

Deployment and Maintenance Document
CMPT 370

Group C4

Jack Huang

Brandon Jamieson

Ixabat Lahiji

Daniel Morris

Kevin Noonan

- the deployment documentation for your system: tutorial/user manual/etc. that meets the goal of informing an end-user about how to use your system---include details of limitations and restrictions
 - e.g. mailboxes don't work
 - e.g. load-time code is not checked for validity
 - not all FORTH words are supported
 - only 2 teams are allowed
- the programmer maintenance documentation for your system
 - an as-build architecture
 - details of tricky/intricate/important bits of your system
 - external libraries you rely on
 - how to compile and run your system
 - what's the main class?
 - with the purpose of helping the next programmer in their task of maintaining or extending your systems
- the standard delta document
 - what requirements (e.g. UI, networking, etc) were not met
 - what designs didn't hold up to construction
 - what bugs remain

User Manual

Brief Description of Game:

[INSERT PICTURE OF

The game itself is played on a hexagonal board, with all three of each team's robots starting in opposite corners. Robots have varying stats like movement points, damage, and range. Each round, players will move and shoot with their fastest available tank, until all tanks have spent their movement points. This concludes a turn, meaning movement points are replenished and the cycle continues. Once there is only one team left standing, the remaining team is declared the winner and the game is over.

[INSERT SCREENSHOT OF MAIN MENU]

Upon entry of the program, the user will be prompted to either play the game, selection the options, or exit the game. The options menu allows the user to configure robot teams by importing JSON files, which will be discussed in further detail later. The play button will take the user to a play menu where they will choose board options.

[INSERT SCREENSHOT OF PLAY MENU]

The board size represents hexagons per side, and can be either five or seven. The total number of players allowed depends on the board size: two to three for a size of five, or three to six for a size of seven. Once the player has selected a board size, the specific teams may be configured. Each team will be represented by one of the following colors: red, yellow, orange, green, blue, or purple. The user will designate colors for each team from the dropdown box (there may be duplicate colors). Users will also select three scripts for each AI team, corresponding to a scripted-behavior for each robot. These may be imported from a remote server through the options menu.

Once the user has selected their preferences, they can press the play button to start.

[INSERT SCREENSHOT OF IN GAME MENU FULLY REVEALED]

At the beginning of the game, the user will be presented with each team's robots will be positioned in opposite corners of other teams. All robots will be at full health and movement points, and player one's scout will be the first active robot. If the team is an AI team, it's robot will perform its behavior script and will end its turn once complete. If the team is a human team, the user will choose the robot's course of action.

[INSERT PICTURE OF ROBOT TEAM DISPLAY PANEL]

The team display panel to the left of the board in the GUI provides information on the current player's robots. For each robot, there will be an image representing their respective game piece as well as their stats. Each robot's attack, range, movement points, and health are displayed beside their image. This will be updated once the current player has ended their play, either hiding itself when an AI is next, or displaying the robot team for the next human.

[INSERT PICTURE OF HEX INDICATOR PANEL]

The hex indicator panel is on the right side of the board in the GUI, and provides information on the currently targeted. Hexes become targeted when the current player enters shooting mode and aims at an occupied hex. Each occupant is shown, along with their robot image in their team's color. This allows you to gauge your opponent whenever you are in range, as users will be able to see enemy health and movement points.

[INSERT PICTURE OF BUTTONS AT THE BOTTOM]

The buttons underneath the hexagon board are for human player's to manipulate their current robot. The "*" button will toggle between moving and shooting mode, changing the action button to represent the current mode. The action button is in the middle, and will either say "Shoot" or "Move" depending on what mode the user is in currently. If shooting, the action button will shoot the current hex, applying the current robot's damage to ALL robots in the target hex (including allied robots). If moving, the action button will move the current robot to the green, highlighted hexagon. Beside the action button, there are left and right buttons for either turning left and right when moving, or for cycling through target hexes when shooting. The "End Play" button will end the current player's play, and will be done whenever a player has finished moving and/or shooting. The "Forfeit" button will forfeit the game for the current player, while the "Exit" button will end the game completely, for all players.

