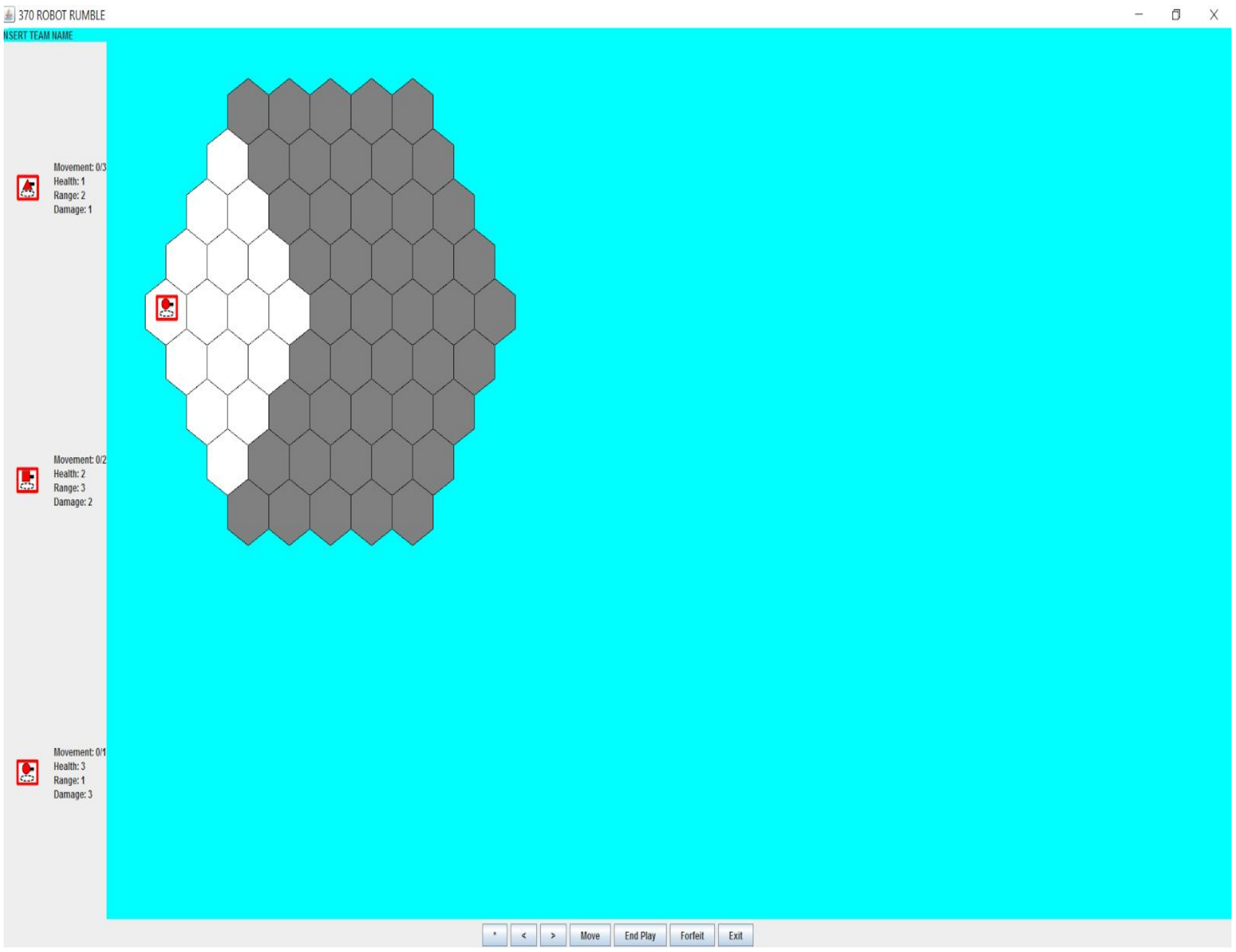
Deployment and Maintenance Document

CMPT 370

Group C4
Jack Huang
Brandon Jamieson
Ixabat Lahiji
Daniel Morris
Kevin Noonan

User Manual

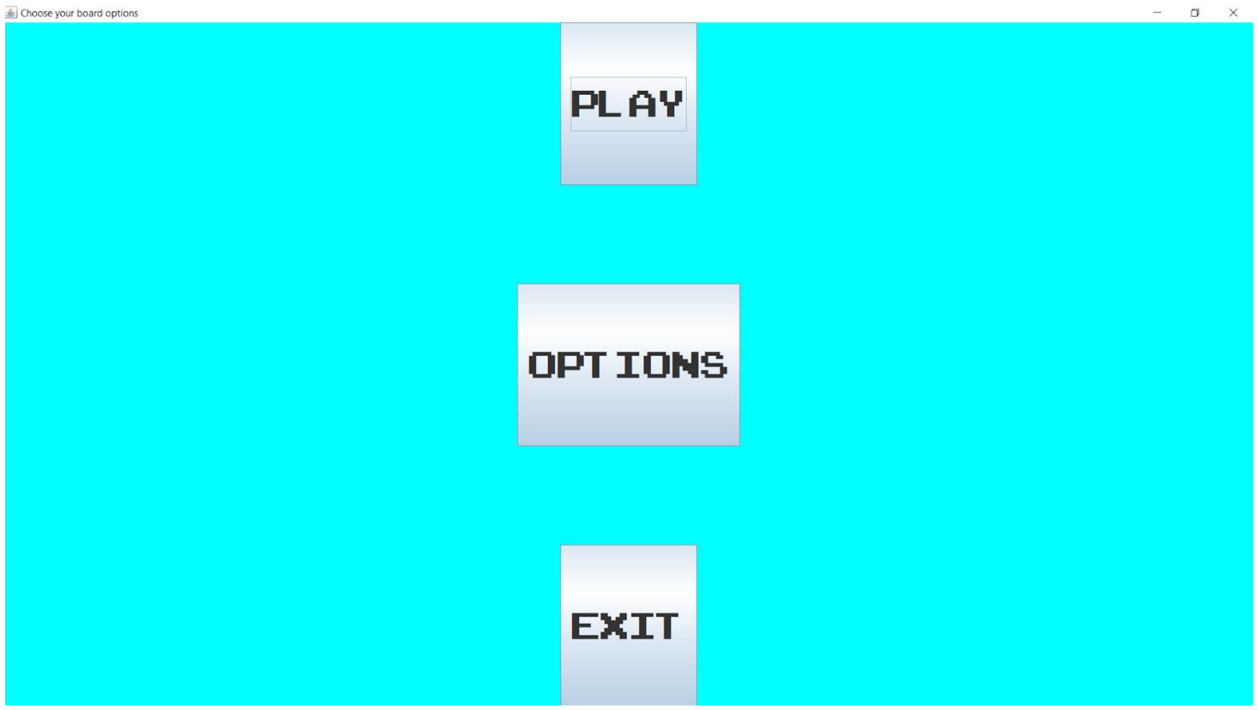
The game itself is played on a hexagonal board, with all three of each team's robots starting in opposite corners. Robots have varying stats like movement points, damage, and range. Each round, players will move and shoot with their fastest available tank, until all tanks have spent their movement points. 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.



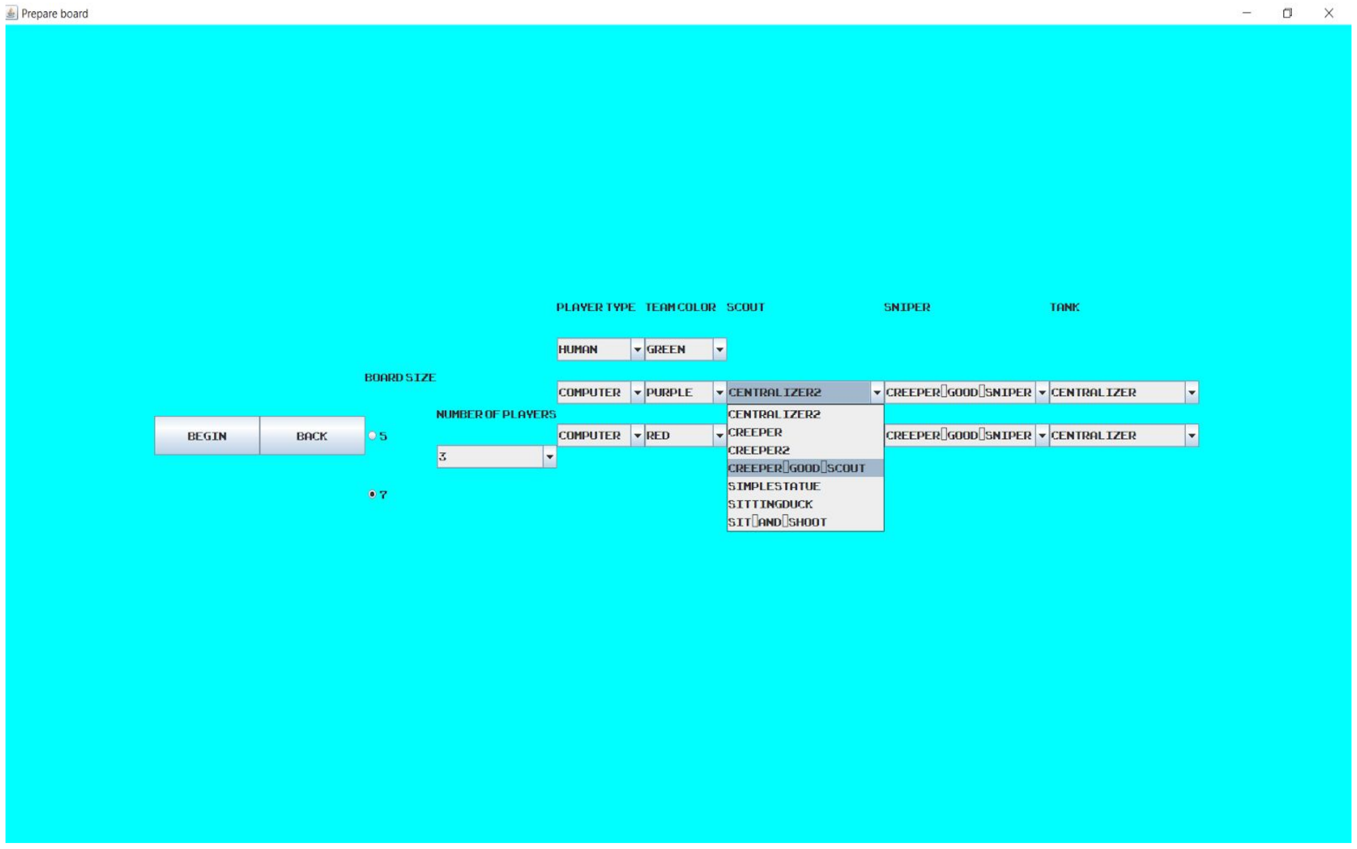
Movement: 0/3
Health: 1
Range: 2
Damage: 1

Movement: 0/2
Health: 2
Range: 3
Damage: 2

Movement: 0/1
Health: 3
Range: 1
Damage: 3



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.


