What Tiles, Mr. Wolf?

ESC Group 11 Cohort 2 Final Report

Jun Neng (1002286) | Kimberlyn Loh (1002221)
Thomas (1002168) | Hao Wen (1002497)

Table of Contents

1. Project Overview

1.1 Project Description

What Tiles, Mr Wolf? is a territorial acquisition game where players are to convert as many tiles as possible to their own colour. However, there is an AI (wolf) on the map which rotates randomly to detect players' motion. Player must stop moving when detected by AI or 5 of his/her tiles will be converted to his/her opponent. To make things more challenging, the timing for the wolf is not fixed but rather randomised. Players can occupy a tile by stepping onto opponent's coloured tile and the tile will be converted to the player's colour. Power ups will spawn at random and collecting them will increase the player speed. The game ends in 50 seconds and the player with the most number of tiles wins.

1.2 Game Mechanic

1.2.1 Power Ups

Power ups are randomly spawn across the map to allow players a chance to increase their current speed and convert more tiles quicker. This aid in making the game more interesting as the player who might be losing at the start will have a chance in reverting the situation. It also makes the gameplay more exciting as the players will have to consider the option of speeding up as part of their strategy. An indication will appear on top of the player when the power up is in used.

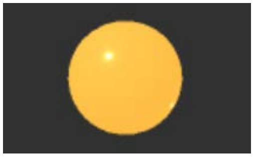


Figure 1.2.1.1.
Power Up Prefab

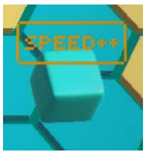


Figure 1.2.1.2.
Speed up indication

1.2.2 Wolf Timing

By implementing the wolf turning penalty, players will have to anticipate when the wolf be turning and stop in order not to lose any tile to their opponent. The gameplay becomes more interesting as the wolf timing is randomised and not fixed. This allows players to not just focus on the tile collection but also the possibility of losing tile. In addition, to balance the game, the wolf would 'vibrate' on the spot as an indication of it turning soon. This allow players to decide on whether they should prepare themselves to come to a stop or take the risk of collecting one more tile.

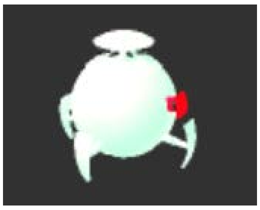


Figure 1.2.2.1. Wolf Prefab

1.2.3 Penalty Alert System

As the game play becomes more exciting, players may be oblivious to the fact that they are given a penalty when they move while the wolf is facing them. Thus, the implementation of penalty alert system was done. The player will be notified of the penalty by

(a) Vibrating of the phone
(b) Sound Effect of penalty
(c) Tile convert will fade into opponent colour.
(d) Deduction point indication

1.2.4 Player Indication

To avoid players' confusion in starting the game, an indication is placed above the player's cube as shown in Figure 1.2.4.1

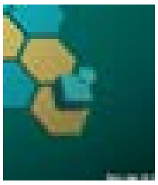


Figure 1.2.4.1 Player Indication at Start of Game

1.3 Game Flow

The game flow can be observed in 2 different scenarios: from the server and from the client. Both scenario will start from the menu scene and end to the end scene as shown below. In addition, both players will observe a start scene before the game begins. This is to allow both players to prepare for the start of the game. After 5 seconds from displaying the end scene, the game will revert to the start scene for the player to continue challenging other players.

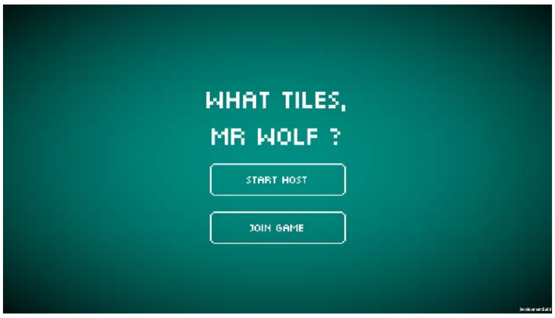


Figure 1.3.1. Menu Scene

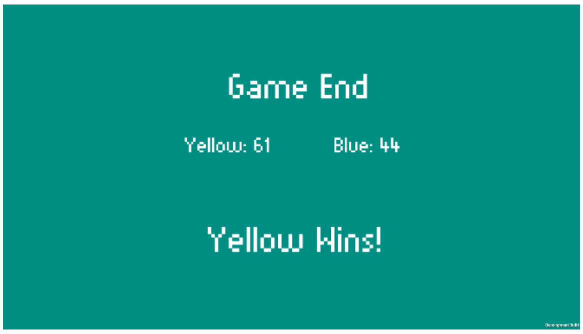
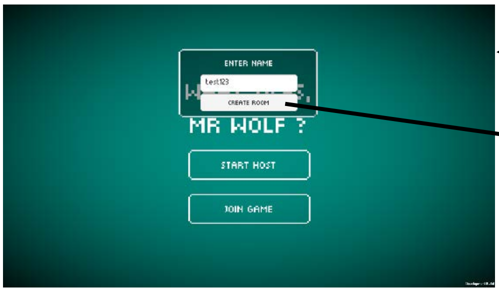


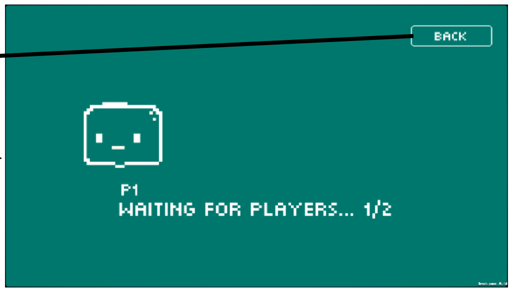
Figure 1.3.2. Menu Scene

Figure 1.3.3. Start Scene

1.3.1 Hosting the Game (Server)

The figure number will indicate the normal flow of gameplay and arrows on the figure indicate the event if the button was to be pressed. The back button implemented, as seen in Figure 1.3.1.2, is a way for the player to quit hosting the game if it is too long to find an opponent and join an open room instead. Once the game end, it will head to the end scene to display the results.

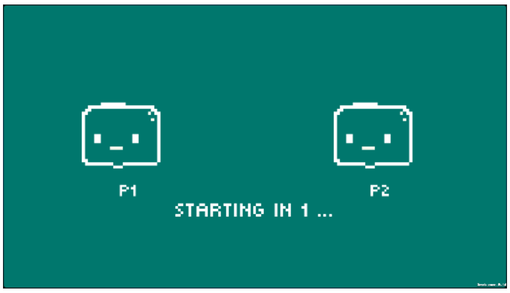




WAITING FOR PLAYERS... 1/2

Figure 1.3.1.1. Create Room

Figure 1.3.1.2. Waiting opponent



STARTING IN 1 ...

