea + full implementation<br>• FPS text and logic<br>• In-game scoreboard (appears while holding Tab) - together with Wonseok<br>• ConfirmActionSubScene |

| | |
|---|---|
| | ● Displaying each (human) player's inventory and cooldowns on screen<br>● GameOutcomeScene (together with Wonseok)<br>● Worked on GameViewController and GameLogicController (the class model was made by Alex)<br>● ScreenShotManager - together with Wonseok<br>● Testing (together with Wonseok): ItemTest, PreGameUITest, PostGameUITest, TileTest<br>● GameExecuteController (running the game without rendering, for AI training purposes) - together with Alex<br>● Specification document: risk analysis and interface design (together with Wonseok) |
| Craig John James Kerr | ● Contributed to original idea with a focus around AI<br>● GameViewController - with Weronika towards the start of the project<br>● HotPotatoException - simple exception<br>● Renderers package hierarchy design and implementation<br>● Original PlayerData, ItemData, TileData classes - for collecting data to send to AI<br>● LastGameState class - later became DataCollector after rework<br>● Data classes hierarchy<br>● JSON format idea for AI communication<br>● Refactored ItemType enum to store data within<br>● Test report with some help from Munir - test table, testing strategy, etc<br>● Provided feedback on introduction and risk analysis of specification document<br>● JUnit tests - RendererTest, PlayerTest and contributing a little to ItemTest and TileTest<br>● Python Unittest - testing of A* and Q Network foundations - help with formatting from Alex<br>● Black box testing<br>● Improved testability of code - adding optional and default parameters to classes/methods to allow easier testing |
| Maria Alejandra Rivas | ● Tile behaviour skeleton.<br>● Tile: slow, booster, teleport, conveyor behaviour on Map<br>● Items: Initial behaviour and initialisation structure skeleton<br>● Items: Freeze, Immunity.<br>● MapChange: Item behaviour towards map (excluding quarantine)<br>● Network: current version of network architecture modelled after Weronika's original version.<br>● Network: Python Client Structure - together with Weronika<br>● Map Maker Util Application<br>● Map: file retrieval and creation system.<br>● Render: Changeable sprites based on entity status<br>● DataCollection: Data points collection for game data except for teleport data. JSON format provided by Craig. |

| | |
|---|---|
| | • Data Collection: PlayerData, MapData, ItemData structure and relationship to DataCollector<br>• Players: Player initialisation, Original player structure, botAgent logic modelled after Weronika's original version, botAgent movement, botAgent communication, botAgent termination.<br>• Round System: RoundData saving entities and server between rounds (no bot termination), Saving training bots between episodes, maintaining bot connection between rounds and episodes.<br>• GameExecuteController (running the game without rendering, for UI training purposes) - together with Vlad<br>• GameExecuteController: Toggle to view rendered version, customizable rounds and episodes.<br>• GameExecuteController : SpecialAgent training - together with Vlad<br>• Game View Controller and GameLogic Factory model.<br>• Game View Controller: Render loop separation<br>• GameLogicController: Episodic score log, separation of rendering and logic dependencies.<br>• UI: DevelopersPreGameSubScene: UI and Logic, modelled after Vlad's and Wonseok's.<br>• UI: PlayerInformations Class and Collection logic, modelled after Vlad's and Wonseok's work.<br>• UI: All Game Image Assets/Artwork<br>• AI: Worked on aStar with weronika<br>• AI: data conversion and formatting (logInterpreter /Grid /Tile /TileNode) with Weronika<br>• AI: Q network saving and loading.<br>• Documentation: Software Design with weronika<br>• Documentation: UML diagrams figures 3s(Software Design) and 9s (Appendix)<br>• Documentation: Coding Standards and Glossary of terms. |