Prepare board

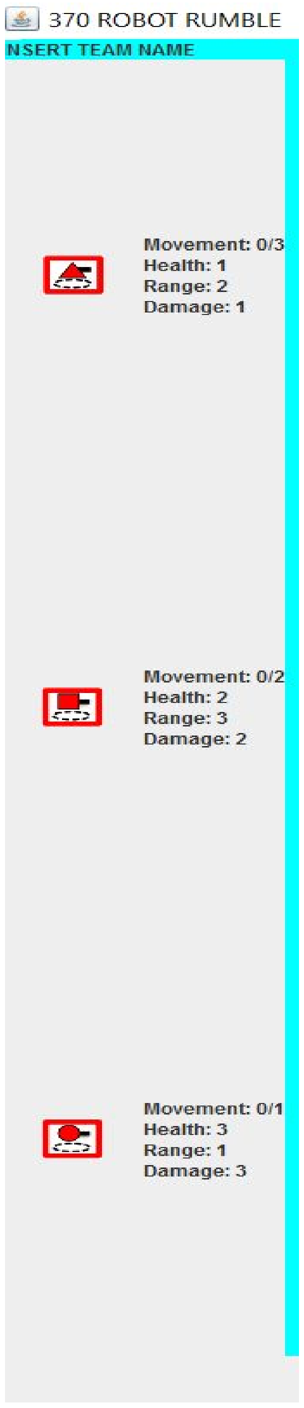
BEGIN BACK

BOARD SIZE: 5 [✓ 7]

NUMBER OF PLAYERS: 3

PLAYER TYPE	TEAM COLOR	SCOUT	SNIPER	TANK
HUMAN	GREEN			
COMPUTER	PURPLE	CENTRALIZER2	CREEPER□GOOD□SNIPER	CENTRALIZER
COMPUTER	RED		CREEPER□GOOD□SNIPER	CENTRALIZER

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.



Movement: 0/3
Health: 1
Range: 2
Damage: 1

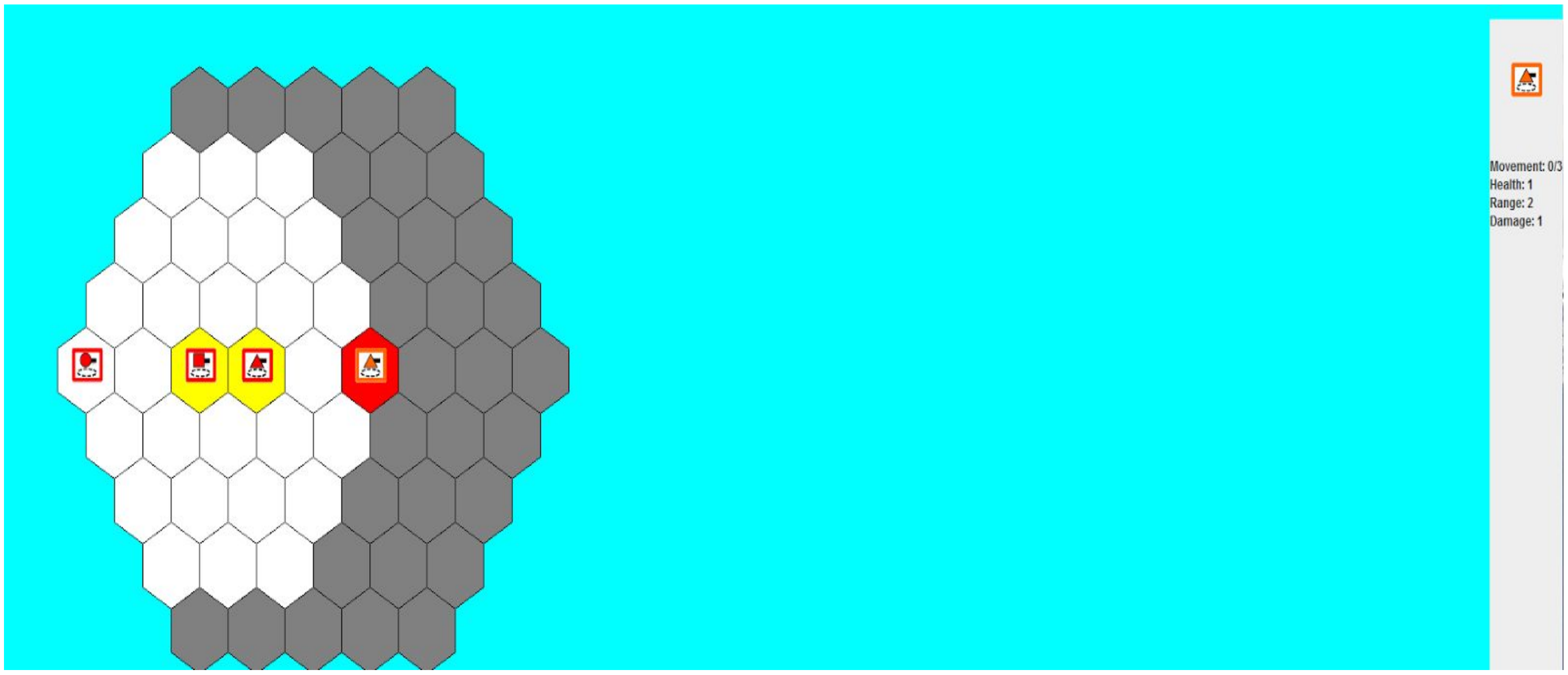
Movement: 0/2
Health: 2
Range: 3
Damage: 2

Movement: 0/1
Health: 3
Range: 1
Damage: 3

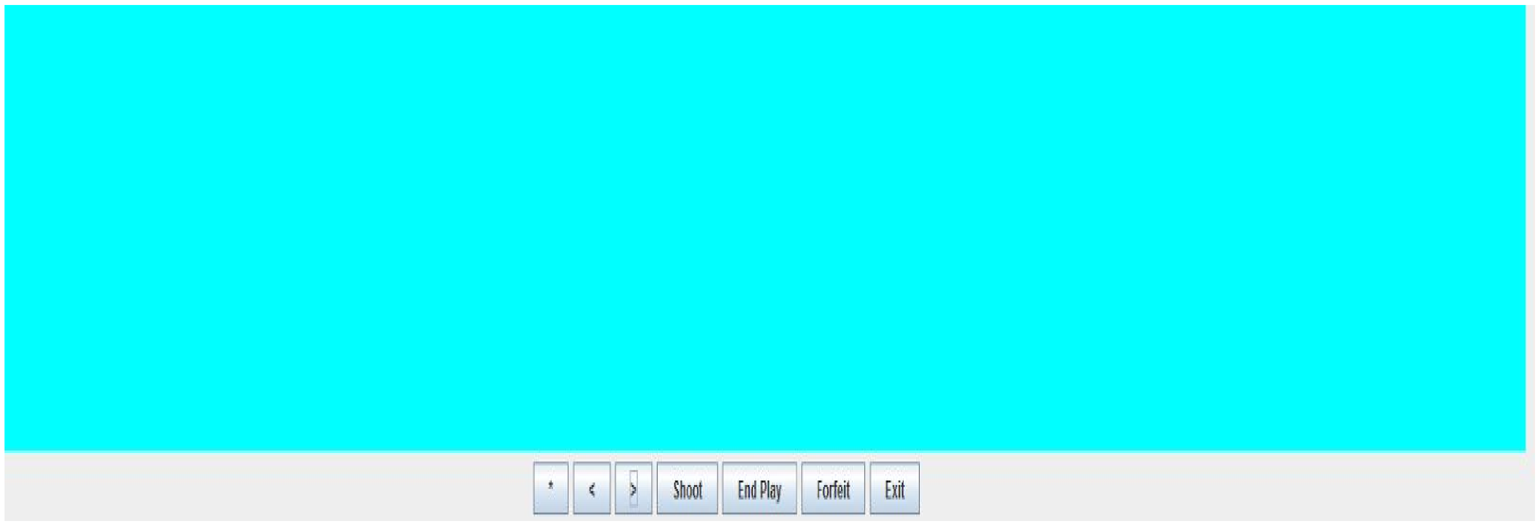
Once the user has selected their preferences, they can press the play button to start.

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.

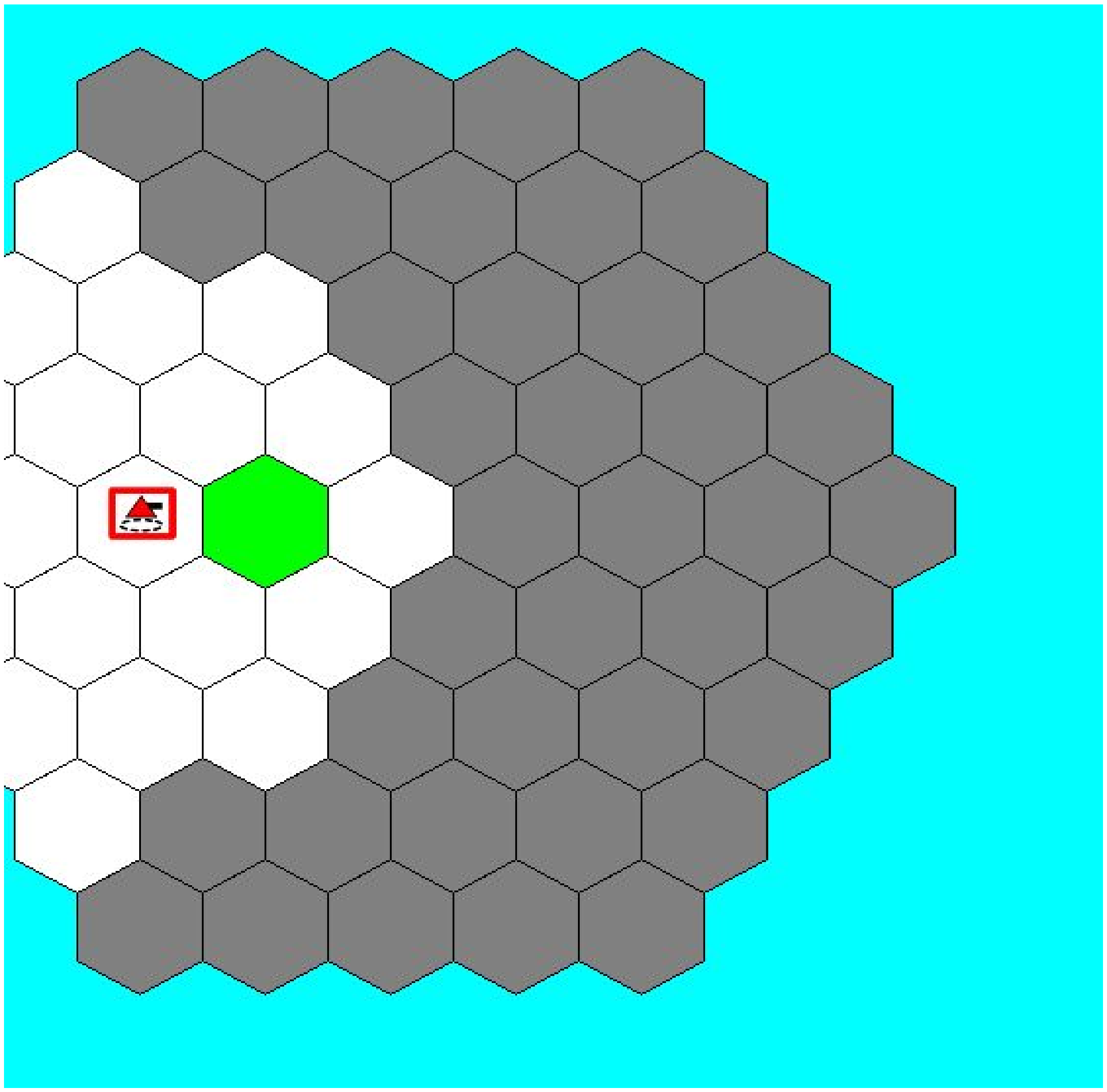
The team display panel to the left of the board in the GUI provides information on the current player’s robots. For each robot, there will be an image representing their respective game piece as well as their stats. Each robot’s attack, range, movement points, and health are displayed beside their image. 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.