Figure 1.3.1.3. Opponent arrive

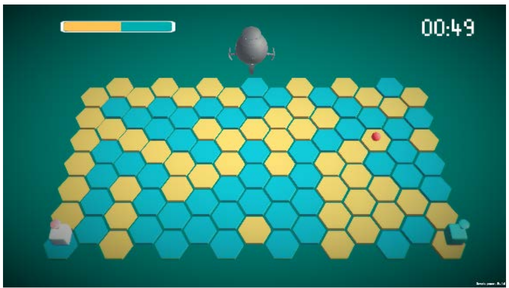


Figure 1.3.1.4. In Game Scene

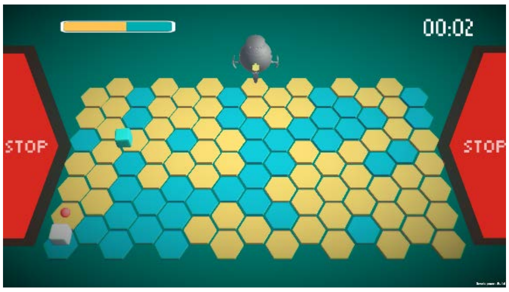


Figure 1.3.1.5. Penalty Scene

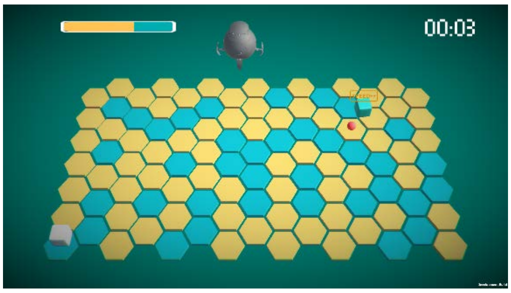


Figure 1.3.1.6. Power Up Scene

1.3.2 Joining the Game (Client)

The figure number will indicate the normal flow of gameplay and arrows on the figure indicate the event if the button was to be pressed. Once the game end, it will head to the end scene to display the results.