[INSERT EXAMPLE OF FOG OF WAR]

The board itself is initially grayed out for the most part. This is the game's "Fog of War", which hides hexagons that are not within range of any of the current player's robots. This means players will have to explore the map to find enemy robots before they can shoot them. Players can also use this to their advantage in various ways, such as creeping around the sides of the board or hiding in one place.

[INSERT PICTURE OF THE FIRST PLAYER’S VIEW BEFORE ANY ROBOTS HAVE MOVED]

Once the game begins, the user (assuming player one is a human) will take control of the quickest robot on the team: initially the scout. The scout has three movement points initially, so the user may move three times before they can no longer move. Robots may shoot once per turn, although since there are no opponents in sight right off the bat, the only possible target would be the allied team. Once the player has made their moves, they can press “End Play” to move onto the next player’s turn.

[INSERT SEQUENCE OF IMAGES DEPICTING THE SCOUT MOVING THREE HEXES THEN ENDING THEIR TURN]

If the next player is an AI, they will make their move then end their play automatically. If the next player is a human, a panel covering the board will pop-up, allowing the first player to get up and let the second player have their turn without revealing their pieces in the transition.

[INSERT IMAGE OF NEXT PLAYER SCREEN]

Once the players have switched, the new current player will assume control of their scout and make their play. This process continues until all scouts have made their plays, then repeats with sniper and tank robots. If a robot has been shot and no longer has any health, that robot is destroyed and will no longer be playable or in the game at all. Shooting deals the firing robot’s damage to all robots within the target hex.

[INSERT IMAGE SEQUENCE OF ROBOT TARGETING A HEX AND DESTROYING ROBOTS IN IT]

Gameplay continues with each player moving and shooting with their robots until there is only one team left standing. This team will be declared the winner, and will be presented with a screen notifying the respective player. Once this is done, the user will be brought to the main menu where they may decide what’s next from there.

[INSERT IMAGE OF OPTIONS MENU]

Users may also import robot scripts from a remote server via the options menu for use in-game. The default robot server to connect to has the host name “gpu0.usask.ca”, and is stationed at the U of S. If the server is down, or not available from the current location, the options screen will simply be blank. While it may be possible for others to implement their own

server for this purpose, there is currently no functionality for switching hosts without modifying the code itself. Upon entry, the options menu will present the user with a list of importable robots, as well as various stats of theirs. Users may import their selected robots by pressing the “Import” button on them. This will save the robot’s script to the program’s local directory which can be found in the folder labelled “Model”. These scripts may be accessed through the play menu and assigned to AI player’s robots when the game is created.