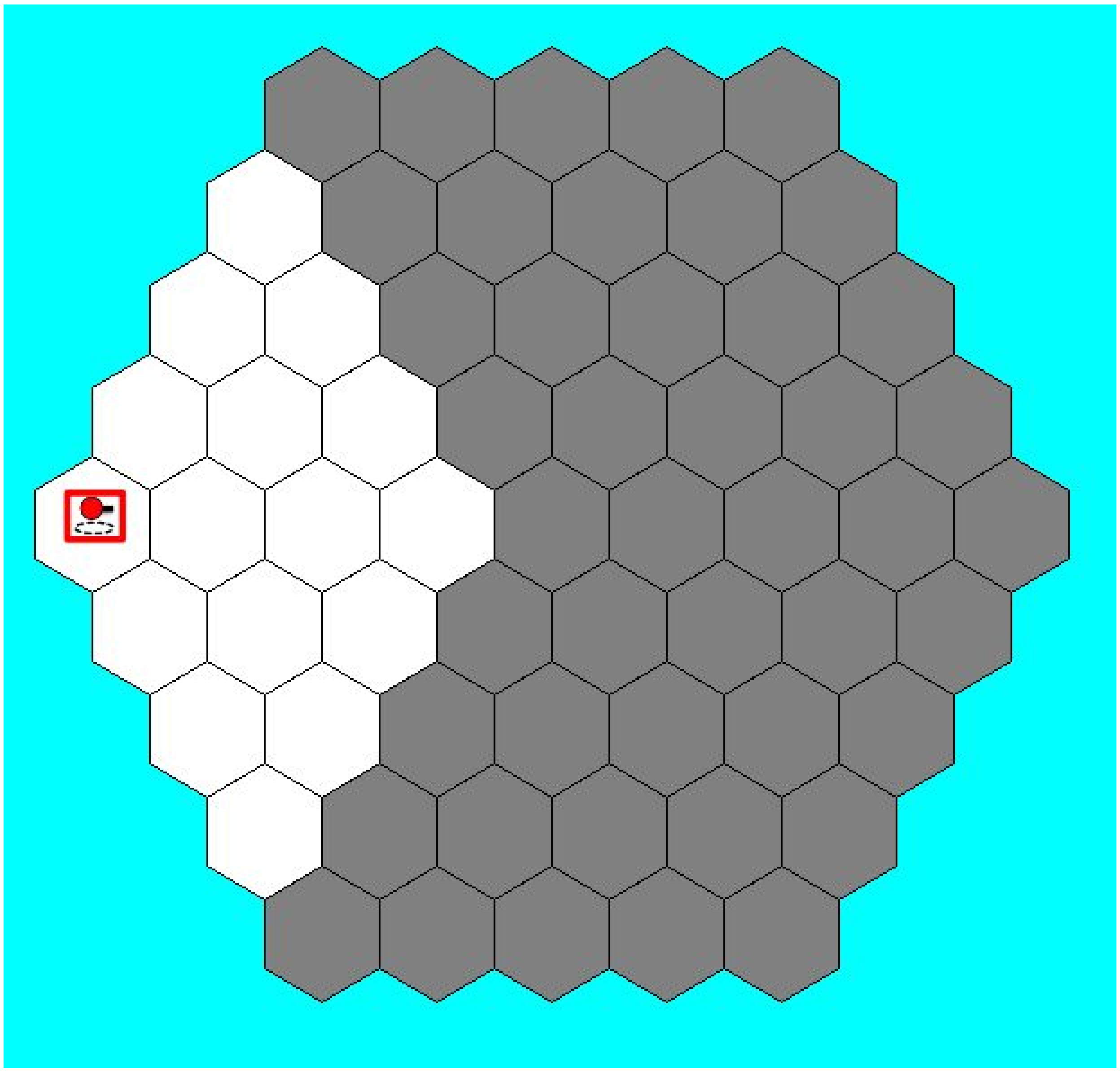
The hex indicator panel is on the right side of the board in the GUI, and provides information on the currently targeted. Hexes become targeted when the current player enters shooting mode and aims at an occupied hex. Each occupant is shown, along with their robot image in their team's color. This allows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



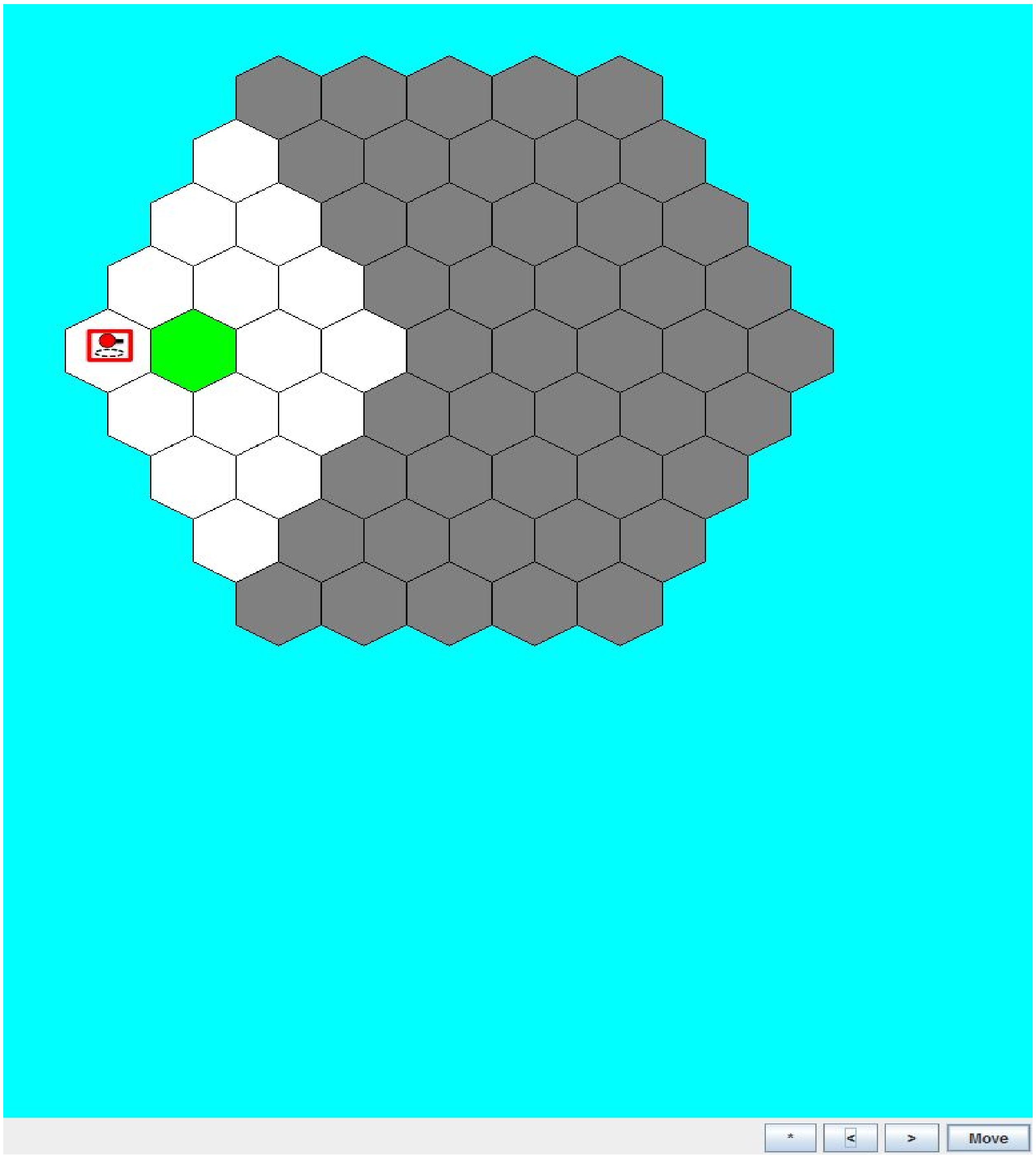
The buttons underneath the hexagon board are for human player's to manipulate their current robot. The "*" button will toggle between moving and shooting mode, changing the action button to represent the current mode. The action button is in the middle, and will either say "Shoot" or "Move" depending 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.