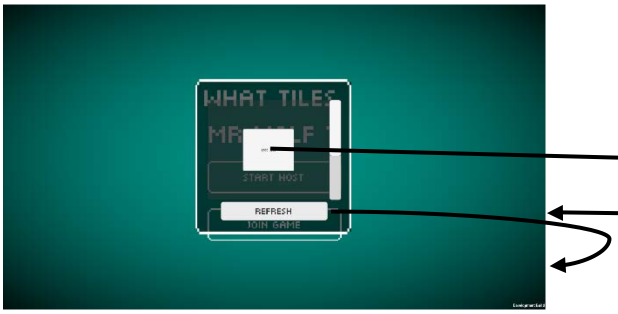


Figure 12. Room Lobby

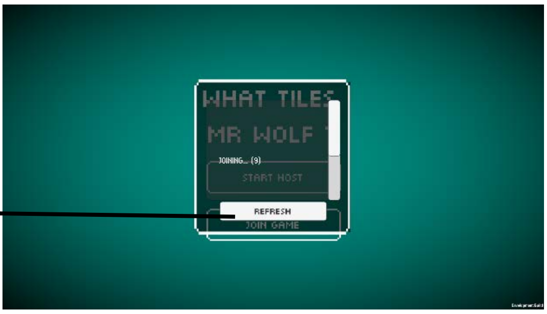


Figure 13. Joining Room

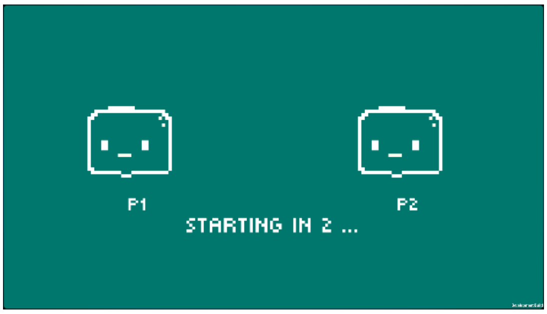


Figure 14. Confirm Room Join

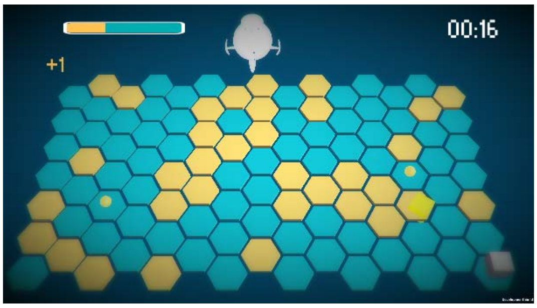


Figure 15. In Game Scene

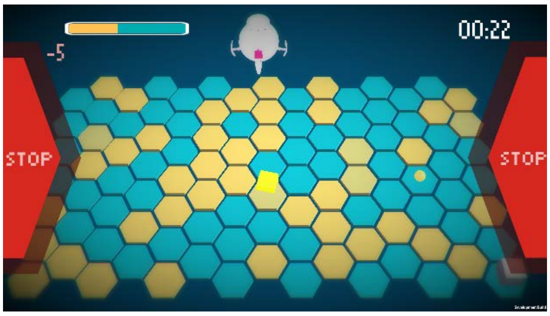


Figure 16. Penalty Scene

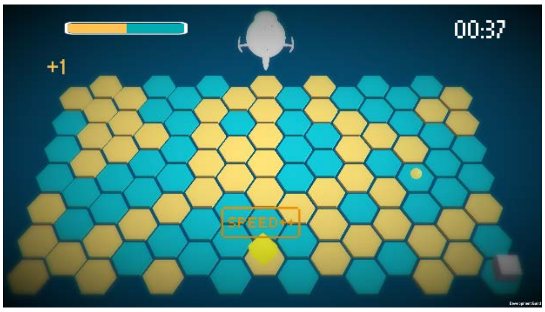


Figure 17. Power up Scene

2. Project Requirement

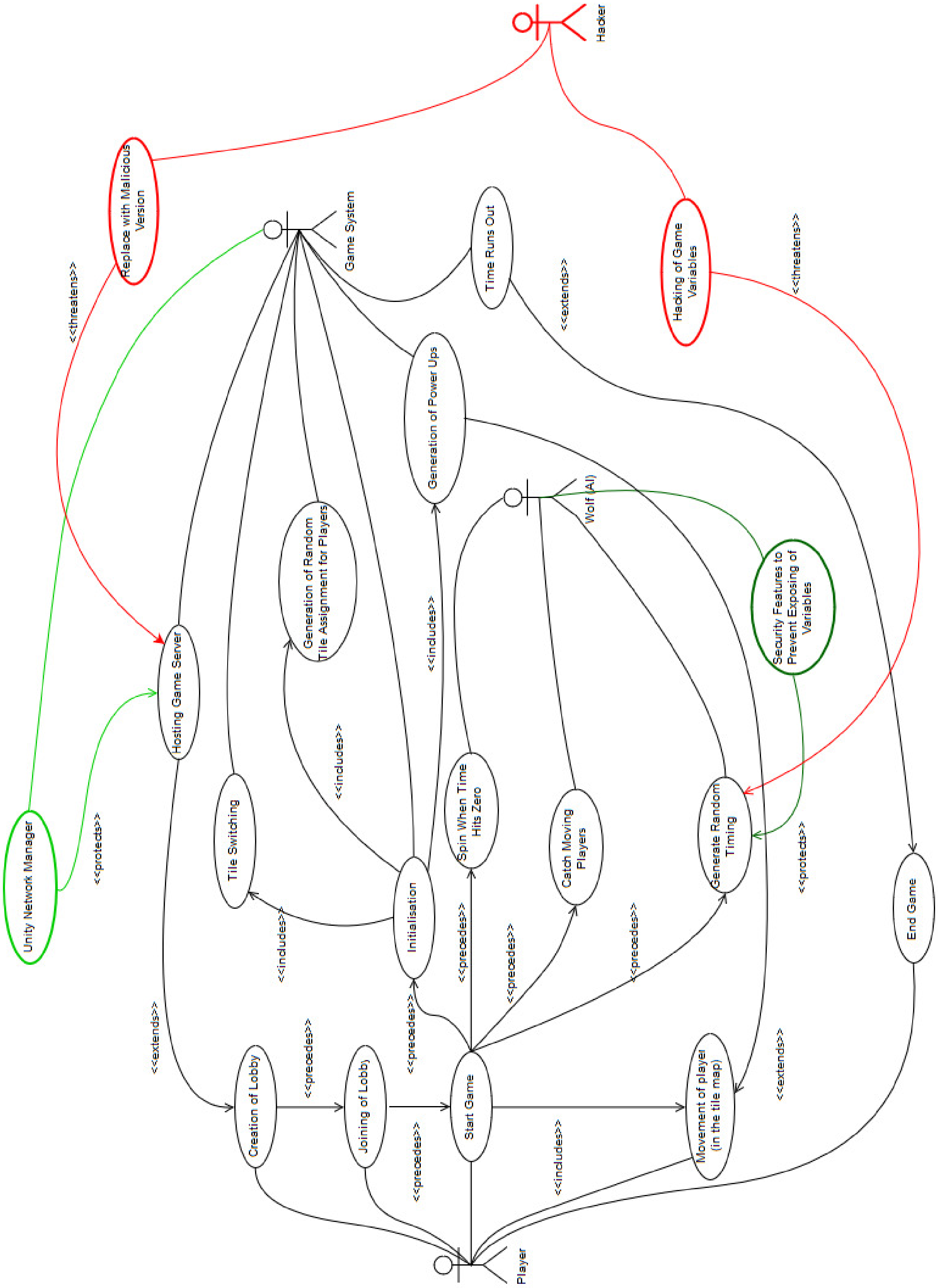


Figure 2.1. Use Case Diagram

3. Project Design

3.1 Visitor Design Pattern for Player

Whenever a player unit enters a tile, a different interaction will occur depending on the kind of player. A visitor design pattern is implemented to confirm the identity of the unit that enter and will call the correct behaviour accordingly.

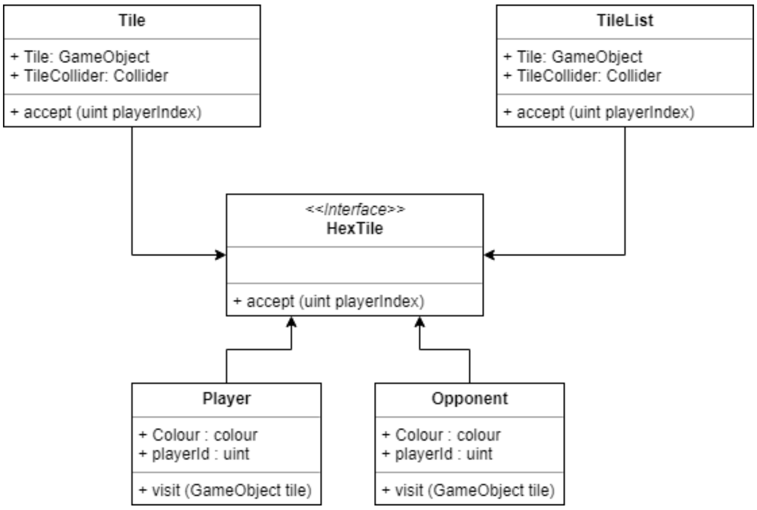


Figure 3.1.1. Visitor Class Diagram

3.2 Factory Design Pattern for Game Objects

Many of the game objects have the same class and types but differ in certain variables and functions. A factory design pattern is implemented to generate multiple objects of the same class, while overriding certain parameters accordingly to its type and uses.

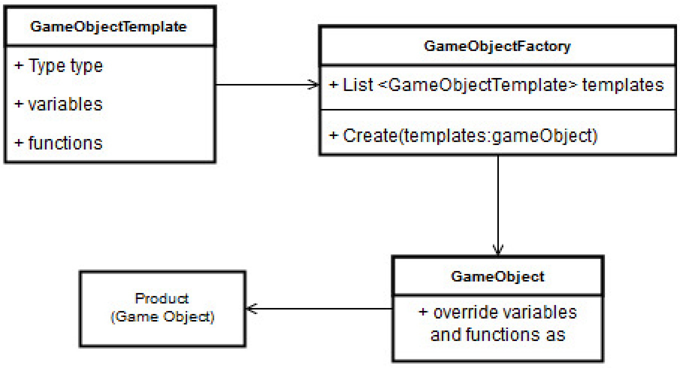

GameObjectTemplate
+ Type type
+ variables
+ functions
GameObjectFactory
+ List <GameObjectTemplate> templates
+ Create(templates:gameObject)
GameObject
+ override variables and functions as
Product
(Game Object)

Figure 3.2.1 Factory Class Diagram

3.3 Observer Design Pattern for Synchronisation

As synchronisation requires player's variable to be sent over to other players in the network, an observer design pattern can be implemented to broadcast each player status over the network

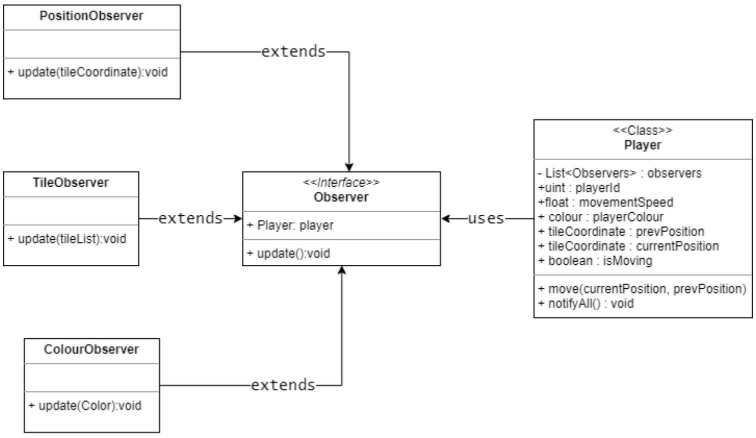


Figure 3.3.1. Observer Class Diagram

3.4 Strategy Design Pattern for Spawning

Spawn class is a base class that holds functions on how a typical spawn should interact with other game objects. Each specific spawn will override the behaviour and this design pattern allow the spawn to be easily define in the spawn manager.