| | |
|---|---|
| Benjamin Norton | ● Handled Project skeleton and gradle configuration with Munir<br>● Introduction with Munir.<br>● Map design with Munir.<br>● Created prototype for movement within the game<br>● Created visual feedback for immunity, freeze item and teleporter usage<br>● Changed item spawning logic to minimise states within Q learning<br>● Refined movement logic including Conveyor tiles to iron out movement discrepancies of bots and players.<br>● Modified state structure of Q learning algorithm to minimise states due to players<br>● Worked on reward function of q learning algorithm with Weronika<br>● Created additional in-game sound mechanisms to handle audio<br>● Handled sound logic, created unique audio for the game.<br>● Trained Q learning algorithm with support of AI team<br>● prepared presentation<br>● Added UI features such as the BotInfo Page |
| Wonseok Choi | ● Settings view (in game and pre-game)<br>● Dash ability<br>● CSS styling of the UI - together with Vlad (all the background images were made and provided by Alex)<br>● PreGameSubScene (where the user player his username and the bots he wishes to play against) - together with Vlad<br>● Audio and sounds(Media Controller) - logic done together with Wonseok, sound files provided by Ben<br>● In-game scoreboard (appears while holding Tab) - together with Vlad<br>● GameOutcomeScene - together with Vlad<br>● ScreenShot Manager - together with Vlad<br>● CountDown in the game<br>● Score system and Score Board in the game<br>● Testing : ItemTest, PreGameUITest, PostGameUITest, TileTest - together with Vlad<br>● Specification document: risk analysis and interface design together with Vlad<br>● Implement DQN and A2C structure<br>● Modify DQN and A2C to communicate with Java application (with Weronika and Alex)<br>● AI Knowledge support : Did a lot of self-study for Reinforcement learning and Keras library: UCL-Deepmind lectures delivered by David Silver, CS234 from Berkeley, CNN Lecture(purchased), Attend RL study-group every week, Read 4 books : 3 books about RL and 1 book about Keras, etc. |
| Weronika Wiesiolek | ● Contributed to original idea with a focus around AI, |

| | |
|---|---|
| | combined group's ideas into a coherent concept<br>● Early identification of high-level architecture design<br>● Creating the first document and communications framework, facilitated software planning<br>● Created an MVP of the game engine - Game Logic Controller<br>● Refined Players, Entities, Map, and integrated into early GLC<br>● Created initial collision system refined by Ben<br>● Designed the Java-Python communication framework (PythonThread, changes in BotPlayers)<br>● Research different script execution methods and Jython<br>● Added Jython to project architecture and resolved dependencies<br>● Wrote a client for the socket communication - pyClient.py, pyclientML.py<br>● Created structure for the bot Python scripts: MainAStar, MainDQN, MainQNetwork, MainA2C, skeleton of LogInterpreter developed by Alex<br>● AStarSolver development with Alex<br>● Designing AStar improvements with Alex<br>● JSON data passing design and development (LogInterpreter, Java buffers)<br>● Library research and usage for data pipeline - pickle, ast.literal_eval, JSON<br>● QNetwork development - updating QTable, creating a state-action dictionary<br>● Wrote State representation in Python<br>● Game mechanisms adjustment based on the needs of AI<br>● Created saving and loading mechanisms for most Python files (loading save_model folder in DQN and A2C, saving memory buffer, saving scores in Python)<br>● Integrated initial DQN files with networks architecture into MainDQN, Environment, and DQN Agent<br>● Fixed ML errors related to architecture with Wonseok<br>● Researched AI architectures as seen in the bibliography<br>● Learnt RL principles from materials found by Wonseok<br>● Trained ML AI and conducted model refinements with Wonseok |

# Individual Reflections

The reflections of each member of our group after completing this project can be seen below:

## Munirudeen Maricar

Throughout the project, our group learned a lot and managed to create a full-fledged game by combining our skills. I benefited a lot from working in a group because I now know how to share responsibilities and collaborate more efficiently. Since our project used the Rapid Application Development approach, feedback from every group member was vital to the game's success. Therefore, my communication skills also improved. To specify, there was an issue where the difficulty setting in the UI was no longer necessary. After I brought up the case, the team agreed that we could remove the setting and instead indicate which AI algorithm is playing the game instead, bringing consistency to the game.