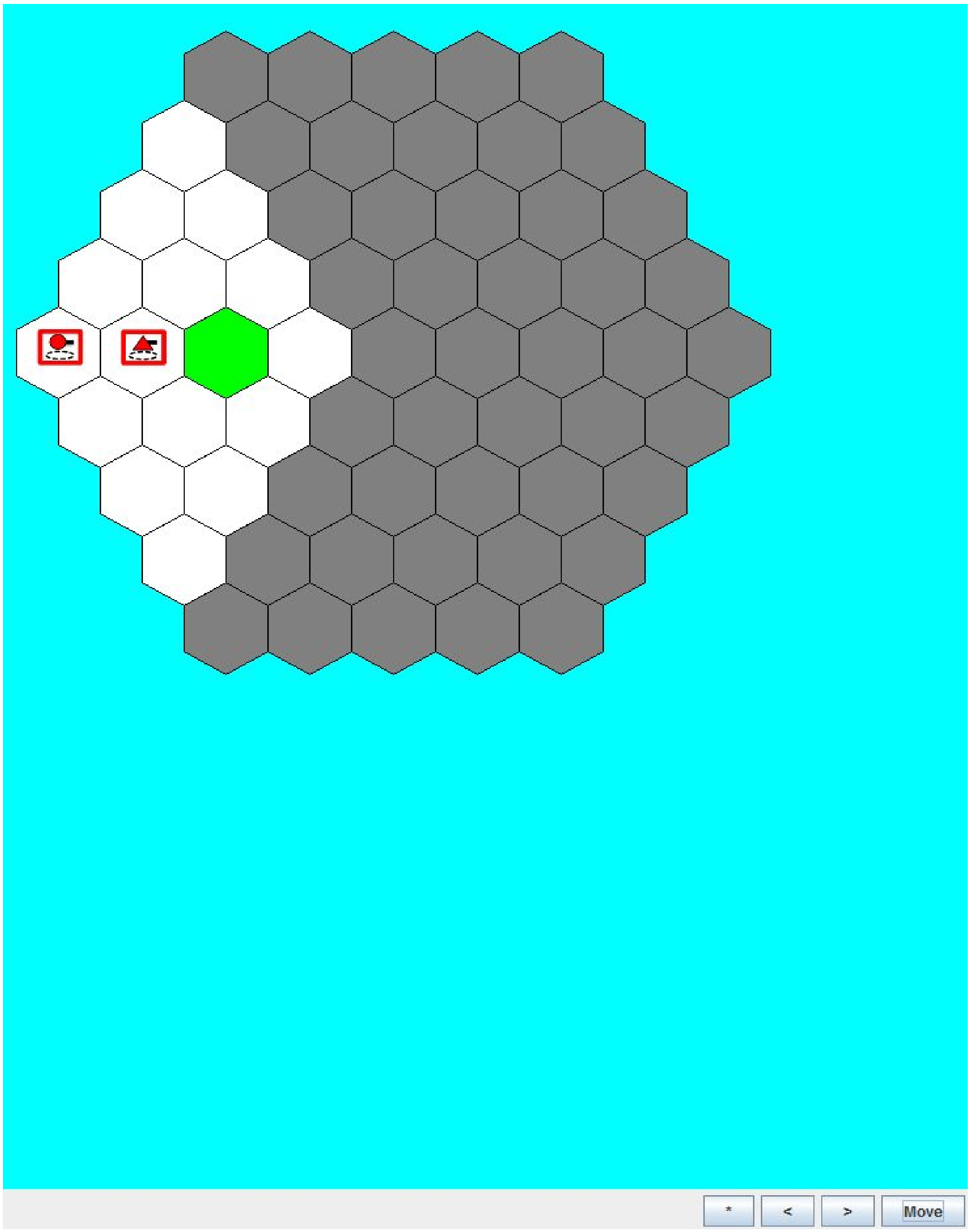


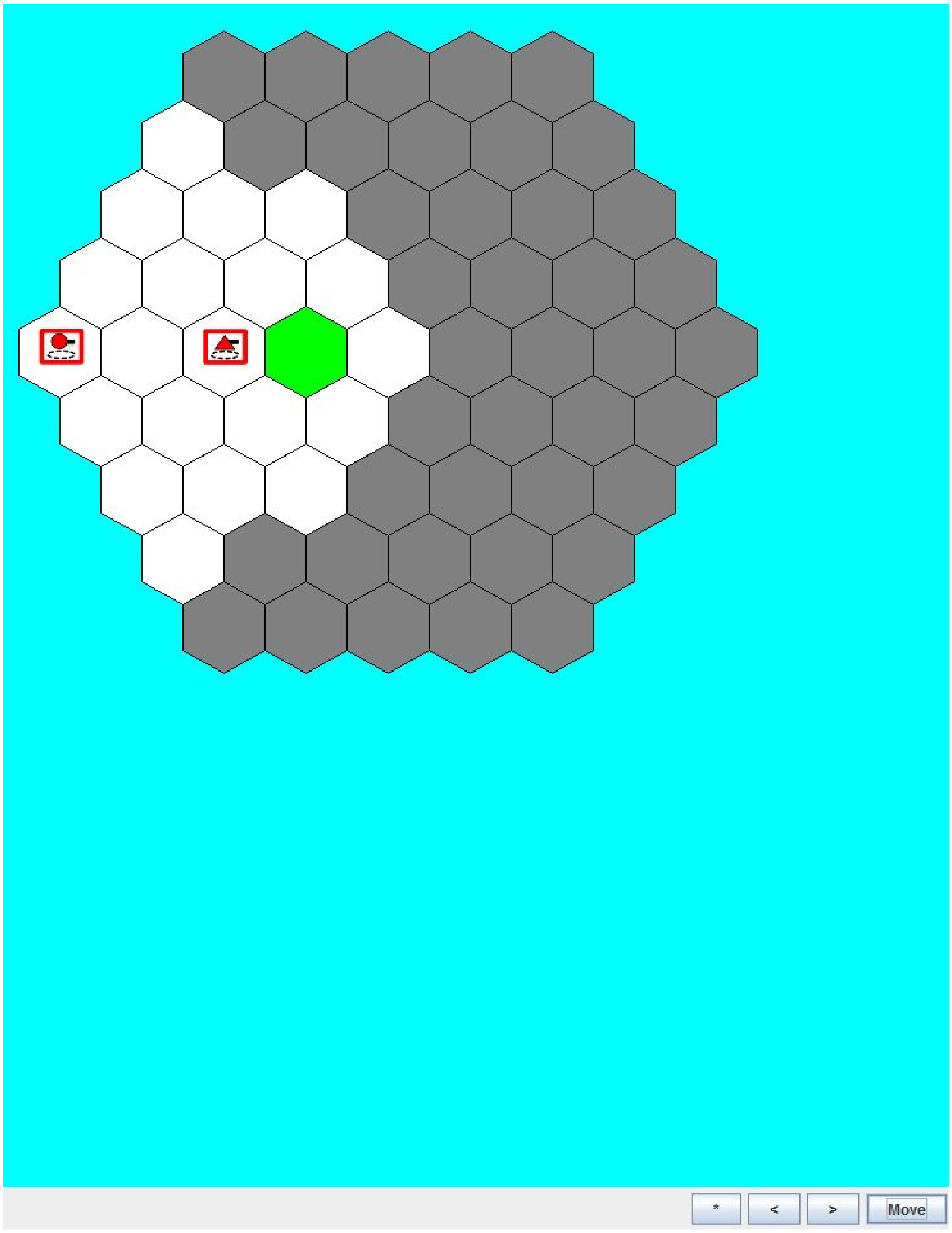
The board itself is initially grayed out for the most part. This is the game's "Fog of War", which hides hexagons that are not within range of any of the current player's robots. This means players will have to explore the map to find enemy robots before they can shoot them. Players can also use this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

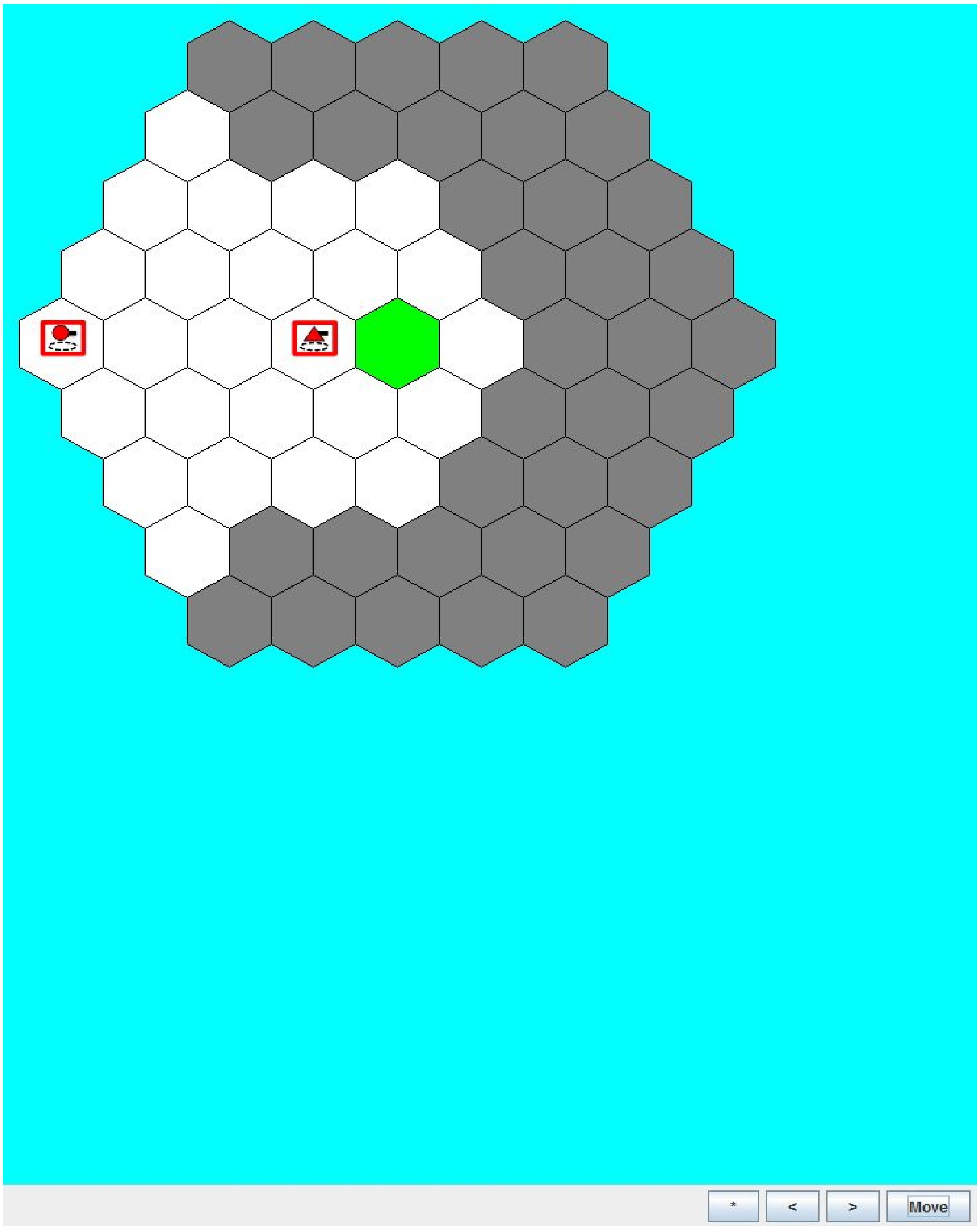


Once the game begins, the user (assuming player one is a human) will take control of the quickest robot on the team: initially the scout. The scout has three movement points initially, so the user may move three times before they can no longer move. Robots may shoot once per turn, although since there 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.

