**REQUIREMENTS**

**GROUP D1, CMPT370**

Nico Dimaano, ned948

Niklaas Neijmeijer, nkn565

Kyle Seidenthal, kts135

Brendon Sterma, bws948

Jiawei Zang, jiz457

**INTRODUCTION**

This document outlines the requirements for creating a system for the “Robot Arena” game. The “Robot Arena” game is a simple turn-based game, played on a board. There can be 2, 3 or 6 players at a time. Each player begins with three robots, each with different statistics. The players take turns moving their robots, starting with the robot with the highest movement. On their turn, a player may move, shoot or do nothing. The last player with a robot on the board wins the game. The system we are creating will implement this game and its rules on a computer system that allows the game to be played by multiple players at once. This will make the game less tedious to keep track of, and allow a comfortable gameplay experience than playing the game on a traditional tabletop board. One of the main features of the game is a visibility system, where the tank can only see as far as its range, which brings out new strategy and difficulty to the game. Another feature is that you have the ability to face AI opponents. In addition, the game can be played over a network to allow the player to conveniently play from his or her own machine. The rest of this document will outline what such a system will require to be successful.

GIT TAG GOES HERE WHEN COMPLETE --------------------------------------------------------

**SYSTEM DIAGRAM**

Below is a high level diagram of the system:

+---------------------------------------------------------------------------------------------------------+

TODO

+---------------------------------------------------------------------------------------------------------+

**ACTIONS AND THEIR SCENARIOS**

Below is an overview of actions and scenarios compatible with the system.

One of the actors in this system is the user this represents a person who has started the program and is interacting with the system. The user can take the actions: Navigate Menu, Create Game, and Join Game.

**User: Create Game Scenario**

When the user has chosen to create a game from the menu, they are presented with option regarding the number of human and non-human players are to be part of the match. A new game is initiated and the user becomes a player in the system. Should the player decide to cancel, they are sent back to the main menu.

**Preconditions: -**The player has successfully started the program and can navigate the menu screen.

**Post-conditions:** -The user is a player

**-**A new instance of the game is created

**Error-conditions:** -If the player does not have a network connection and chooses two or more human players, the game cannot be created and is not joinable by other human players.

**FIG x.x:**  *The above figure shows the sequence of events taken by a user who has chosen to create a new game. Note that at any time, should the player decide to cancel this action, they are sent back to the main meu screen.*

**User: Join Game**

The user may also choose to join a game as a player. In this scenario, the user choses the “Join Game” option from the menu screen. The user is then prompted to enter a form of match identification for the match they would like to join. If the user has a valid network connection, they are prompted to choose to be a player or an observer. connected to the match as a player.

**Preconditions:** -A valid game must exist

-There must be an empty player slot in the game

- The user and game must both have valid network connections

**Post-conditions:** -The user becomes a player

-The user has joined a match

**Error-conditions:** -The user has no network connection

**FIG x.x:** *The above figure shows the sequence of events when the user chooses to join a match as a player.*

**Secondary scenario: User – Join as observer**

Should the player choose to join the game as an observer instead of as a player (as in step 5 of the diagram above) they are connected to the game as an observer. This allows them only to watch a game in progress.

When a user has successfully joined a match, they become a player. The player is part of a match and as such can take turns. On their turn, they can take the actions: Forfeit, Inspect Tile, Move, Attack, and End Turn.

**Player: Forfeit**

When the user is part of a match, they become a player. Should the player desire to quit the game during an active match, they have the option to forfeit the game. They can do so by selecting the forfeit button, at which time they will be prompted with a confirmation dialogue. They may cancel the action and return to the game, or they can confirm the action and become an observer of the match. They can then leave the game at any time.