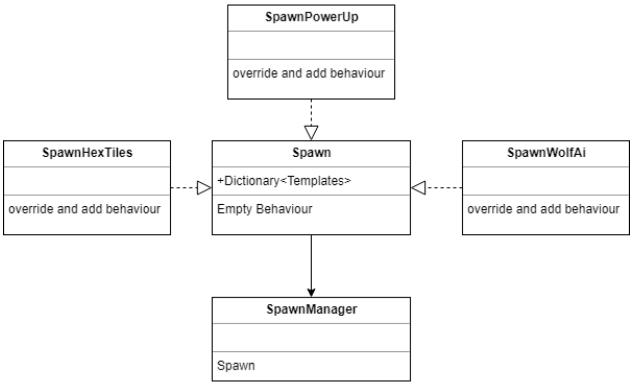


Figure 3.4.1. Strategy Class Diagram

3.5 Singleton Design Pattern for Networking

Singleton design pattern only allows the instantiation of a class to one object. In this case, only an instance of network manager is required to handle the networking aspect of a multiplayer game.

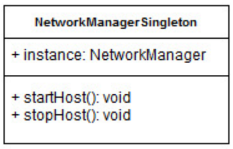


Figure 3.5.1. Singleton Class Diagram

3.6 No Design Pattern for Hex Map Interactions

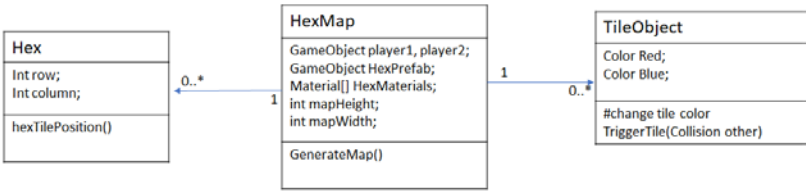


Figure 3.6.1. Hex Map Class Diagram

3.7 No Design Pattern for Player and Wolf Interaction

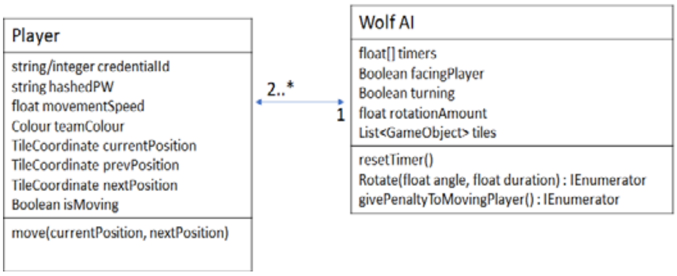

Player
string/integer credentialId
string hashedPW
float movementSpeed
Colour teamColour
TileCoordinate currentPosition
TileCoordinate prevPosition
TileCoordinate nextPosition
Boolean isMoving
move(currentPosition, nextPosition)
2..*
1
Wolf AI
float[] timers
Boolean facingPlayer
Boolean turning
float rotationAmount
List<GameObject> tiles
resetTimer()
Rotate(float angle, float duration) : IEnumerator
givePenaltyToMovingPlayer() : IEnumerator