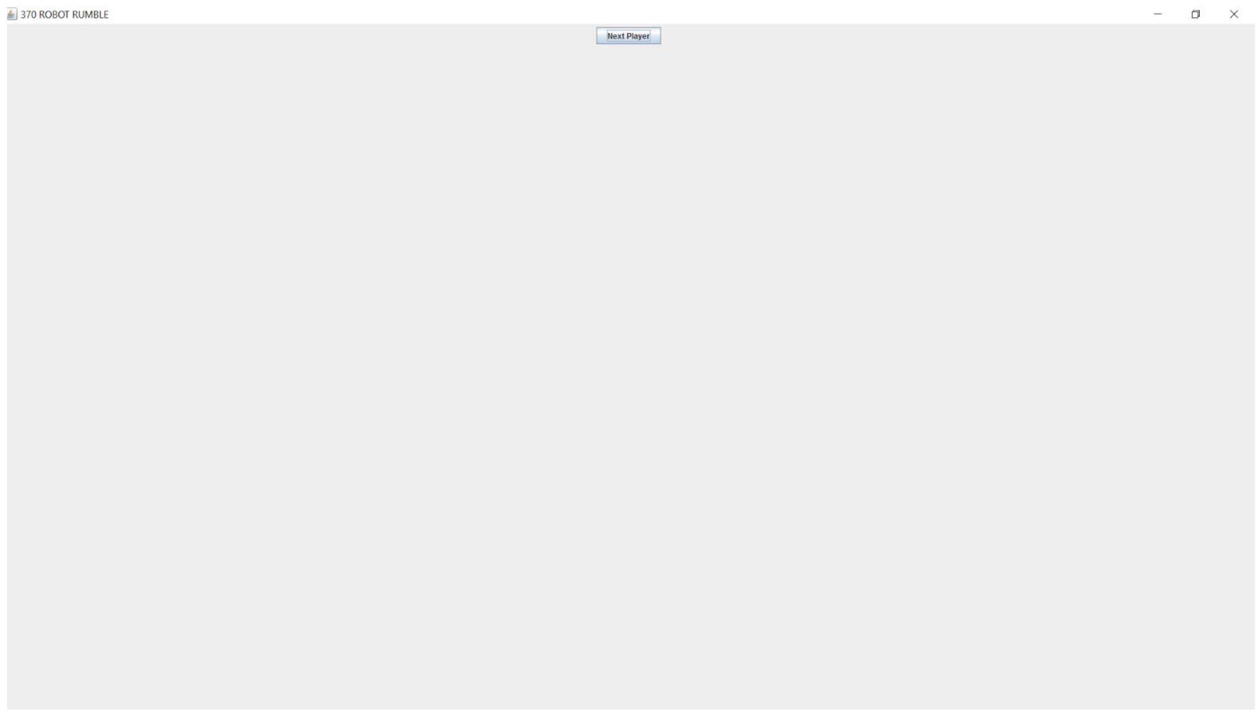



*
<
>
Move

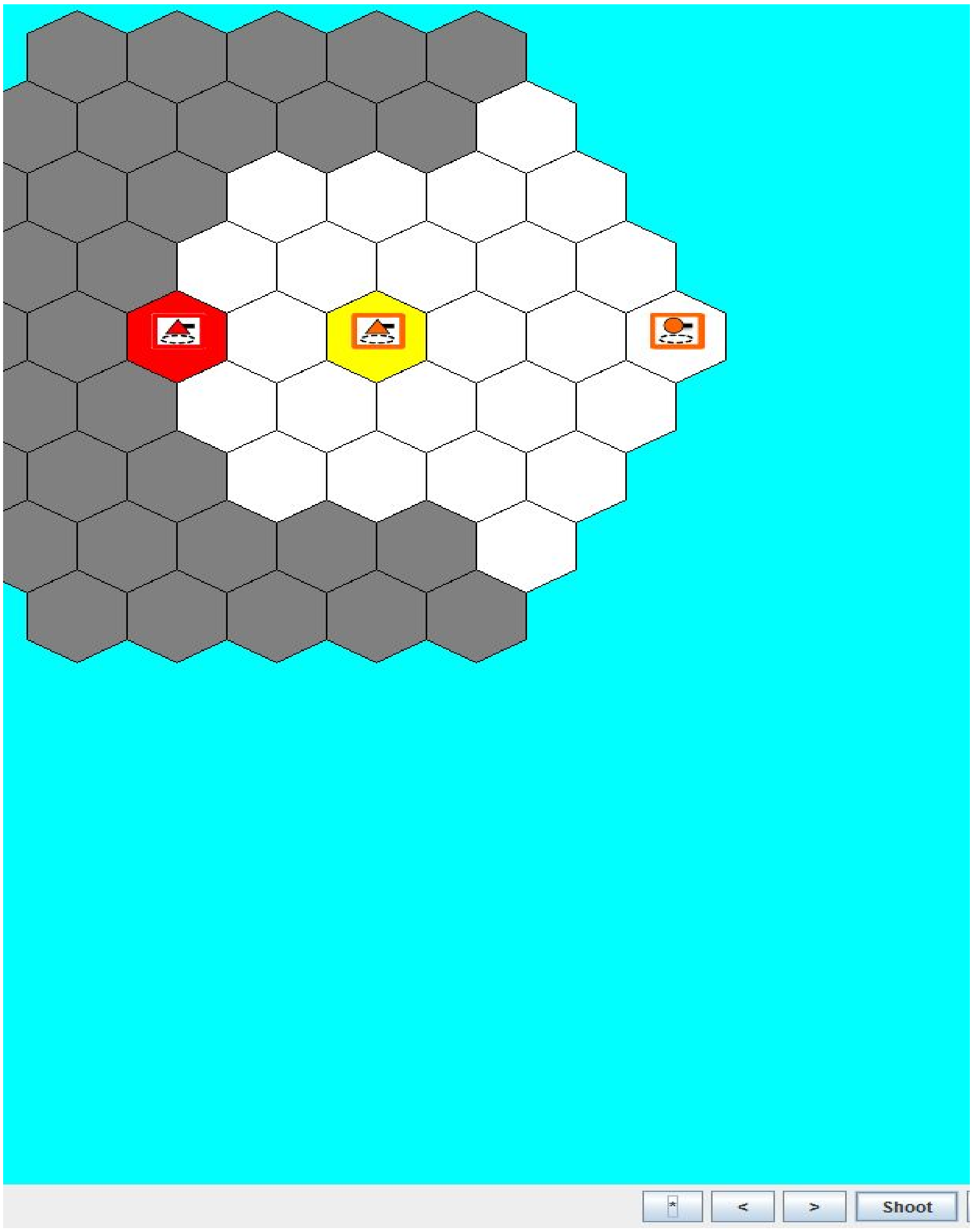

*
<
>
Move

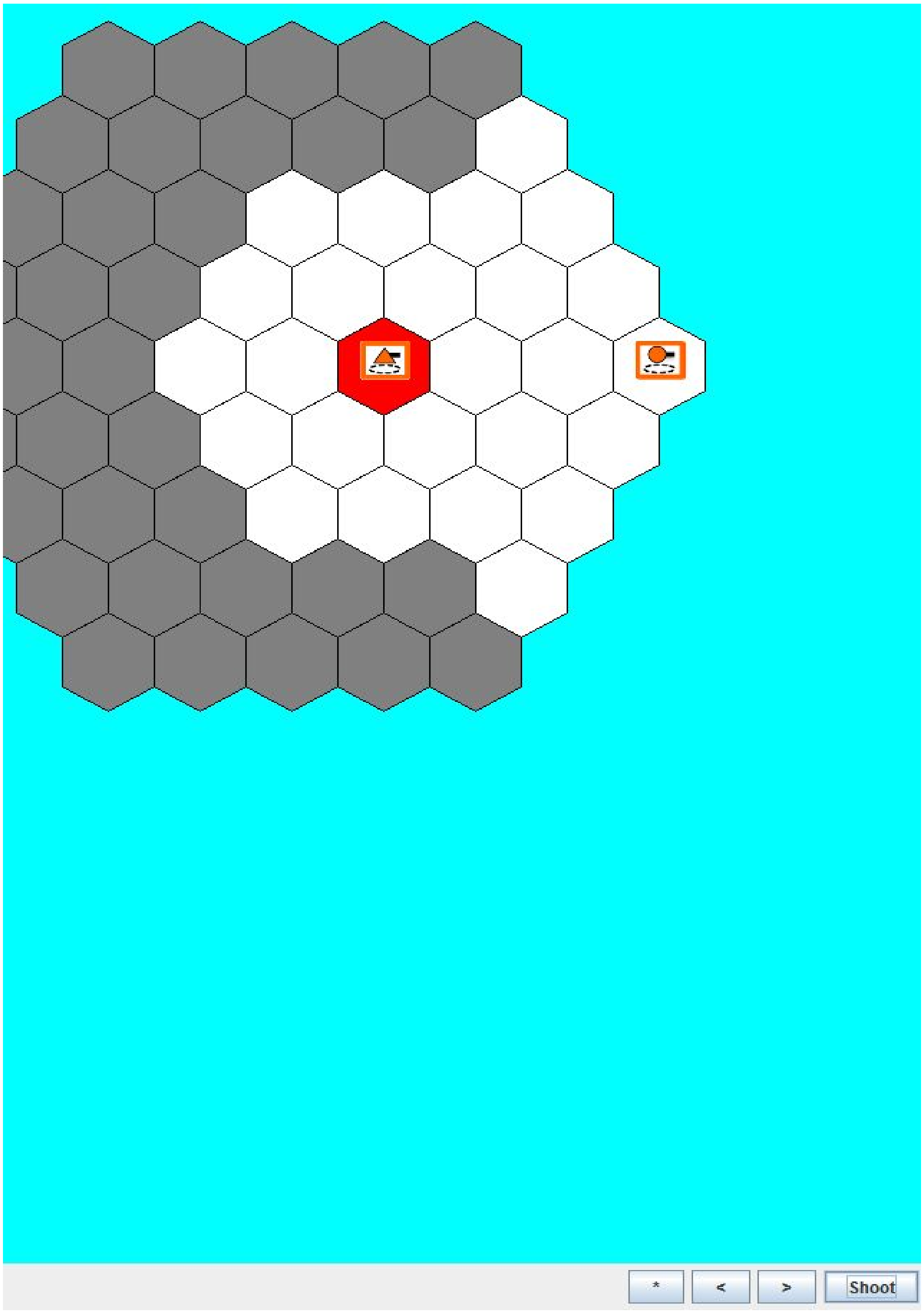

*
<
>
Move

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.

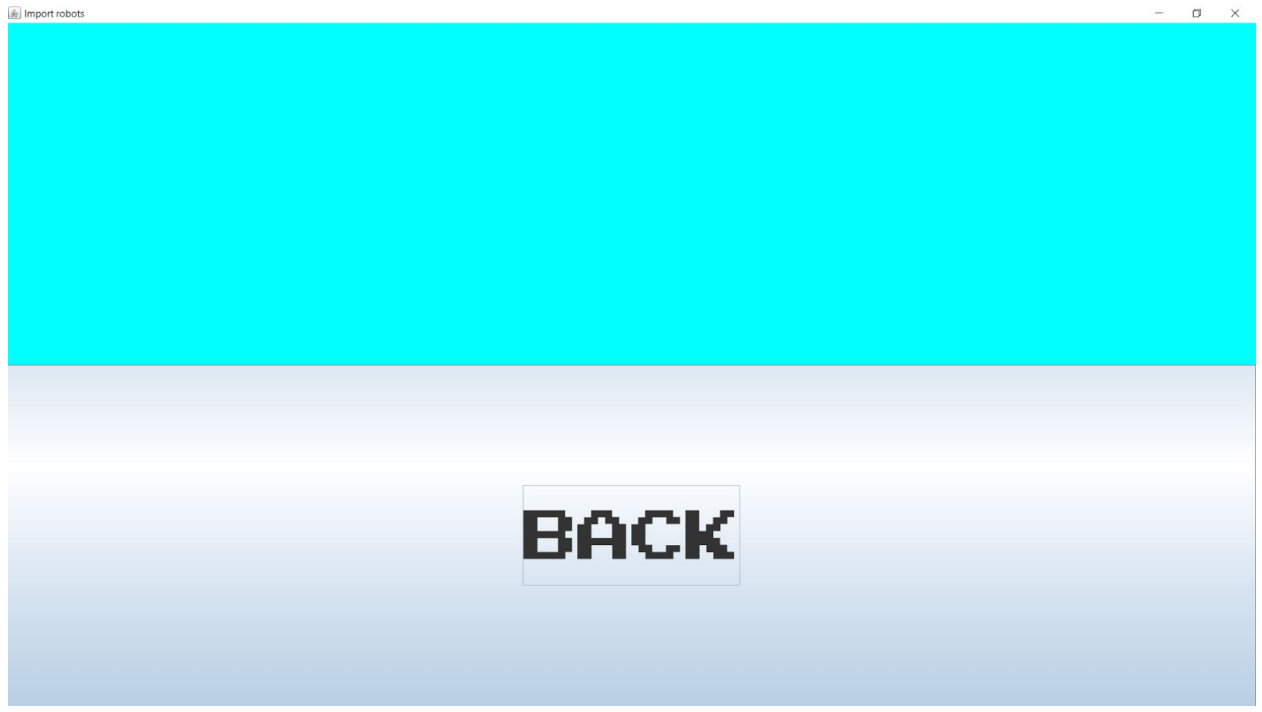


Once the players have switched, the new current player will assume control of their scout and make their play. This process continues until all scouts have made their plays, then repeats with sniper and tank robots.If a robot has been shot and no longer has any health, that robot is destroyed and will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