**Preconditions: -**The player is part of an active match

**Post-Conditions:** -The player becomes an observer and is no longer part of the match.

-The player’s robots are removed from the game

**FIG x.x:** *The above figure shows the sequence of events taken by a player who considers forfeiting the game. When the player forfeits, they become an observer. They may also cancel their forfeit.*

**Player: Inspect Tile**

On their turn, a player may inspect a tile. A tile can only be inspected if it is in range of one of the player’s robots. If the tile has any tanks on it, a window with a list of all robots on the tile and their stats will be shown to the player. If the tile is empty, the window will not be shown.

**Preconditions:** -The tile is within range of one of the player’s robots

**FIG x.x:** *The above figure shows the sequence of events the player takes when inspecting a tile.*

**Player: Move**

Along with inspecting a tile, on their turn a player may move one of their robots to a tile within range of the robot. They do so by selecting the move option and selecting a tile in range and confirming their choice via a confirm button. The robot will then move to the selected tile, and the line of sight for the player will be updated. Note that a player can repeat this action as long as the robot has not moved a number of tiles equal to its movement range.

**Preconditions:** -It is the player’s turn

-The robot the player is moving has not moved its movement range

**Post-conditions:**  -The robot has been moved to the tile selected

-The number of tiles traveled this turn is updated

to match the number of tiles the robot has moved

**Error-conditions:** -The player chooses a tile out of range of the robot

-The player has moved the movement amount for the robot

**FIG x.x:** *The above figure shows the sequence of events for a player who has chosen to move a robot. This action results in a robot being moved to the selected tile, and the player’s field of view being updated. Note that if the robot has moved its movement range, the robot can no longer be moved.*

**Player: Attack**

Along with moving, a player can attack on their turn. The player can attack a tile if it is in the shooting range of one of their robots. The selected tile then takes the amount of damage the player’s robot can give, and subtracts it from the health of all robots on the tile. Note that a player can move and shoot in the same turn, and can move after shooting, provided the robot’s movement range has not been exceeded. Also, a player can damage their own robot.

**Preconditions:** -It is the player’s turn

-The enemy robot is within shooting range of the player’s robot

**Post-conditions:** -The enemy robot takes the amount of damage the player’s robot can give

**FIG x.x:** *The above figure shows the sequence of events taken by a player*

*who has chosen to attack. Note that a player can damage their own robot.*

**Player: End Turn**

Finally, a player can end their turn. This can be done at any time during a player’s turn. The player chooses the “End Turn” button and will be prompted with a confirmation prompt. When the player confirms, their turn is over, and the next player takes their turn.

**Preconditions:** -It is the player’s turn

**Post-conditions:** -The player’s turn is over, and the next player takes their turn

**FIG x.x:** *The above figure shows the sequence of events taken by a player choosing to end their turn. This can be done at any time during the player’s turn. The next player gets their turn after this action.*

The Robot librarian is another actor in the system. It is used to manage robot programs. The Robot Librarian can: Enumerate, Download robot programs, and Update Match Statistics after a match.

**Robot Librarian: Enumerate**

The Robot Librarian can query for robot programs by different fields such as: Team, Name, Wins, Matches Played, Win/Loss Ratio. The Robot Librarian can also display statistics.

**Post-conditions:** -The query returns the list of robots in a sorted order

**Error conditions:** -The robot programs cannot be accessed

**FIG x.x:** *The above sequence diagram illustrates the sequence of events taken by the Robot Librarian when asked to search for robot programs.*

**Robot Librarian: Download**

The Robot Librarian can download robot programs so they can be run with the system.

**Preconditions:**  -A robot program is selected

**Post-conditions:** -A robot program is downloaded into the system

**Error conditions:** -The robot programs are unavailable

**FIG x.x:** *The above sequence diagram shows the sequence of events taken by the Robot Librarian when it is required to download a robot program. The program can then be used by the system.*

**Robot Librarian: Update Stats**

The Robot Librarian is also able to update the stats for a robot program. After a match has finished, the Robot Librarian uploads the new stats for the selected program to moodle.cs.usask.ca.