However, there were also challenges along the way, namely, incoordination for meetings, unequal workload distribution, etc. For instance, there were times during Ramadan (specifically after the Easter break) where I wish I contributed more to the project. There were also times where I found myself without any tasks to complete, where I could have stepped in and helped out other members of the team. As a result, there were communication hold-ups and extra stress for other members of the group.

Although the group project resulted in a great learning experience, I would like to have worked more on the AI aspects of the game. Some of the AI architectures implemented in the game were unfamiliar to me. Therefore, I would like to have done more research on them, even if I was not involved in their development. Additionally, I would like to have worked more in terms of the UI of the game. JavaFX was also foreign to me before the project. However, I could have researched more about it and helped out with the UI development.

To summarise, even though the group project is a success, there were situations where hiccups were present in my views. Regardless, my team members were highly supportive and helped one another most humbly. After all, it was a great learning experience for me.

## Vlad Dimitrie Florea

Overall, working on this project was a pleasant experience mainly thanks to Weronika, Wonseok and Alex, people with whom I would gladly work again on any other project. I also wish to mention that meetings in which an individual didn't participate for justified reasons (and announced the group before the meeting) are not counted. However, I personally do not accept excuses like "I forgot about the meeting", especially if it happened more than once. Here are my personal thoughts about each member of our team:

Wonseok Choi:

I have directly worked with him on many aspects of the User Interface and it was an amazing experience, mainly thanks to his ambition, seriousness and reliability. He also actively participated in the game and came up with a lot of good ideas which were implemented in the final game, some of which being: the round system, the score system and other AI-related ideas. Furthermore, he has done extensive research about what kind of AI we can use in our game and how exactly we can implement it (and also worked on the implementation of DQN and A2C bots). On top of this, he has participated at every group meeting.

Weronika Wiesiolek:

She was our team's leader, she was the one responsible for dividing the tasks between our teammates and leading the flow of the conversation in our group meetings. I firmly believe she did a fantastic job at leading our team and I'm glad we had such a strong leader. Not only was she a remarkable leader, but she was also a great teammate, doing a lot of fantastic work, especially regarding the AI component of our game. All in all, there is nothing bad to say about her since she is the kind of person one can totally rely on. She has also participated at all of our group meetings.

Maria Alejandra Rivas:

She was always keen on taking on lots of tasks while still meeting the deadline for completing those tasks. Moreover she was very hard-working, ambitious, quick and efficient at doing her work. Nothing bad to say here as she was really serious about the project, always did her tasks properly and participated at every meeting.

Munirudeen Maricar:

Overall, I wasn't very happy to work with him mainly because of his lack of seriousness and will to work hard. He has missed some of our group meetings and checking by his git commit history, he also hasn't done a substantial amount of work (like some of our other team members did).  Another reason why I wouldn't willingly work with him again in a future project is because he kept finding excuses. One such example, which I personally find unacceptable, was telling the whole group that he couldn't do anything for one full week because he was "busy with the Easter chaos", a Christian holiday which he doesn't have in his calendar. However, he did good work related to the document, presentation and git merges.

Benjamin Norton:

I will divide this into 2 parts: before and after the END of the holiday. Before the end of the holiday, he did an extremely little amount of work (judging by his git commit history) and he also missed quite a lot of our group meetings, having a low involvement overall. However, after the holiday, it was like working with a completely different person: he started to bring good ideas, to take on many tasks and do them properly, he didn't miss any group meetings and he showed lots of involvement, but this only happened for 4 weeks, after the holiday.

Craig Kerr:

Personally, I would not willingly work with him again in a future team project, because I believe he lacks ambition, seriousness and he often didn't meet the deadlines for his tasks. He missed some of our group meetings and judging by his git commit history, the amount of work he did is rather low. Furthermore, he had many periods of inactivity and also made unjustifiable excuses (like being stressed). However, he did good work regarding the Unit testing part (although he wasn't the only one working on it) and he also did an excellent job at writing the test report in the final document.