Figure 3.7.1. Player and Wolf Class Diagram

3.8 Sequence Diagram

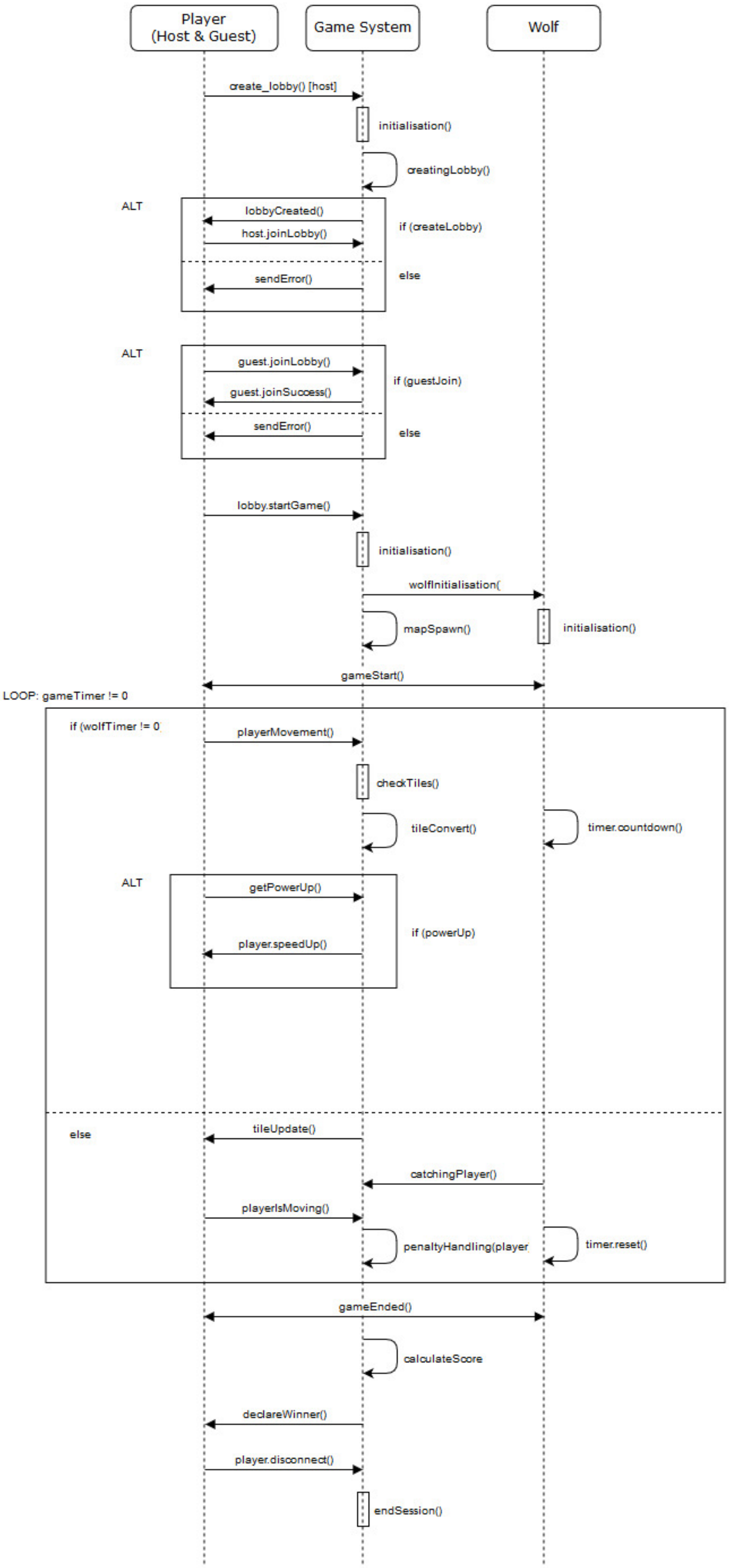


Figure 3.8.1. Sequence Diagram

4. Implementation Challenges

4.1 Engineering Challenges

4.1.1 Game Design

The challenge faced in the game design was that the game ideas proposed was not unique due to heavy influences from the games the team members previously played. Initial ideas that were proposed were rejected as they felt too like games that are already currently on the market. During the team's first meeting with Mr Yoga, he pointed out various aspects that the team were severely lacking in the parts that gave games their entertainment value. These aspects included the lack of compelling game objectives, no unique selling points due to large similarities with existing games and insufficient game elements. The team were also advised to flesh out the game mechanics more clearly as the proposed ideas were too vague - more specific with the details such as damage values of spells and cooldowns was required.

Building upon Mr Yoga's advice, improvements on the ideation of game to various degrees were made. During the private meetings with Mr Yoga, the team was advised that combination mechanics from different games to generate original content. This allowed the team to start brainstorming for ideas from different perspectives - picking out mechanics of games that were entirely different from each other and combining them to design a game. This combination of game mechanics ensures the minimisation of any similarities with games that are in the market. At the same time, this will raise the game's unique selling point, something that had been severely lacking in before. This process of game ideation eventually concluded with the current game design of territorial acquisition mechanics and wolf mechanics from a childhood game.

4.1.2 Game Implementation

The next challenge faced in the game design was the implementation of game functions and mechanics. When drafting the initial game plan, the team looked at various engines available that could be used to support and handle the multiplayer aspects of the game. The initial choice of engine was Google Play Services due to the various features that it provided. Not only could Google Play Services handle the networking part of the game, it also ensured the security integrity of the game with the current security features that it has to offer.

However, despite the features that Google Play Services offered, synchronisation issues occurred after the first iteration of our game prototyping. While testing a multiplayer game session, bugs, such as inconsistent map layout for both client and server, were discovered. Furthermore, there was a limited number of tutorials and information on Google Play Services that was available to resolve these issues. After careful consideration, the team decided to drop Google Play Services in favour of Unity Network Manager as a wider range of tutorial and information source is available for support.

4.1.3 Synchronisation and Networking

Following up from the conversion to Unity Network Manager from Google Play Services, synchronisation issues were resolved through the utilisation of Unity's Higher-level API (HLAPI) through its Network Behaviour. HLAPI allows the flagging of attributes with [SyncVar] and [SyncListUInt] flags, and certain methods with [RpcClient] and [Command]. Specific details on how these flags and functions work will not be gone into in this report; however, similarities can be drawn with how synchronised variables and methods are utilised to solve synchronisation issues in Java. Networking issues are also resolved through utilisation of these solutions, making the use of Google Play Services redundant.

4.1.4 Concurrency

Due to the multiplayer aspect of the game, there is heavy dependence on concurrency for the multiplayer gameplay as both players are simultaneously interacting in real time. Exchanging of information between the server and client must be done concurrently to ensure that both players have access to identical and updated information. Coroutines are utilised in the game's scripts to solve this issue - but with certain limitations. To execute a coroutine, the main thread calls and redirects resources to the coroutine. Any processes executed by a coroutine is done by the main thread with its own resources. In this aspect, coroutines are not thread safe as parallelism is used. Measures must set in place to ensure that executed coroutines do not consume too much resources; otherwise the main thread will be frozen. Despite the limitations, coroutines are still employed as any functions or parameters accessed from Unity Engine API must be called by the main thread. The condition thus largely limits what can be called in a new thread, such as inherited classes from MonoBehaviour and Network Behaviour. Therefore, the limitations of coroutines can be circumvented by carefully setting limits to the number of coroutines executed as well as the sizes of task carried out by any executed coroutine.

4.2 Algorithmic Challenges

One main feature of the game is the scoring based on territorial acquisition – the player who conquers a larger area wins. As such, players must interact with the arena map that they will be playing in, making the arena map an integral part of the game. The team designed the arena map to be split into many individual parts, where each part will be counted towards the score of each player. To facilitate the counting of scores, it was decided to make use of tiles, which players will be able to interact with as well as to better the appeal of game visuals.

The choice of tile shape was done with careful consideration. The shape of the tile should not only provide unbiasedness when players interact with the tile from all possible approaches, but also should appear regular enough to provide a nice pattern for the arena map. Triangle was rejected since there was biases in interaction with the corners; square was rejected due to its plainness and common use in many games. Thus, it was decided to go with the hexagon shape as not only was it relatively round compared to squares and triangle. Furthermore, the hexagon shape gave the game a nicer, more ordered pattern as compared to using circles. Dividing the arena map into columns and rows, factory design pattern was used to generate identical tiles of respective colours in the centre of each individual part.

4.3 Testing Challenges

A critical aspect of games is that physical interactions from players are required to work game mechanics and functions. This holds true for the game as physical inputs from players are required for the game to properly function, such as hosting and join of games, and screen tapping for movement. In Java, software testing of the code can be automated using JUnit and JMock. In Unity however, the language used is different - C# instead of Java. An alternative solution for software testing that was employed was Unity Test. Incorporating NUnit, which is largely like JUnit for testing purposes, the game's code was ran through Play Mode testing to test the integrity of the code.

Physical interactions, however, cannot be automated using Unity Test as physical inputs from a live player is required. To mitigate this problem, physical testing is implemented to test the robustness of the game mechanics and functions. Dividing each game component into individual test scenarios, each test scenario is vigorously tested for a minimum of 8 times before it passes. Different play testers were also used for each iteration of every test scenario to ensure unbiasedness in the test results.

5. Testing

5.1 Testing plan

The game testing was broken down into two phases; one for software testing and another for physical testing. The different phases are mainly due to the need to test the physical interactions the game has with the player.

The following steps of the test plan was carried out for each iteration of the project and the flow chart diagram of the test plan can be found below.

The game is built upon the interactions between the many game objects and, if, one of the game object is missing, the game is broken. Thus, starting with the software phase, the testing of the generation of objects were tested first before moving on to unit testing.