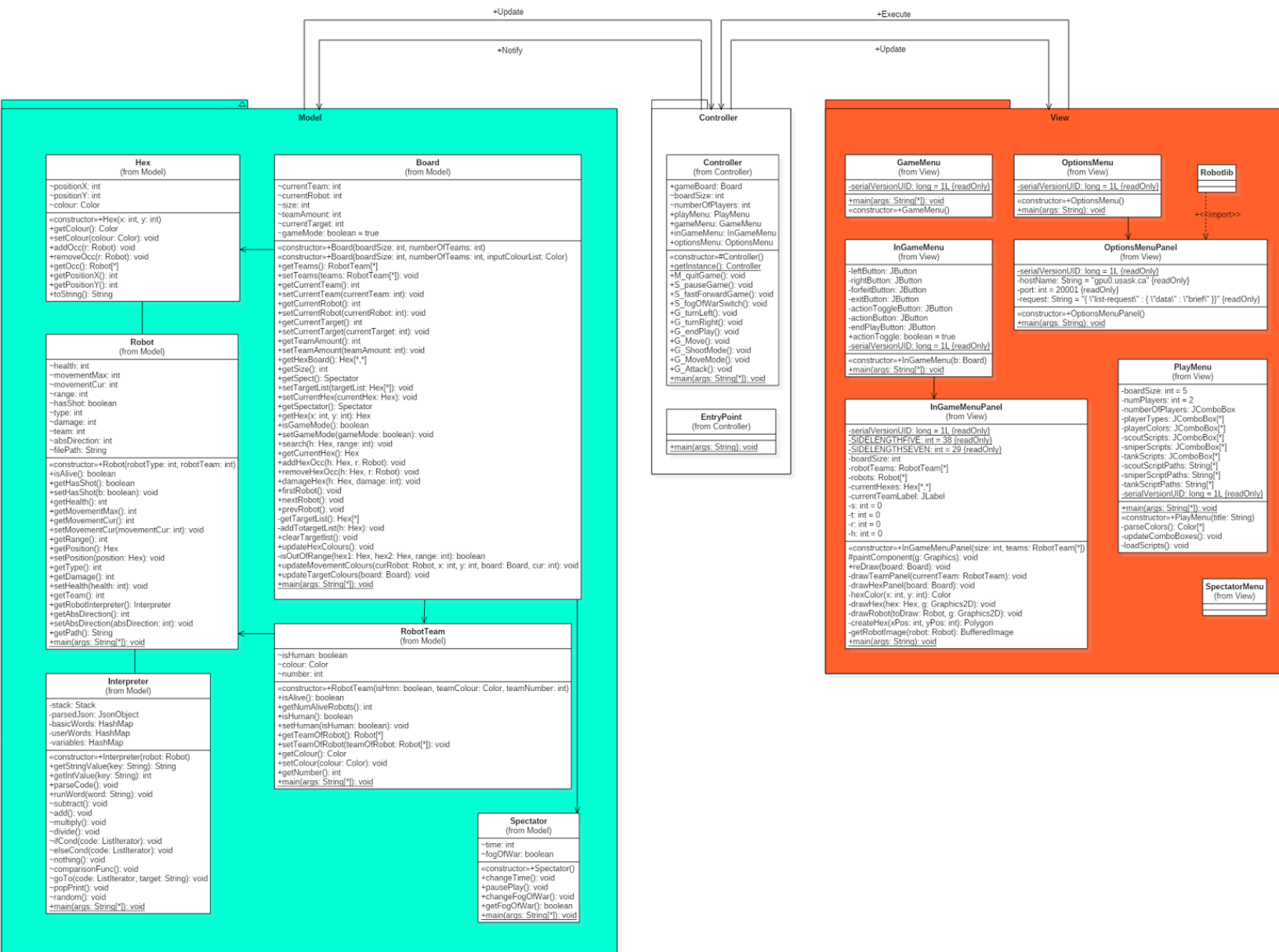
[INSERT PICTURE OF EXAMPLE ROBOT SCRIPT]

The robot scripts themselves make use of the .json file for storing their stats and behaviours. These scripts are parsed upon import to the system, and updated when a game is completed to keep certain stats, such as matches played, current. Each robot’s behavior script is based on a modified version of Forth, the RoboSport370 language. All behavior scripts MUST contain a play() function, which is called to notify the robot that it is their turn to play. All other functionality is entirely up to the writer, though they will have to take the game’s specifications into account (for example, it would be unwise to create a script with a board size of nine in mind, as our system only supports five and seven).

Though the system is complete and functional, there are several restrictions our system is bound by. For one, the system does not support robot mailboxes that some other implementations support, meaning any scripts that make use of these will not run correctly. Though mailboxes were never originally in our design, they were part of the assignment specifications.