Reflection of myself:

I believe I showed lots of seriousness throughout the project by always completing my tasks by the end of their deadline and participating at every group meeting. I have primarily worked on the UI, but I also worked on implementing some of the items, as well as on testing. Furthermore, I have actively participated on the project by constantly working on some task, without any periods of inactivity. As for what I could have done better, I think I should have tried to directly help with the AI component by acquiring more knowledge regarding the AI architectures we used. This project has been a great opportunity to learn many new things ranging from raw programming skills to teamwork related skills like communication. Although some team members have disappointed me at times, I believe it could have been worse. Everybody in the group definitely had their own contribution, either by doing more programming or by coming up with more ideas. It can also be that my disappointment towards some of the others is because I had overly high expectations, which might be something I need to work on in the future. Nevertheless, I had something to learn from every team member and I'm grateful that I was given the opportunity to work in a team environment.

## Craig Kerr

During this semester while working on 'WannaCry', our team has grown a lot. We have all gained a better understanding of how to work in a team with ever-changing ideas, problems, and goals; we have now got more experience with working on a large code base in a group, which will help those of us wanting a future in software development.

This project helped me gain a better understanding of real-world code practices. I worked a lot on the testing of the game, and it made me realise how vital it is to create unit test classes before working on a component of a product.

Our group has worked well as a team, having 1-2 meetings per week, allowing us to all keep on track and let us all have something to contribute to. Regarding myself, I struggled with motivation sometimes, some of the code I worked on overlapped with code others were writing, and so I found it difficult to see something as 'my own', resulting in lower motivation at some points for myself.

There were many challenges throughout the development of the project; constant idea changes and the discovery of bugs sometimes meant that parts of the codebase needed reworking almost entirely. This was a setback for us during development.

In the project, I worked on the base game engine features, like the view controller and renderer classes towards the beginning of the project. I also worked on utility classes, like the data collector classes used to collect data which will be sent to the Python section of the program. As mentioned previously, I worked a lot on the testing of the code, I wrote JUnit tests for the Player class, Renderer class, and contributed to the Tile and Item test classes.

In conclusion, if I could start over, I would have been more assertive with my concerns about working on the same parts of the code as other people, since this affected my motivation. I also wish I worked more on the AI than I did, since it is an area of interest for myself.

## Maria Alejandra Rivas

Over the past few months, our team worked on developing 'Wanna Cry'. The development has tested not only our understanding, knowledge of AI and Programming; but also the qualities needed to approach a shared project with multiple developers with very many different ideas and approaches to such. There is no doubt that this experience has personally benefited me including, improving upon interpersonal skills, motivation and self-regulation.

Concerning development: this project has helped me develop context and experience for a large ongoing repository, more rapid and better coding practices, and encouragement to approach things differently as I would normally.

However, this does not mean there weren't challenges. Balancing an equal share of the workload, keeping track of the tasks completed, and many keyboards slammed from seeing the code that worked great in your machine completely break and display the dreaded wheel of 'no response'. Nonetheless, these have not been challenges without reward, for I can say that the experience was needed and appreciated.

To be specific, an essential factor in solving these was maintaining open and timely communication. No one was unresponsive or late when answering any queries or clearing up some issues. The team members made it very comfortable to ask them about and discuss ways of solving the encountered problem.

I'd like to say that I contributed by volunteering to do any 'unwanted tasks' in addition to whatever I had assigned at the time. Although I may have spotted my fair share of bug, taking in bug reports from my peers and applying a solution was also something I took part in.

If I were allowed to start over, I would put more emphasis on planning and deadlines. Having a steady workload output would have helped lessen some struggles. In addition, it would have also reduced hour-long coding sessions into manageable ones. It would have also been dice to properly manage the ever changing course of the project. Constant new ideas where being brought up which sometimes meant scrapping preexisting framework and starting anew. We didn’t manage to develop our DQN and A2C bots as much as we would have liked, this could have been caused by their relatively late inclusion on the project. Given the chance to start anew, I would have push further any worries or suspicions rather than keeping them quiet due to a lack of trust in my own judgement.

Despite all this, the group did deliver a product we can be proud of.