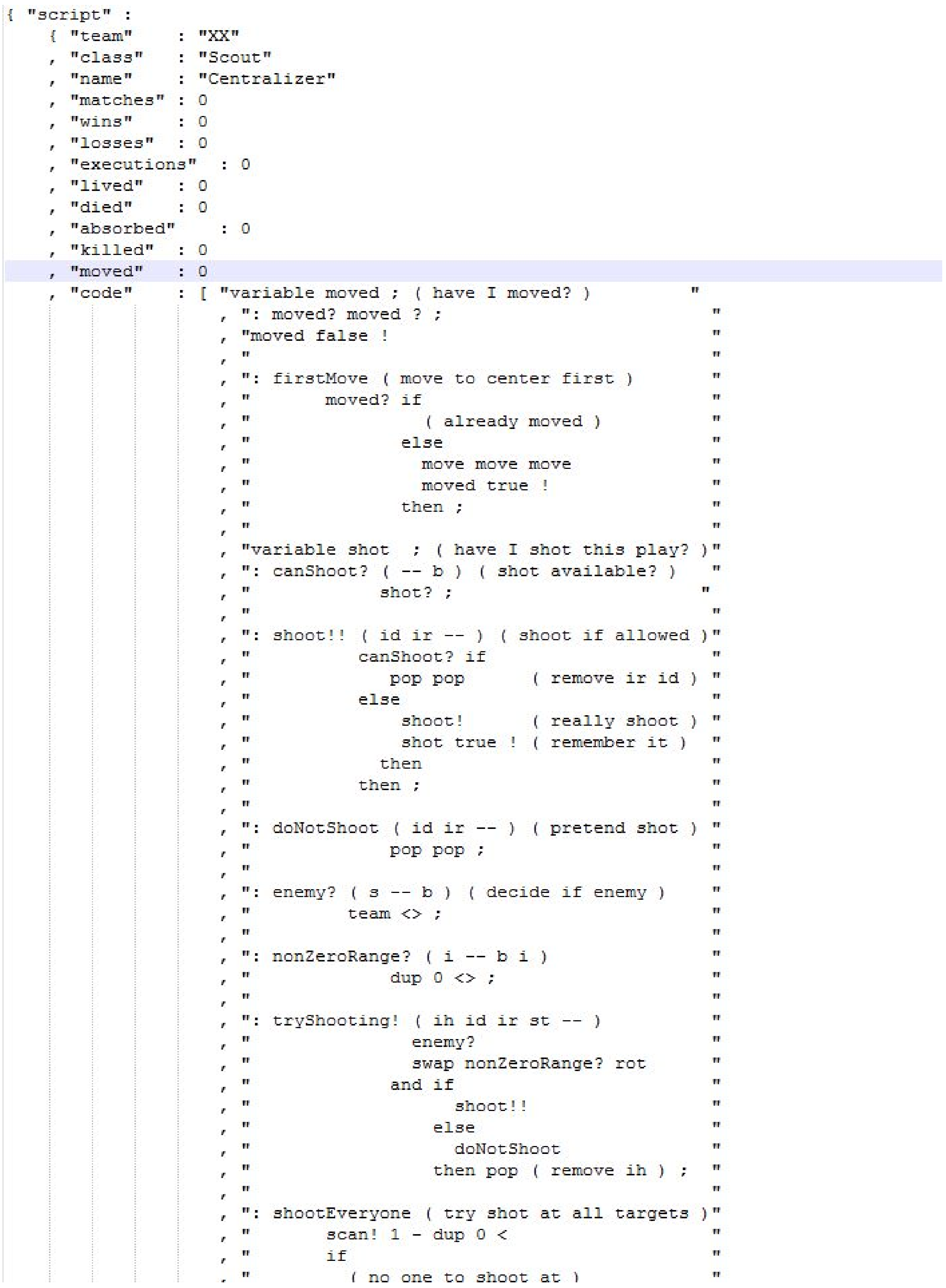

*
<
>
Shoot


*
<
>
Shoot

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.



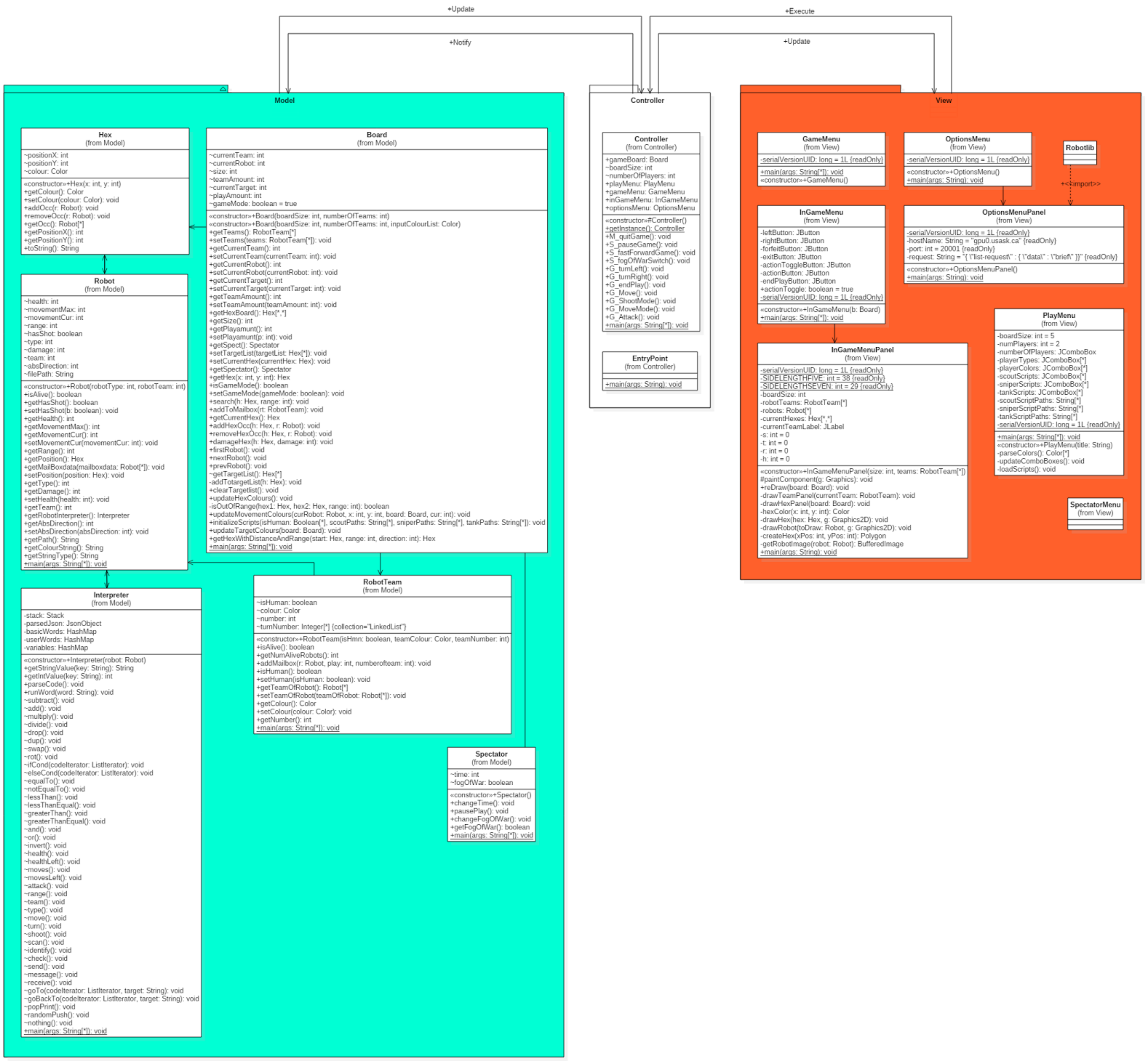
Users may also import robot scripts from a remote server via the options menu for use in-game. The default robot server to connect to has the host name "gpu0.usask.ca", and is stationed at the U of S. If the server is down, or not available from the current location, the options screen will simply be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



```
{ "script" :
    { "team"    : "XX"
    , "class"   : "Scout"
    , "name"    : "Centralizer"
    , "matches" : 0
    , "wins"    : 0
    , "losses"  : 0
    , "executions"  : 0
    , "lived"   : 0
    , "died"    : 0
    , "absorbed"    : 0
    , "killed"  : 0
    , "moved"   : 0
    , "code"    : [ "variable moved ; ( have I moved? )         "
                    , ": moved? moved ? ;                         "
                    , "moved false !                              "
                    , "                                           "
                    , ": firstMove ( move to center first )       "
                    , "       moved? if                           "
                    , "                ( already moved )          "
                    , "              else                         "
                    , "                move move move             "
                    , "                moved true !               "
                    , "              then ;                       "
                    , "                                           "
                    , "variable shot  ; ( have I shot this play? )"
                    , ": canShoot? ( -- b ) ( shot available? )   "
                    , "            shot? ;                       "
                    , "                                           "
                    , ": shoot!! ( id ir -- ) ( shoot if allowed )"
                    , "          canShoot? if                     "
                    , "             pop pop      ( remove ir id ) "
                    , "          else                             "
                    , "              shoot!      ( really shoot ) "
                    , "              shot true ! ( remember it )  "
                    , "            then                           "
                    , "          then ;                           "
                    , "                                           "
                    , ": doNotShoot ( id ir -- ) ( pretend shot ) "
                    , "             pop pop ;                     "
                    , "                                           "
                    , ": enemy? ( s -- b ) ( decide if enemy )    "
                    , "         team <> ;                         "
                    , "                                           "
                    , ": nonZeroRange? ( i -- b i )               "
                    , "             dup 0 <> ;                    "
                    , "                                           "
                    , ": tryShooting! ( ih id ir st -- )          "
                    , "               enemy?                      "
                    , "               swap nonZeroRange? rot      "
                    , "             and if                        "
                    , "                   shoot!!                 "
                    , "                 else                      "
                    , "                   doNotShoot              "
                    , "                 then pop ( remove ih ) ;  "
                    , "                                           "
                    , ": shootEveryone ( try shot at all targets )"
                    , "       scan! 1 - dup 0 <                   "
                    , "       if                                  "
                    , "         ( no one to shoot at )            "
```

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 is restricted to importing scripts from a remote server, or manually adding them to the folder's script directory. It would have been nice to have a way to change the location of the server through the options, instead of having to modify the code itself. Another restriction of our system is that multiplayer is restricted to a single computer. While we never intended to include networking in our design, it certainly would have been a benefit overall.