**Preconditions:** -A match has finished with the selected robot program

-There is a valid network connection

**Post-conditions:** -The stats of the robot program are uploaded to moodle.cs.usask.ca

**Error Conditions:** -Data is unable to be sent to moodle.cs.usask.ca

**FIG x.x:** *The above sequence diagram shows the sequence of events for a Robot Librarian uploading new stats to moodle.cs.usask.ca.*

The server is the actor that handles communications between multiple machines for network gameplay. It can: Save the Game State and Receive and Respond to Requests for updates.

**Server: Save Game State**

The server periodically saves the state of the game. This allows it to send updates to other players on request.

**Preconditions:** -There is an active game

**Post-conditions:** -The state of the game has been saved to the server.

**Error-conditions:** -The server loses connection to the game

-The game is corrupted upon saving

**Server: Receive and Respond to Update Requests**

The Server is able to receive and respond to update requests. A player’s computer will periodically request updates of the game state from the server. The server can respond to these requests with an updated instance of the current match.

**Preconditions:** -The server is connected to a game

**Post-conditions:** -The server has sent an updated instance of the game to the player

**Error-conditions:** -The game data is corrupted during transmission

-A player disconnects from the match

**FIG x.x:** *The above sequence diagram shows the steps taken by the server when it has received an update request from a player’s computer.*

The observer is the last actor in the system. It is similar to a player, but cannot directly influence gameplay. The observer can see all tiles on the board and the robots on them. The observer has the ability to: Inspect Tiles, and Leave.