In unit testing, utilising the 'Play Mode' of Unity Test Runner, automating the software test cases was possible and different critical variables were tested. For example, ensure that the player's tile count and the generated tile game object is the same.

Next, function testing was performed. Functions are tested since several functions work concurrently with other function to perform a single event. This intricate working, if not tested, may lead to unforeseen complications and cause the game to be broken. An example would be testing for the penalty event. The penalty event should only be executed when the player is moving while the wolf is facing the player, and this requires 3 different functions to be executed correctly. More example of the different kinds of test cases can be seen in the next section.

Afterwards, physical testing is implemented, where the main objectives is to test the robustness of the game, locating potential bugs and gathering feedbacks.

For robustness, playtests, which are detailed in the next section, were conducted to determine how well the game mechanics worked together when implemented. This involves mainly components that have player interactions such as the matchmaking system, player movement, wolf and penalty mechanics. It is to note that the playtest passes overall only after passing 8 times.

The game then proceeds on to allow new test users to play the game to gather feedback and improve the current iteration. Throughout the process of allowing new test users, it was possible to locate several bugs in different parts of the game such as synchronisation bugs. Fixes were implemented, and the test plan is repeated until the final iterations. The types of bugs and fixes can be found in the below section.

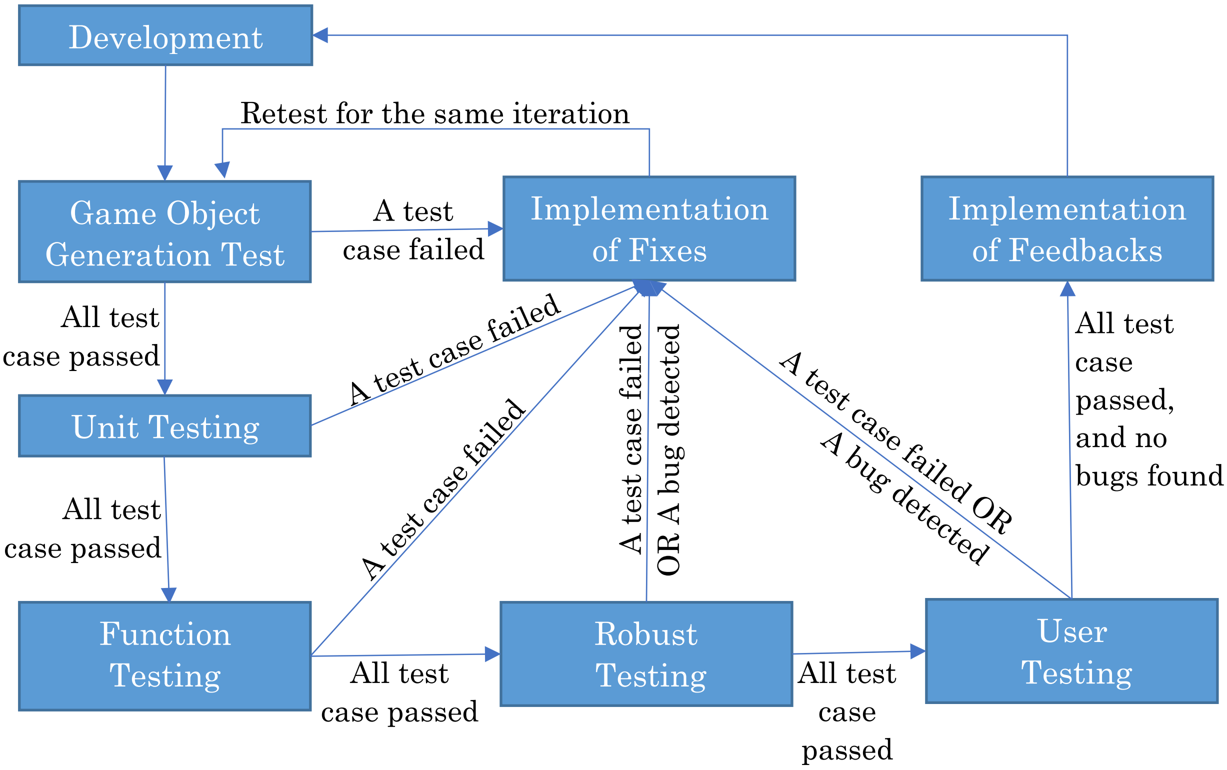


Figure 5.1.1. Test Plan Flowchart

5.2 Test cases

The software tests are conducted via NUnit provided by Unity test runner. The hardware tests for each feature will require it to pass 8 times before it counts as an overall pass. Below is the checklist for each testcase.