## Benjamin Norton

At the beginning of the semester, me and my team were tasked with creating a game using the Java language from scratch, being an AI based project, the main complexity of our game we set out to implement was to be in it's array of AI architectures.

During the inception of the game idea, I felt like I was an important voice in creating the vision of the game. Once we had chosen an aesthetic for Wannacry collectively I felt strongly about how the game should look and took strides to make sure it stuck to it's Digital aesthetic.

When it came to implementing the functionality of our game however, I encountered a difficult learning curve, the complexity of different features in the game in conjunction with each other made adding my own features quite a daunting task as I didn't want to disrupt the work of others. But when I took the time to talk through my plans with the group they were incredibly receptive and gave me as much help as they could to get me started.

With that initial difficulty in mind, I can confidently say that my contributions to the game were important additions. I enjoyed my time reworking legacy code to be more robust such as the movement aspect of our game, work that inadvertently fixed alot of our AI teams issues; once they implemented their work with my changes it helped address alot of their problems. My peer work on design elements such as the map with Munir or talking ideas on how to improve the Q networks functionality with the AI team are just a few examples of time that teamwork was crucial to finish this project.

Though I feel like my work was integral, some of my additions I made to the game were either in new ideas I brought to meetings or working out bugs in the code that someone else had implemented. So sometimes I felt as if my work was overshadowed by others.

Working on Wannacry was my first project I have worked on as a team. I am incredibly proud of the progress I have made as a software developer, and I feel I have gained vital experience in taking a project to fruition with peers of different areas of expertise.

What I would do differently.

Were I to start a project like this again, I would try harder to immerse myself in the different facets of the development process. I found at times that I would only be of use to the team when working on specific aspects of the game. I would try harder to collaborate on parts of the game (networking, complex AI structures, UI) to gain a broader understanding of all the game parts in motion.

## Wonseok Choi

As a 'Wanna Cry' team member, I have studied and learnt so many skills. I've studied not only Reinforcement Learning but also communication skills and the whole development process from designing to AI training.

A week before the start of the second semester, I started to study reinforcement learning for the team project. At the beginning of the team project, me and Vlad mainly worked on the UI and presented many ideas for the game. We did screen sharing almost every day for discussing the game. After completing the UI, I originally intended to create the new AI architectures, but I realized we need to implement other things in order to define the MDP otherwise we would have to change the architectures whenever we add or change features . Vlad and I went through almost all the code, and created new abilities, items and tests for each error we found. The Easter vacation started and I only focused on studying Reinforcement Learning because I wanted to make AI bots that act like humans. After the Easter vacation, I mainly focused on DQN and A2C, which were created using the Java-python communication that Weronika and Alex worked on. There were many difficulties in making these models, but we were able to make training them possible in the end.

I always have regrets about things that have happened. However, I have more regrets than usual in regards to this project. As I've written, I have put so much of my time into studying reinforcement learning. To be perfectly honest, the time I put into studying for the ai2 and security and networks modules does not even compare to the time i've spent on this. Because I studied on my own, I've researched many different lectures and read books. But looking back on it now, I feel that I should have studied together with another team member as well. Even though I did say we should study together, I feel that I could have suggested it a bit stronger. Studying alone delayed starting implementation of DQN and A2C significantly. I have had to fix the game so that reinforcement learning could be applied but studying alone prolonged the time it took to do this. Another regret is that we, as a group, should have delegated the more tasks to group members who didn't work hard. Three members were not doing much in our group, so seeing that the java-docs were not made yet, I thought that they would continue to not work (which actually happened until the end). If I hadn't done anything else, I would have made the architectures for AI bots earlier, started training earlier and experimented more and made AI bots that perform better(using CNN). The biggest regret I have is the insufficient training of the bots. Thanks to Weronika, Alex and Vlad, I could try to train the AI bots at least.

I 've learned a lot of things through this project.Firstly, I am very happy to have found such an interesting field of research, namely, reinforcement learning. Secondly, it's my first time having participated in this kind of large scale project and I feel that I have developed a lot from the beginning to the end of this project. Using this experience as a foundation, I feel that it was very fun to actually implement the ideas and designs that I came up with. Lastly, having discussions with others and working together on the same program has really developed my communication skills and again I'd like to thank Vlad, Weronika and Alex for working hard with me. I've written in more detail about these group members in the individual contributions part.