**Observer: Inspect Tiles**

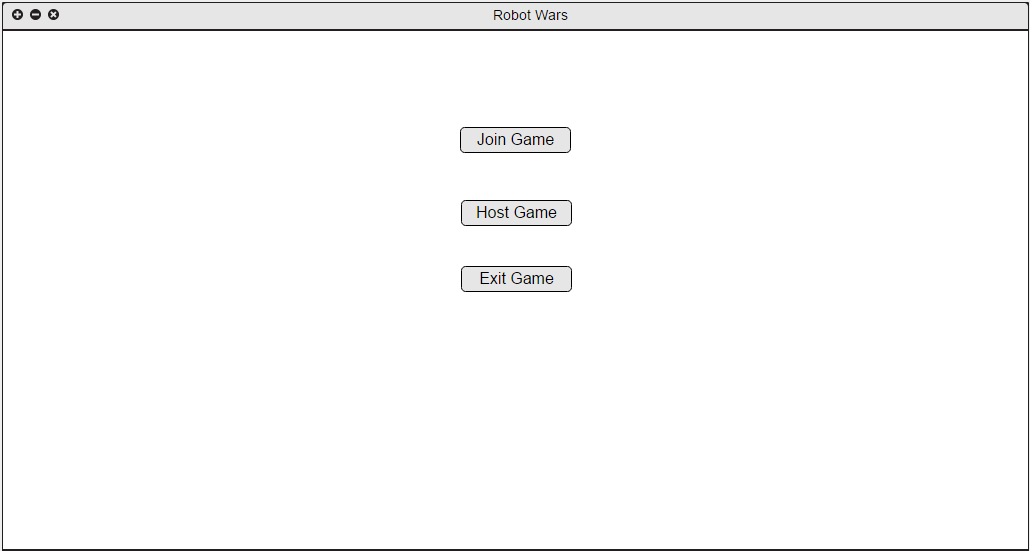
**// TODO**

**Observer: Leave**

**// TODO**

**Mock Ups**

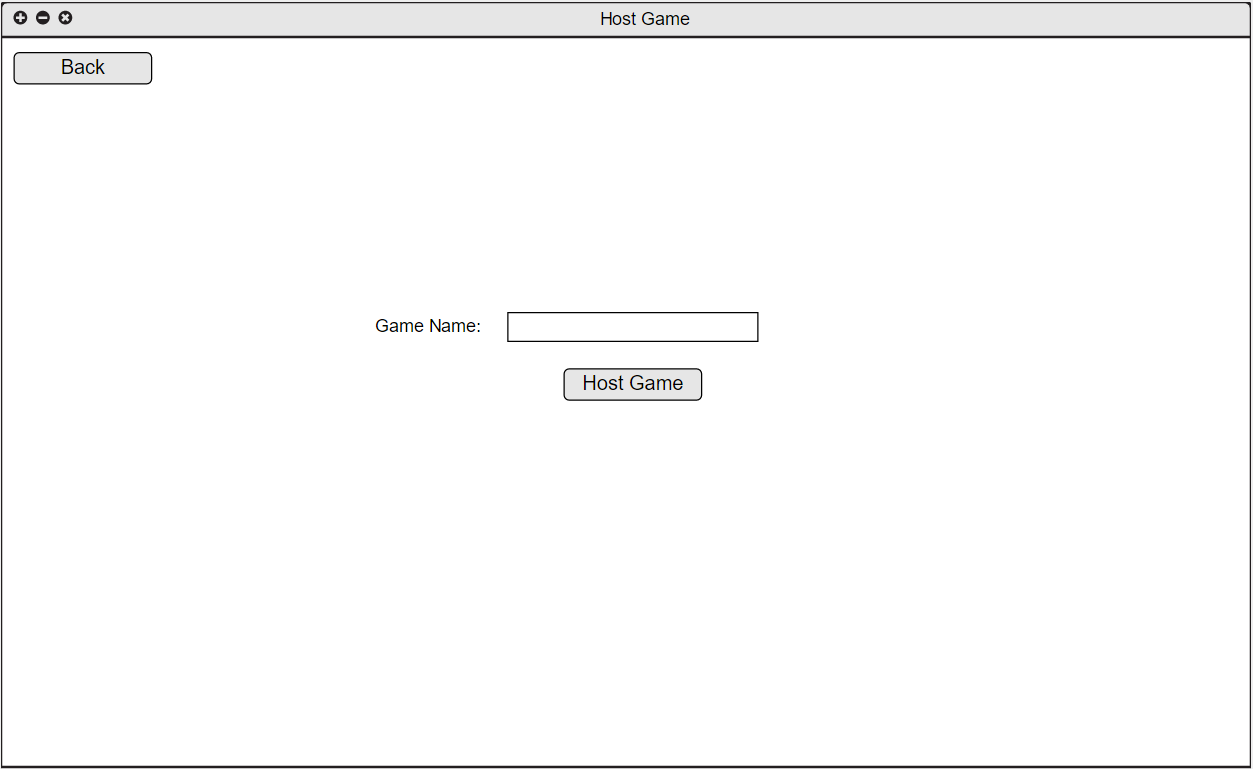
**Mock Up: Start Screen**

****

Description:

This is the Start Screen the user first sees when accessing our Robot Game. He will see three buttons right in the middle which are “Join Game”, “Host Game”, and “Exit Game”.