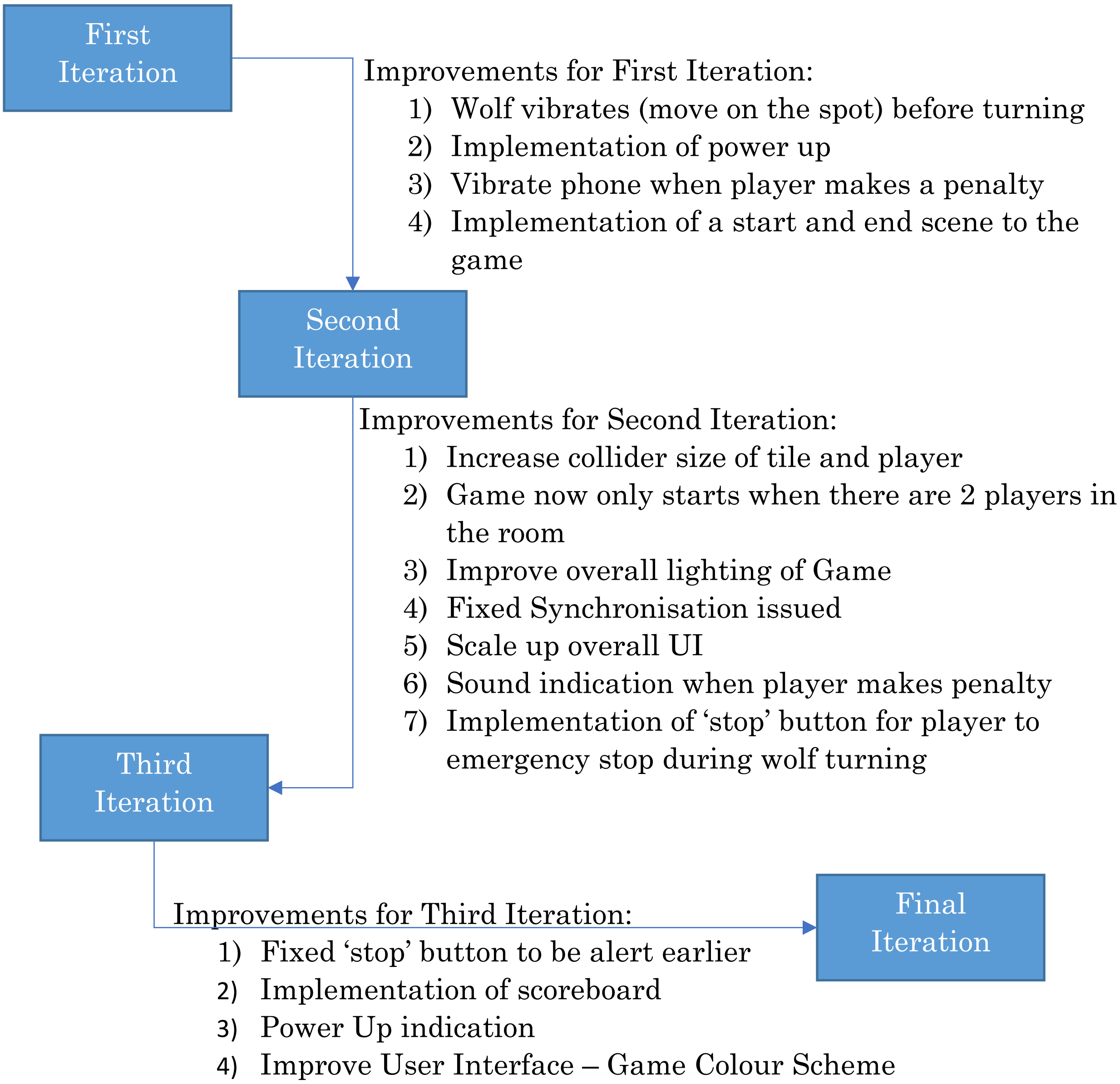
Type	Functionality	Results
Game Object Generation		
Software	Wolf Game Object	Passed
	Hex Map Game Object	Passed
	Blue Tile Game Object	Passed
	Red Tile Game Object	Passed
	Player Game Object	Passed
Gameplay – Tile Logic		
Software	Randomised tile generation	Passed
	Correct tile counts for both players when a penalty is assigned	Passed
	Correct tile counts for both players when a player moves over tile	Passed
Hardware	Tile change when player is closed to the centre of tile	Passed
	Visible Penalty Indication – Gradual Fade in Opponent Colour	Passed
	Tile map generated is the same for both player	Passed
	Changed in Tile Colour when penalty occurs	Passed
Gameplay – Wolf Logic		
Software	Correct wolf timing	Passed
Hardware	Wolf can be seen moving before turning	Passed
	Wolf turn at the same time for both player	Passed
Gameplay – Player		
Software	Increase in player speed with powerups	Passed
Hardware	Player moved to tapped position	Passed
	Increased in player movement speed up (before and after powerup pickup)	Passed
	Vibrations and sound are observed when penalty is made	Passed
Networking		
Hardware	Synchronisation of all feature	Passed
	Player can create room	Passed
	Player can join room	Passed
	Both player will be at a waiting room before game start	Passed
	Both players are at the same room during gameplay	Passed

5.3 Game Review

Each test phase consists a few testers who will be asked the following questions after the gameplay.

1) What are some difficulties in playing the game?
2) Is the gameplay intuitive?
3) One feature that you like and/or dislike?
4) General Comments

After each test phase, the keywords for each comment were noted and based on the severity of the issue, improvements were made for the next iteration as seen below. The game reviews made can be seen in the Appendix 2-5.



Improvements for First Iteration:

1) Wolf vibrates (move on the spot) before turning
2) Implementation of power up
3) Vibrate phone when player makes a penalty
4) Implementation of a start and end scene to the game

Improvements for Second Iteration:

1) Increase collider size of tile and player
2) Game now only starts when there are 2 players in the room
3) Improve overall lighting of Game
4) Fixed Synchronisation issued
5) Scale up overall UI
6) Sound indication when player makes penalty
7) Implementation of 'stop' button for player to emergency stop during wolf turning

Improvements for Third Iteration:

1) Fixed 'stop' button to be alert earlier
2) Implementation of scoreboard
3) Power Up indication
4) Improve User Interface – Game Colour Scheme

Figure 5.3.1. Iteration Flow

5.4 Bugs and Fixes

Below are the bugs and fixes that occurred through each iteration.

S/N	Summary	Fixes	Iteration occurred	Status
1	Tiles are not synced between the players' devices	Switched to Unity Network Manager	1	Resolved
2	Player movements are not synced across different devices	Switched to Unity Network Manager	1	Resolved
3	Penalty is not handled when players move during Wolf facing	Switched to Unity Network Manager	2	Resolved
4	UI not scaled properly during gameplay	Dynamically scaled the UI using canvas scaler	2	Resolved
5	Lighting of game is too dark during gameplay	Increased lighting	2	Resolved
6	Power ups always spawn in fixed position	Randomised position to spawn power ups	2	Resolved
7	Host player is unable to return to main menu when waiting for players to join game	Added back button in the lobby scene	3	Resolved
8	"Phantom" room when host quits the game before any player joins the room		3	Unresolved
9	Powerups spawn below/in tile	Adjust spawn position of power up to centre of tile with offsets	3	Resolved

10	Starting of game is not synced when players join a hosted game	Rebuild APK	3	Resolved
11	Power up effect (SPEED++) is permanently visible on opponent's model	Use RRPC command	3	Resolved
12	Stop button overlaps with the Hex Map on smaller phone screens		4	Unresolved
13	End Game Mask sometimes do not appear	Separate animation from the end mask (have 2 layer of animation)	4	Unresolved

6. Conclusion

6.1 Software Development Process

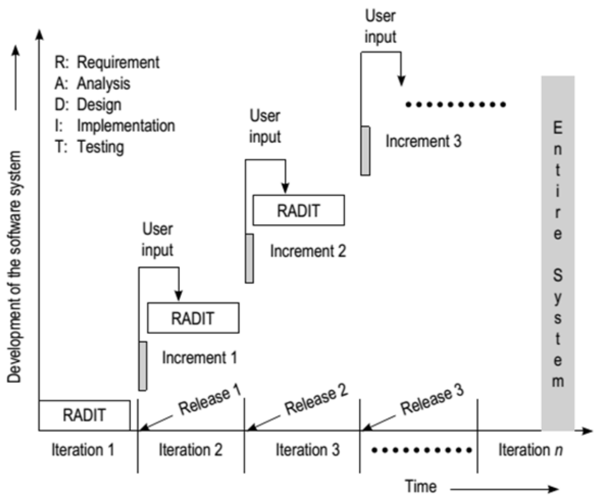


Figure 6.1.1. RADIT flow[1]

The software development process that was followed is iterative and incremental development. The reason the group chose to follow this model was to allow the new features such as the power up to be added onto the basic idea of the game. It was also due to the feedback from the design challenges that the team realise it was best to choose the method that allow the addition of game features. However, this was not that easy as there is no limit to the kind of game features that the team can choose to implement. The team had to filter out fanciful ideas such as bouncing the player and decide on practical ideas for the game.