Discuss limitations and restrictions of our system-----

Maintenance Document



Architecture of our system (Including new UML)

Our system follows the model-view controller architecture as originally designed. This architecture helps encapsulate the data into one area, as well as ensuring that all user-interaction is handled separately from the rest of the system. As well, this architecture keeps all game logic and flow in one designated piece: the controller. This gives our system much more flexibility in terms of maintenance and expansion, as each component has a sole purpose which can be modified without harming the rest of the system.

There are several classes that are much more demanding than others. For instance, our game's board class stores almost all information pertinent to the game, and simply makes use of several other classes in the model for helping store the information. To achieve this, it contains a number of functions for manipulation of the board's data, such as moving robots or scanning hexes for targets. For anyone looking to extend aspects of the board's representation, the functionality of robots, or even how robot teams are stored, the board class would be the place to begin.

Another intricate part of the model is the interpreter class. Robot scripts that our system uses implement the RoboSport370 language, which is far different than our system's native language, Java. The purpose of the interpreter is to intercept incoming and outgoing commands for each robot and serves as a translator for this language and the rest of the system. It performs this by parsing each script, breaking them down into functional words, then feeding either the robot or the system the input/output based on what is read or produced. While the core functionality should remain mostly the same, extensions in this class could provide the system with a wider range of possible scripts, better handling of invalid scripts, and more.

Apart from the model, another intricate piece of coding is the system's controller class. This class is responsible for all the game's logic, flow, manipulation, and serves as the in-between for the model and the view. Input from the user is gathered via the view, then forwarded to the controller which modifies the model appropriately. This class was implemented as a singleton, providing ease of access from both the view and the model. For someone looking to modify the rules of the game, change how the game progresses, or add to what the user can do, the controller would be the place to begin.

Finally, one last major component of the system is in-game menu. This class, along with its panel counterpart, are responsible for drawing the current state of the board, and providing the user with buttons for performing robot operations like moving and shooting. On top of this, the in-game menu also displays information on the current team, the targeted hex, and the current team playing. To do this, functionality is implemented for redrawing the board whenever changes occur. Since this class handles the visual display of this, possible extensions include changing the look of the board, the robots, or the overall GUI itself. As well, if additional functionality is added to the controller that requires user input, a way of gathering that input will be required in this menu.

Important parts of our system (Interpreter, board, controller, view)

External libraries/file format usage

Execution of our system

Tips/Ideas for future maintenance or extension

Changes:

- **Model:**

- **Hex:**

The hex class now contains a colour attribute, used for determining if the colour of the hex when determining either the fog-of-war or the targeting system. The hex also includes a toString method for testing, and a constructor used in construction of the board.

- **Robot:**

The robot class has changed significantly since the design document. The class now contains 2 new field variables. These variables are hasShot, which stores the boolean value of if the current robot has shot in it's current play, and filePath, which stores the file path of the json file to be parsed if the robot is an AI. The class also now includes getter and setter methods for these two fields.

- **Robot Team:**

The Robot team has a new integer field now that stores the index number of the team in the array of teams in the board. This allows for easier access to the robot team in other methods throughout our system. New methods to the class include the constructor, the getNumAliveRobots method, and the getter method for the team number field. The constructor takes in values for if the team is human, the colour of the team, and the index number of the team. The getNumAliveRobots method returns the number of robots that are still alive in the team.

- **Spectator:**

The Spectator class was unfortunately not used and was not changed from our design.

- **Board:**

The Board class was changed drastically from our design. Although no more field variables were added there were 9 new methods implemented. The new methods include two constructors, firstRobot, clearTargetList, addToTargetList, updateHexColours, isOutOfRange, updateMovementColours, and updateTargetColours. One of the constructors is used when the colours of the teams are not known and creates a board with the default colour options (this constructor is only used for test purposes). The second constructor constructs a board with specified attributes needed to be known for the creation of teams playing the game

- **Interpreter:**

The Interpreter

Requirements that were not met (project-wise and design-wise):

Designs that were altered/removed during construction

Remaining bugs:

Here is a list of the remaining bugs that still exist in the system that we are aware of but were unable to fix.