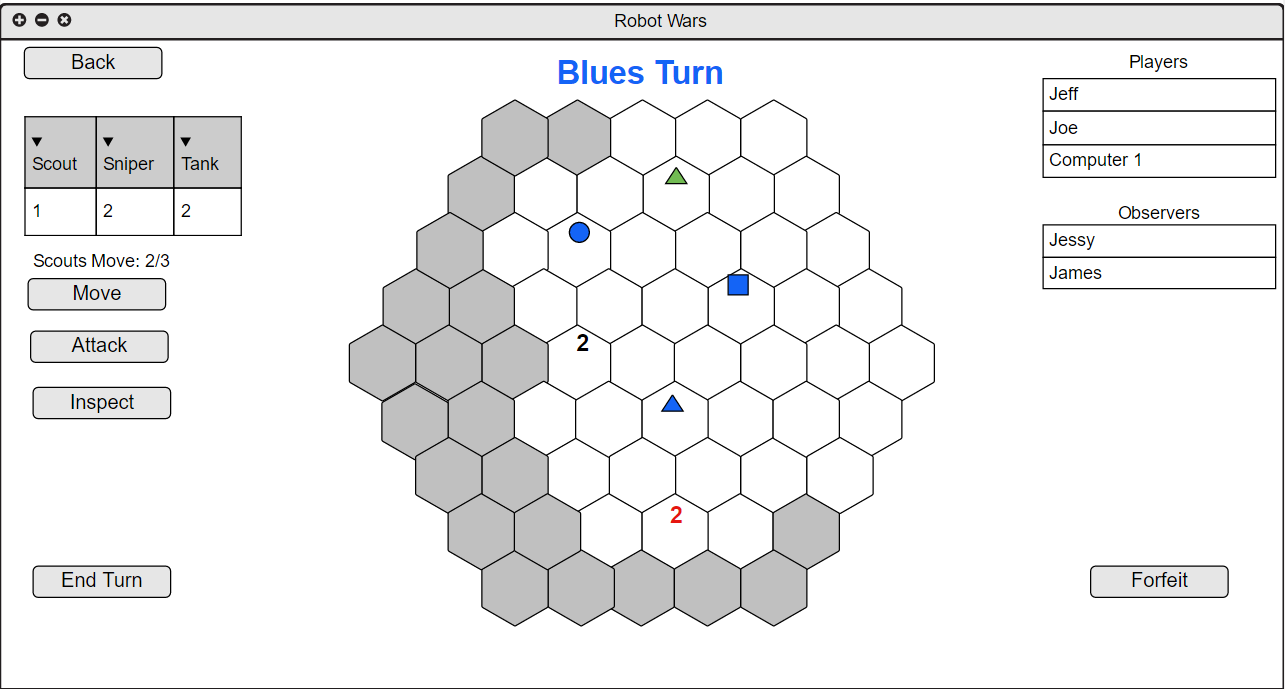
**Mock up: Game Screen Host**

****

Description:

This is what the user sees when “Host Game” is selected. He/She will be required to input a Game Name in the text box which and the button “Host Game” will be selected once finished.

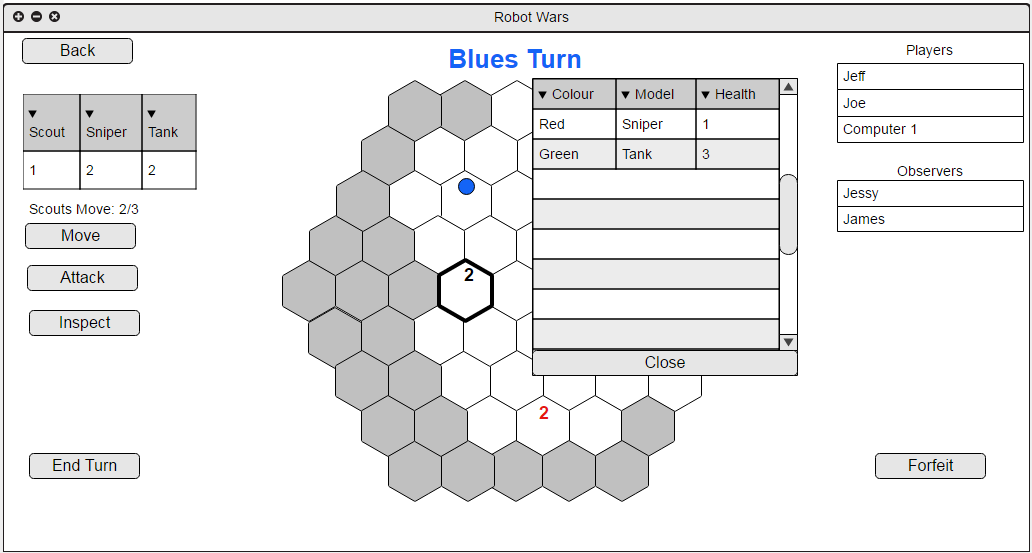
**Mock up: Game**

****

Description:

This is the current board as seen by the blue player. White hexes are board spaces that are within range of the blue player and grey spaces are out of vision. UI is show in both sides of the screen with player moves and statuses on the left side and current players on the right side. Blue’s robots will always be shown as singular pieces on their position of the board as for enemies will be shown as individual pieces if the are solo on a hex and then shown as a number with their corresponding color if there are more than one. In case of more that one enemy player on a square it will be shown as a black Number.