In summary, this project has left me with a lot of regrets. My aim was to create an AI that behaves like a human playing, and I'm disappointed that the time that I've had on it has not been enough. However, I am very satisfied with the skills I've developed through this project, and I will work hard to treat this experience as a foundation on which to achieve better results in the next project. I have worked significantly harder than what was my responsibility and have done more work than my fair workload. Therefore, I have no regrets about the effort I've put to this project.

## Weronika Wiesiolek

As a newly added team member, I was a bit hesitant how the cooperation of the team would look like. Firstly, this was my first software project of this size, and secondly, I came to the team later when all the other team members had already had contact with each other. Luckily, I was greeted by a team of enthusiastic peers who seemed to be eager to cooperate and full of ideas.

During the initial discussions, I was very interested in hearing everyone's ideas, and tried to form a coherent core concept around everyone's expectations. A lot of different games were mentioned, including the previous main candidate, which was a fighting game, but eventually we settled on a 2D action game with a strategic element. I contributed to the selection process by pointing out the main advantages and disadvantages of each idea and refining the selection pool based on teams' interests and wishes.

As we have started working on the design of the game, I naturally started to moderate some of the discussions relating to high-level architecture. I also helped in task division and establishing short-term goals. It was a priority for me to have a strong and scalable architecture idea which will allow for modular work. Everyone was very clear in their expectations, so after establishing the overall direction, I helped in drafting key goals for our Minimum Viable Product components, and created an early version of the main engine. This taught me a lot about Software Development, and taught me I really enjoy conceptual software design.

At further stages I have progressed to establishing core physics functionalities. Afterwards, I was proud to have merged the first few advancements of all group members into a coherent game. It is crucial to stress the efficiency and ease of cooperation in our team at that point – everybody was really excited to see a (very provisional) working game, and hence contributed as well as they could and did all the tasks on time.

Around Week 4 I moved on to working on the Java-Python communication. Thanks to other team members who took care of developing the game mechanics and UI after discussing the TODOs with us I could focus mostly on the robustness of the communication and viability of high-speed AI calculations. The creation of the servers as they are now would not be possible without Alex – her role in developing the server-client threads, and the data passing system was crucial for their success.

Later on we have progressed to AI development aside polishing other game aspects. Me and Alex have started designing the Python framework. Although the overall process and outcome were satisfying, due to the speed at which everything happened it was difficult to get other team members involved as much as us, as the communication took more than just doing the tasks ourselves (I later learnt about similar problems for Vlad and Wonseok). I believe this was my error to a large extent, as the speed of our improvements may have been overwhelming at times. I intend on improving that in my further projects by clearer communication and involvement of more team members.

As for the AI itself, I believe we have learnt very much during the project. Starting from the A* was a great idea, which ensured we have a working bot in the final project. Later on, when we have started the other architectures as well, things have become quite complicated due to the unsatisfactory effectiveness of Q-learning training. This is why we have divided the tasks into refining A* to ensure top quality gameplay, and investigating alternative behaviours to gain knowledge. The ML bots, albeit not perfect, have acquired emergent behaviour such as mimicking the A* when hiding, using teleporters, or blocking each other at 2 opposite sides of the obstacles, which was very exciting to see. I did my best to contribute to this part of the project and study RL as much as I could, however, more initial knowledge would be needed to improve our models. I believe I could have improved my AI knowledge before the project, or dedicate even more time to experimentation.

Overall, I think the project was a success, and a great learning experience for me. I had a chance to cooperate on an ambitious project, and develop substantial skills in teamwork, AI and game development. Despite all the challenges we have encountered, we have managed to come up with alternative solutions and ensure our goals are met in the end. I am thankful for my teammates to contribute in a meaningful way to the project, and mutually enhance our skills.