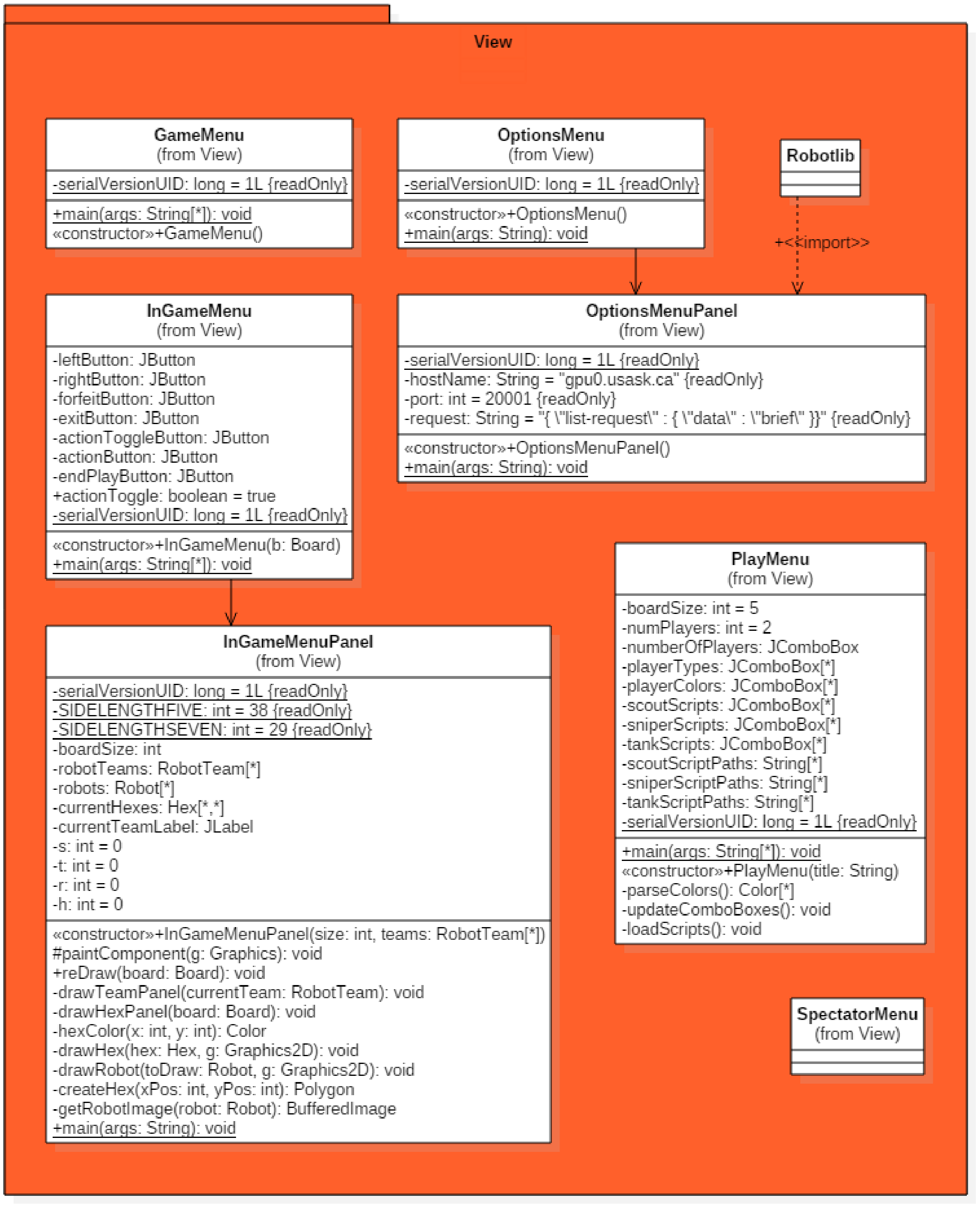

+Update
+Notify
+Execute
+Update
Model
Hex
(from Model)
~positionX: int
~positionY: int
~colour: Color
«constructor»+Hex(x: int, y: int)
+getColour(): Color
+setColour(colour: Color): void
+addOcc(r: Robot): void
+removeOcc(r: Robot): void
+getOcc(): Robot[*]
+getPositionX(): int
+getPositionY(): int
+toString(): String
Robot
(from Model)
~health: int
~movementMax: int
~movementCur: int
~range: int
~hasShot: boolean
~type: int
~damage: int
~team: int
~absDirection: int
~filePath: String
«constructor»+Robot(robotType: int, robotTeam: int)
+isAlive(): boolean
+getHasShot(): boolean
+setHasShot(b: boolean): void
+getHealth(): int
+getMovementMax(): int
+getMovementCur(): int
+setMovementCur(movementCur: int): void
+getRange(): int
+getPosition(): Hex
+getMailBoxdata(mailboxdata: Robot[*]): void
+setPosition(position: Hex): void
+getType(): int
+getDamage(): int
+setHealth(health: int): void
+getTeam(): int
+getRobotInterpreter(): Interpreter
+getAbsDirection(): int
+setAbsDirection(absDirection: int): void
+getPath(): String
+getColourString(): String
+getStringType(): String
+main(args: String[*]): void
Interpreter
(from Model)
-stack: Stack
-parsedJson: JsonObject
-basicWords: HashMap
-userWords: HashMap
-variables: HashMap
«constructor»+Interpreter(robot: Robot)
+getStringValue(key: String): String
+getIntValue(key: String): int
+parseCode(): void
+runWord(word: String): void
~subtract(): void
~add(): void
~multiply(): void
~divide(): void
~drop(): void
~dup(): void
~swap(): void
~rot(): void
~ifCond(codeIterator: ListIterator): void
~elseCond(codeIterator: ListIterator): void
~equalTo(): void
~notEqualTo(): void
~lessThan(): void
~lessThanEqual(): void
~greaterThan(): void
~greaterThanEqual(): void
~and(): void
~or(): void
~invert(): void
~health(): void
~healthLeft(): void
~moves(): void
~movesLeft(): void
~attack(): void
~range(): void
~team(): void
~type(): void
~move(): void
~turn(): void
~shoot(): void
~scan(): void
~identify(): void
~check(): void
~send(): void
~message(): void
~receive(): void
~goTo(codeIterator: ListIterator, target: String): void
~goBackTo(codeIterator: ListIterator, target: String): void
~popPrint(): void
~randomPush(): void
~nothing(): void
+main(args: String[*]): void
Board
(from Model)
~currentTeam: int
~currentRobot: int
~size: int
~teamAmount: int
~currentTarget: int
~playAmount: int
~gameMode: boolean = true
«constructor»+Board(boardSize: int, numberOfTeams: int)
«constructor»+Board(boardSize: int, numberOfTeams: int, inputColourList: Color)
+getTeams(): RobotTeam[*]
+setTeams(teams: RobotTeam[*]): void
+getCurrentTeam(): int
+setCurrentTeam(currentTeam: int): void
+getCurrentRobot(): int
+setCurrentRobot(currentRobot: int): void
+getCurrentTarget(): int
+setCurrentTarget(currentTarget: int): void
+getTeamAmount(): int
+setTeamAmount(teamAmount: int): void
+getHexBoard(): Hex[*,*]
+getSize(): int
+getPlayamunt(): int
+setPlayamunt(p: int): void
+getSpect(): Spectator
+setTargetList(targetList: Hex[*]): void
+setCurrentHex(currentHex: Hex): void
+getSpectator(): Spectator
+getHex(x: int, y: int): Hex
+isGameMode(): boolean
+setGameMode(gameMode: boolean): void
+search(h: Hex, range: int): void
+addToMailbox(rt: RobotTeam): void
+getCurrentHex(): Hex
+addHexOcc(h: Hex, r: Robot): void
+removeHexOcc(h: Hex, r: Robot): void
+damageHex(h: Hex, damage: int): void
+firstRobot(): void
+nextRobot(): void
+prevRobot(): void
~getTargetList(): Hex[*]
-addTotargetList(h: Hex): void
+clearTargetlist(): void
+updateHexColours(): void
-isOutOfRange(hex1: Hex, hex2: Hex, range: int): boolean
+updateMovementColours(curRobot: Robot, x: int, y: int, board: Board, cur: int): void
+initializeScripts(isHuman: Boolean[*], scoutPaths: String[*], sniperPaths: String[*], tankPaths: String[*]): void
+updateTargetColours(board: Board): void
+getHexWithDistanceAndRange(start: Hex, range: int, direction: int): Hex
+main(args: String[*]): void
RobotTeam
(from Model)
~isHuman: boolean
~colour: Color
~number: int
~turnNumber: Integer[*] {collection="LinkedList"}
«constructor»+RobotTeam(isHmn: boolean, teamColour: Color, teamNumber: int)
+isAlive(): boolean
+getNumAliveRobots(): int
+addMailbox(r: Robot, play: int, numberofteam: int): void
+isHuman(): boolean
+setHuman(isHuman: boolean): void
+getTeamOfRobot(): Robot[*]
+setTeamOfRobot(teamOfRobot: Robot[*]): void
+getColour(): Color
+setColour(colour: Color): void
+getNumber(): int
+main(args: String[*]): void
Spectator
(from Model)
~time: int
~fogOfWar: boolean
«constructor»+Spectator()
+changeTime(): void
+pausePlay(): void
+changeFogOfWar(): void
+getFogOfWar(): boolean
+main(args: String[*]): void
Controller
Controller
(from Controller)
+gameBoard: Board
~boardSize: int
~numberOfPlayers: int
+playMenu: PlayMenu
+gameMenu: GameMenu
+inGameMenu: InGameMenu
+optionsMenu: OptionsMenu
«constructor»#Controller()
+getInstance(): Controller
+M_quitGame(): void
+S_pauseGame(): void
+S_fastForwardGame(): void
+S_fogOfWarSwitch(): void
+G_turnLeft(): void
+G_turnRight(): void
+G_endPlay(): void
+G_Move(): void
+G_ShootMode(): void
+G_MoveMode(): void
+G_Attack(): void
+main(args: String[*]): void
EntryPoint
(from Controller)
+main(args: String): void
View
GameMenu
(from View)
-serialVersionUID: long = 1L {readOnly}
+main(args: String[*]): void
«constructor»+GameMenu()
OptionsMenu
(from View)
-serialVersionUID: long = 1L {readOnly}
«constructor»+OptionsMenu()
+main(args: String): void
Robotlib
+<<import>>
InGameMenu
(from View)
-leftButton: JButton
-rightButton: JButton
-forfeitButton: JButton
-exitButton: JButton
-actionToggleButton: JButton
-actionButton: JButton
-endPlayButton: JButton
+actionToggle: boolean = true
-serialVersionUID: long = 1L {readOnly}
«constructor»+InGameMenu(b: Board)
+main(args: String[*]): void
OptionsMenuPanel
(from View)
-serialVersionUID: long = 1L {readOnly}
-hostName: String = "gpu0.usask.ca" {readOnly}
-port: int = 20001 {readOnly}
-request: String = "{ \"list-request\" : { \"data\" : \"brief\" }}" {readOnly}
«constructor»+OptionsMenuPanel()
+main(args: String): void
InGameMenuPanel
(from View)
-serialVersionUID: long = 1L {readOnly}
-SIDELENGTHFIVE: int = 38 {readOnly}
-SIDELENGTHSEVEN: int = 29 {readOnly}
-boardSize: int
-robotTeams: RobotTeam[*]
-robots: Robot[*]
-currentHexes: Hex[*,*]
-currentTeamLabel: JLabel
-s: int = 0
-t: int = 0
-r: int = 0
-h: int = 0
«constructor»+InGameMenuPanel(size: int, teams: RobotTeam[*])
#paintComponent(g: Graphics): void
+reDraw(board: Board): void
-drawTeamPanel(currentTeam: RobotTeam): void
-drawHexPanel(board: Board): void
-hexColor(x: int, y: int): Color
-drawHex(hex: Hex, g: Graphics2D): void
-drawRobot(toDraw: Robot, g: Graphics2D): void
-createHex(xPos: int, yPos: int): Polygon
-getRobotImage(robot: Robot): BufferedImage
+main(args: String): void
PlayMenu
(from View)
-boardSize: int = 5
-numPlayers: int = 2
-numberOfPlayers: JComboBox
-playerTypes: JComboBox[*]
-playerColors: JComboBox[*]
-scoutScripts: JComboBox[*]
-sniperScripts: JComboBox[*]
-tankScripts: JComboBox[*]
-scoutScriptPaths: String[*]
-sniperScriptPaths: String[*]
-tankScriptPaths: String[*]
-serialVersionUID: long = 1L {readOnly}
+main(args: String[*]): void
«constructor»+PlayMenu(title: String)
-parseColors(): Color[*]
-updateComboBoxes(): void
-loadScripts(): void
SpectatorMenu
(from View)