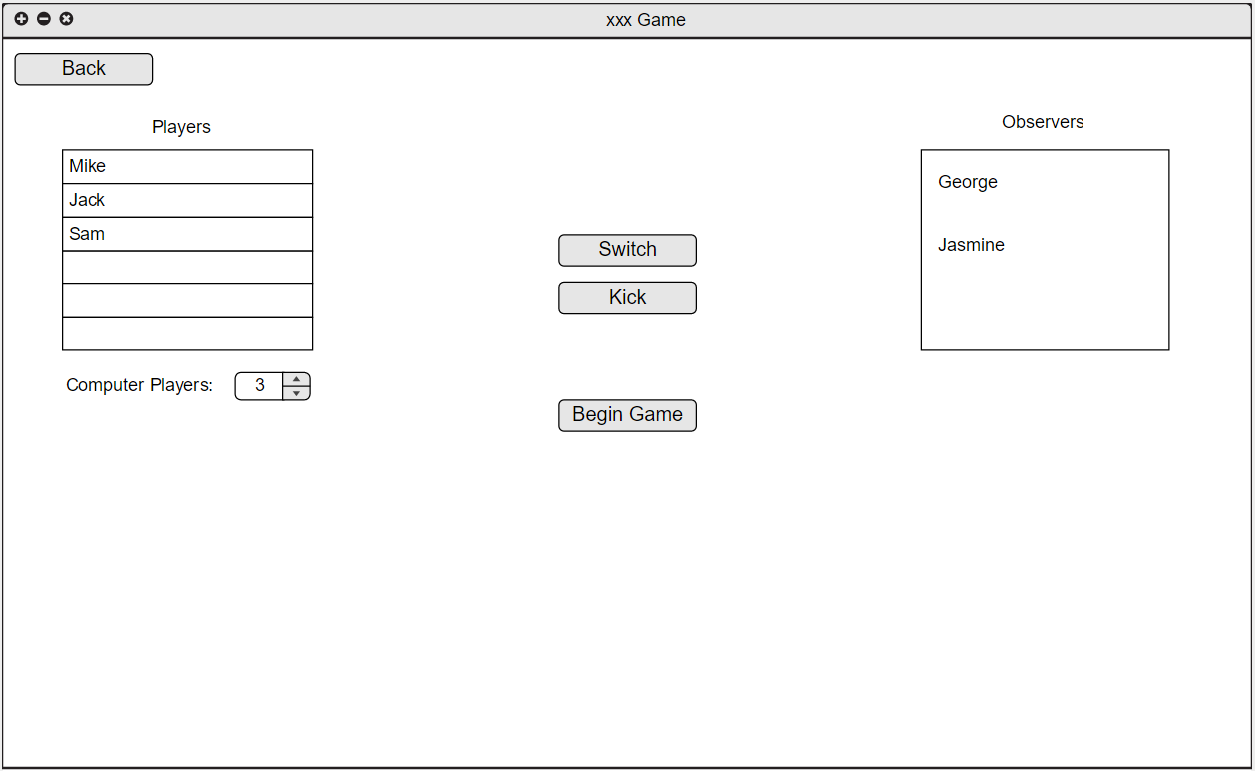
**Mock Up: Inspect Tile**

****

Description:

// TODO

**Mock Up: Lobby Screen**

****

Description:

//TODO

**PLATFORM**

The available platforms for the Robot game will be on WINDOWS and Linux machines with the benefit of facing each other cross Platform. The Game will be allowed to run in tuxworld as well. We will be programming in Java and using Eclipse for editing our code. Git will be our version control software.

**SUMMARY**