# XI. Bibliography

[Software Design] Refactoring Guru, *Refactoring Guru*. *Design Patterns.* (2021). [Available at: https://refactoring.guru/design-patterns. Accessed: 29 April 2021]

[Software Design] Younis, Fares & Sallabi, Omar. (2018). *Dependency Injection (DI) in Java.*

[Software Design] A. C. Ojha, S. K. Pradhan and M. R. Patra. (2007). *Pattern-Based Design for Intelligent Mobile Agents.* 2007 Innovations in Information Technologies (IIT), 2007, pp. 501-505, doi: 10.1109/IIT.2007.4430411.

[Software Engineering]

[Software Engineering] *Manifesto for Agile Software Development*

[Software Engineering] Rapid Application Development

[Coding Standard] Google.github.io. n.d. *Google Java Style Guide.* (2021). [Available at: https://google.github.io/styleguide/javaguide.html. Accessed: 11 May 2021].

[Coding Standard] Programmers at work : interviews. Series 1. (1986). Redmond, Wash.: Microsoft Press.

[AI]  Amado, L., Meneguzzi, F. (2018). Q-Table compression for reinforcement learning. The Knowledge Engineering Review, 33, E22. doi:10.1017/S0269888918000280

[AI] Barto, A., Sutton, R. *Reinforcement Learning: An Introduction.* (2018).

[AI] Dilts, M., Muñoz-Avila, H. (2010). *Reducing the Memory Footprint of Temporal Difference Learning over Finitely Many States by Using Case-Based Generalization.* Springer Berlin Heidelberg. Berlin, Heidelberg. doi:10.1007/978-3-642-14274-1_8

[AI] Franca, I., Paes, A., Clua, E. (2017). *Learning How to Play Bomberman with Deep Reinforcement and Imitation Learning.*

[AI] Mnih, V. et al. (2016). *Asynchronous Methods for Deep Reinforcement Learning.* ICML 2016.

[AI] Park, S.  (2020). *Reinforcement Learning Principles and Algorithms Solved by Mathematics*.

[AI] Roh, S. (2020). *Reinforcement Learning From Scratch.*

[AI] Rosenfeld A. et al. (2017*) Speeding up Tabular Reinforcement Learning Using State-Action Similarities.* Proceedings of the 16th Conference on Autonomous Agents and MultiAgent System, 2017, pp. 1722–1724. São Paulo, Brazil.

[AI] Son, M.  (2019). *Deep Reinforcement Learning Algorithms Basics*.

# XII. Glossary of Terms

| Term | Meaning |
|---|---|
| DQN | Deep Q learning Network |
| A2C | Advantage Actor Critic |
| Players | Participant in a game (ai and human players) |
| Bots | AI participants in a game |
| Human Players | Human participants in a game |
| Entity | These refer to: Players and Items in a game. |
| World | Encompases the map environment and tiles inc. behaviour |
| Network | Referring to the networking infrastructure between java and python. Java holds the server and Python the Clients. |
| Round | Refers to a session played bound by the initial spawning to the moment the virus lifespan expires (i.e. The player dies from the virus) or the virus is successfully transmitted from the initially infected player to another. |
| Game | Refers to a session played through multiple rounds until every player present in the world has started a round infected. |
| AI Team | Refers to the members of the development team in charge of AI: Weronika Wiesiolek, Wonseok Choi, Benjamin Norton, Munir Maricar and Maria Alejandra Rivas. |
| Testing Team | Refers to the members of the development team in charge of testing: Vlad Dimitrie Florea, Wonseok Choi, Benjamin Norton, Munir Maricar and Craig Kerr. |
| BotAgent | The entity that executes a bot's movement and behaviour. |
| BotParent | Entity that talks to and listens to the socket for a bot of specific behaviour |
| Bot behaviour | Refers to whether the bot is rationalizing acts under an A Star Algorithm, Q Network Reinforcement Learning Algorithm, Deep Q Learning Neural Network and Actor Critic Algorithm. |

**[END OF THE DOCUMENT]**

**[THIS PAGE INTENTIONALLY LEFT BLANK]**