[1] Software Engineering: Concepts and Applications by Subhajit Datta (Oxford University Press, 2010)

6.2 Lessons Learnt

This project allows the team to learn the application of design patterns taught in class and the usefulness of them. This project has been a positive experience for the team and would benefit the team in the future. However, the project definitely has room for improvement and the team feels that there are some game features that should have been added in (as stated in the below section). In addition, the teams feel that existing game features could be better integrated and with given time, the project could develop into an amazing and exciting game for all to play.

6.3 Future Work

As mentioned in the above section, the team feels, with given time, the following features can be implemented.

(a) Set predefined/blocked areas
(b) Implementation of different types of power up
(c) Player can choose the length of gameplay
(d) Add preference to volume and vibration
(e) Implementation of pause button

6.4 Source Code

The project source code can be found in the following GitHub link:

https://github.com/mokjunneng/WhatTilesMr.Wolf

The video can be also found in the GitHub repository.

6.5 Acknowledgement

The team would like to thank the game testers mentioned for taking time out to help test the project. In addition, we would like to thank Mr Yoga and Mr Sudipta for their advice in the development and testing phase. Without them, it would be impossible for the project to run so smoothly.

7. Appendix

Appendix 1. Previous Project Reports

Meeting Report No.	Link to report
1	https://docs.google.com/document/d/1kOjMTyLNXXUo2Qvz6uuT0MzMxlTBEA-qRT67JLy689E/edit?usp=sharing
2	https://docs.google.com/document/d/1FQ6vdhgD4S1iLbRvmnIN_2ychGbiyz-t33UdnF9_vSQ/edit?usp=sharing
3	https://docs.google.com/document/d/1yaPSmZpmtWQORff_o5A4A8aZfO07KnEl4yUskcgcW8k/edit?usp=sharing

Appendix 2. Game Review of the First Iteration

Tester Name	Game Review after each test phase
First Iteration – After Project Meeting 3 (PM3)	
Song Shan	Q1) Maybe can make **wolf vibrate** cause a bit hard to see turning Q2) No, I need to anyhow tap on the phone before playing Q3) Hex style Q4) Quite **innovative game**
Rayson	Q1) Some **power up** so that the game will be slightly more interesting Q2) Yes, easy to pick up Q3) Simple player movement Q4) **Interesting game**, maybe can add incentives to make it more interesting
Wen Qi	Q1) The model a bit ugly Q2) Yes, simple gameplay and not too complex Q3) Penalty system but **hard to tell about penalty** Q4) It was quite **easy to play** because I can slide at the side of the tile to gain it
Eunice	Q1) **Alert** me when I **lose a tile** Q2) No, **no start scene before game** Q3) Cute Wolf XD, maybe can **add start scene** Q4) **Fun** game, maybe can change the model
Angelia	Q1) **Alert** me when I **lose a tile** Q2) Yes, quite fun to play and intuitive Q3) Cute Wolf, maybe can **add start scene** Q4) **Concept like childhood game**, makes it easier to pick up
Extra Comments made by Mr Yoga after PM3: 1) Add an alert system for the wolf and player to know about penalty. **2)** Improve the User Interface system for the user; add a start scene / end game scene	

Appendix 3. Game Review of the Second Iteration

Tester Name	Game Review after each test phase
Second Iteration – After Project Meeting 4 (PM4)	
Rachel	Q1) **Hard to tell** that I need to tap the centre to **trigger the tile** and **when penalty is given** Q2) No, I need to have a few attempts to know I need to rigger at the centre Q3) Dark Theme but I can **play lonely** Q4) **Spread the power up**
Reuben	Q1) Wolf AI **lighting too dark** Q2) Yes, easy to pick up and fun to play Q3) Nice models but **timer a bit short** Q4) **Player tile and cube don't match** but overall fun game
Cyrus	Q1) **UI Scene too small** Q2) Yes, simple gameplay Q3) Short Gameplay, nice maybe **increase overall UI size** Q4) **Add a ready button (?)** such that both player will spawn at the same time
Yue Ran	Q1) Game suddenly stop/ **not paying attention to timer** Q2) Yes, quite fun and intuitive except for the **sudden end game** Q3) **Game concept** but the **player cube is not consistent** red cube but blue tiles Q4) **Concept like childhood game**
Jeffery	Q1) **Hard to prevent penalty** when the wolf turn. Cannot **sudden stop** Q2) No, **cannot suddenly stop my player to avoid penalty** Q3) **Power up** makes game **interesting** but need to **spawn evenly** Q4) Add **music** in game abit boring
Extra Comments made by Mr Yoga after PM4: 1) Either increase the collider size or add a dot on the tile as an indication the collision is there 2) Increase the scale of the UI as it is hard to see 3) Add a further indication to the user after tile lost 4) Add a 'stop' button during wolf turning for player to emergency stop	

Appendix 4. Game Review of the Third Iteration

Tester Name	Game Review after each test phase
Third Iteration – Before Final Presentation	
Max	Q1) No **clear instructions** in playing the game Q2) Yes, it is Q3) The game idea itself is very **original**, but I dislike the **animation** Q4) An overall interesting game to play
Kah Hoe	Q1) **Stop** button should appear during **wolf vibration** Q2) Despite having **no instructions**, it was intuitive Q3) I like the overall **theme** of the game; the **font** gave a nice **retro feel** to the game. However, perhaps can **include** a **scoreboard** to show the **progress** of the game Q4) Fun game to play
Ying Jie	Q1) **Instruction** page would allow **the game flow** to be better; No need to tap around randomly to realise how to play the game Q2) No it wasn't, as there were **no instructions** Q3) I have fat fingers, so this makes **pressing the tiles hard**. But I like **the game concept** and it is **very original** Q4) Other than the lack of instructions, it was a nice game
Swee Chong	Q1) Game started **abruptly** for the one **joining the game;** perhaps can add a **lobby area/loading area** that waits for both player to enter Q2) Yes, it is intuitive Q3) Feels like **the childhood** game I use to play. Q4) Interesting game idea, perhaps can work on the **models**
Beng Huan	Q1) **Room creation** have some issues such as **creating with only spaces** Q2) No, it wasn't Q3) The **model of the wolf and player** can be **improved** but I like the **game idea.** Q4) **Increase the time interval for the wolf turning as the time passes**
Extra Comments made by Mr Yoga: 1) Player indicator (For example, arrow at the start to show starting position, aura to indicate players) 2) Stop button appear during vibration 3) Tile colours and player colours changed as red is usually to indicate danger 4) Eviler wolf model and Cuter player model 5) Tile counter in game and More tile lost indication 6) Power up indication (For example, text indication or player aura changes) 7) Shift map downwards a bit and Room name scaling	