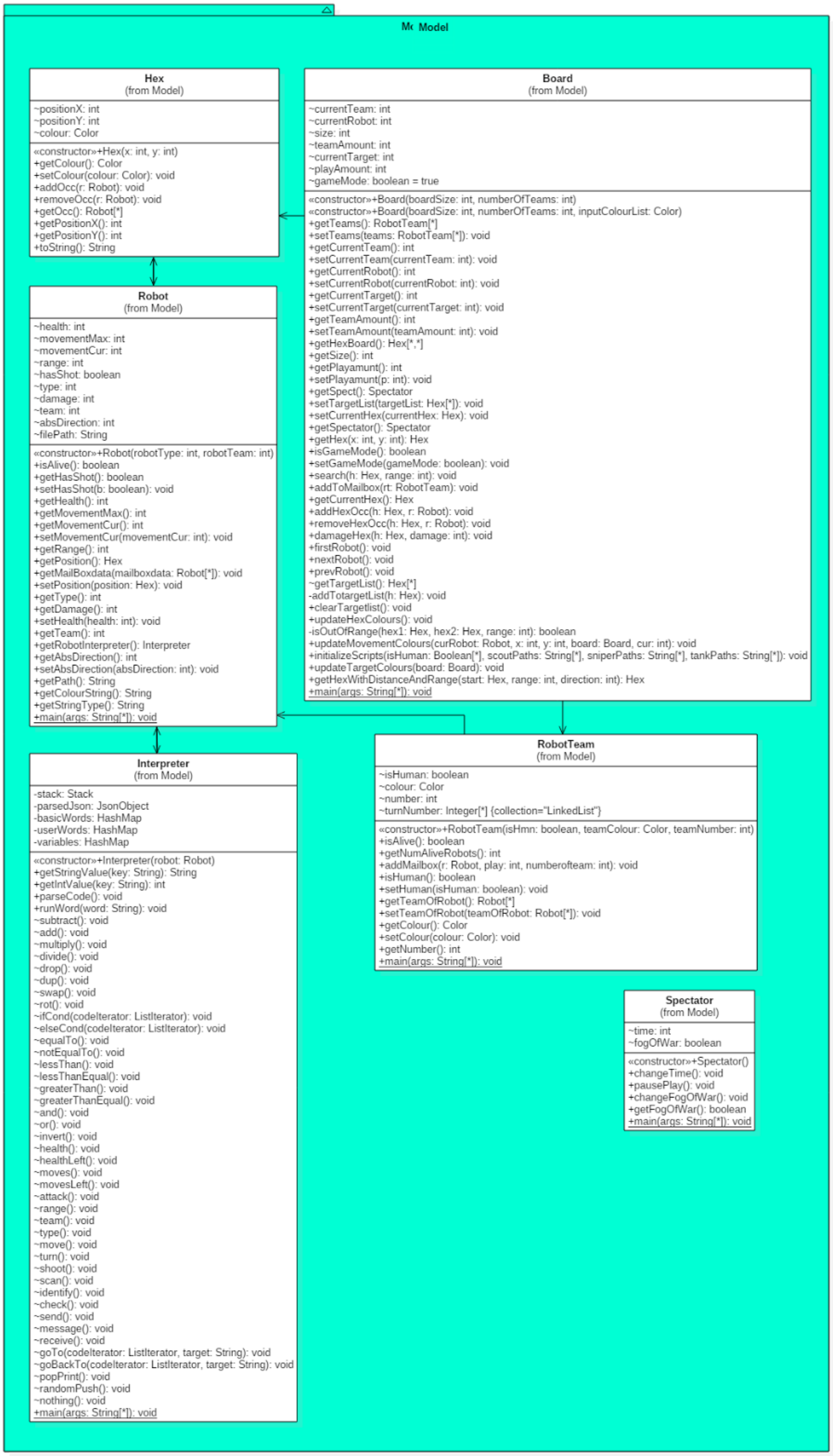
Maintenance

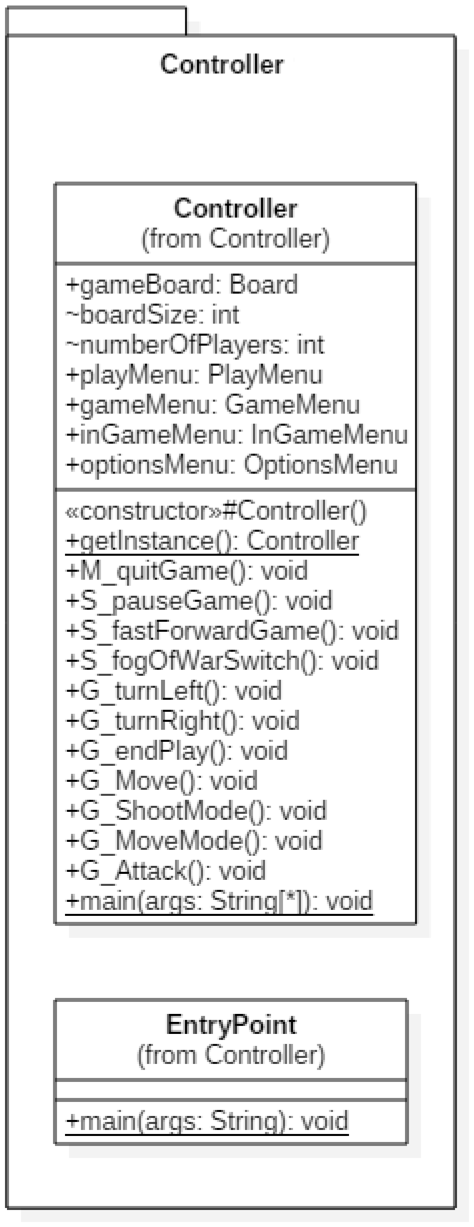
Our system follows the model-view controller architecture as originally designed. This architecture helps encapsulate the data into one area, as well as ensuring that all user-interaction is handled separately from the rest of the system. As well, this architecture keeps all game logic and flow in one 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. The system's EntryPoint class performs as it's name implies. This class launches the system, taking the user to the main menu where they may selection options or play the game.

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 servers 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.


View
GameMenu
(from View)
-serialVersionUID: long = 1L {readOnly}
+main(args: String[*]): void
«constructor»+GameMenu()
OptionsMenu
(from View)
-serialVersionUID: long = 1L {readOnly}
«constructor»+OptionsMenu()
+main(args: String): void
Robotlib
+<<import>>
InGameMenu
(from View)
-leftButton: JButton
-rightButton: JButton
-forfeitButton: JButton
-exitButton: JButton
-actionToggleButton: JButton
-actionButton: JButton
-endPlayButton: JButton
+actionToggle: boolean = true
-serialVersionUID: long = 1L {readOnly}
«constructor»+InGameMenu(b: Board)
+main(args: String[*]): void
OptionsMenuPanel
(from View)
-serialVersionUID: long = 1L {readOnly}
-hostName: String = "gpu0.usask.ca" {readOnly}
-port: int = 20001 {readOnly}
-request: String = "{ \"list-request\" : { \"data\" : \"brief\" }}" {readOnly}
«constructor»+OptionsMenuPanel()
+main(args: String): void
PlayMenu
(from View)
-boardSize: int = 5
-numPlayers: int = 2
-numberOfPlayers: JComboBox
-playerTypes: JComboBox[*]
-playerColors: JComboBox[*]
-scoutScripts: JComboBox[*]
-sniperScripts: JComboBox[*]
-tankScripts: JComboBox[*]
-scoutScriptPaths: String[*]
-sniperScriptPaths: String[*]
-tankScriptPaths: String[*]
-serialVersionUID: long = 1L {readOnly}
+main(args: String[*]): void
«constructor»+PlayMenu(title: String)
-parseColors(): Color[*]
-updateComboBoxes(): void
-loadScripts(): void
InGameMenuPanel
(from View)
-serialVersionUID: long = 1L {readOnly}
-SIDELENGTHFIVE: int = 38 {readOnly}
-SIDELENGTHSEVEN: int = 29 {readOnly}
-boardSize: int
-robotTeams: RobotTeam[*]
-robots: Robot[*]
-currentHexes: Hex[*,*]
-currentTeamLabel: JLabel
-s: int = 0
-t: int = 0
-r: int = 0
-h: int = 0
«constructor»+InGameMenuPanel(size: int, teams: RobotTeam[*])
#paintComponent(g: Graphics): void
+reDraw(board: Board): void
-drawTeamPanel(currentTeam: RobotTeam): void
-drawHexPanel(board: Board): void
-hexColor(x: int, y: int): Color
-drawHex(hex: Hex, g: Graphics2D): void
-drawRobot(toDraw: Robot, g: Graphics2D): void
-createHex(xPos: int, yPos: int): Polygon
-getRobotImage(robot: Robot): BufferedImage
+main(args: String): void
SpectatorMenu
(from View)


Model
Hex
(from Model)
~positionX: int
~positionY: int
~colour: Color
«constructor»+Hex(x: int, y: int)
+getColour(): Color
+setColour(colour: Color): void
+addOcc(r: Robot): void
+removeOcc(r: Robot): void
+getOcc(): Robot[*]
+getPositionX(): int
+getPositionY(): int
+toString(): String
Board
(from Model)
~currentTeam: int
~currentRobot: int
~size: int
~teamAmount: int
~currentTarget: int
~playAmount: int
~gameMode: boolean = true
«constructor»+Board(boardSize: int, numberOfTeams: int)
«constructor»+Board(boardSize: int, numberOfTeams: int, inputColourList: Color)
+getTeams(): RobotTeam[*]
+setTeams(teams: RobotTeam[*]): void
+getCurrentTeam(): int
+setCurrentTeam(currentTeam: int): void
+getCurrentRobot(): int
+setCurrentRobot(currentRobot: int): void
+getCurrentTarget(): int
+setCurrentTarget(currentTarget: int): void
+getTeamAmount(): int
+setTeamAmount(teamAmount: int): void
+getHexBoard(): Hex[*,*]
+getSize(): int
+getPlayamunt(): int
+setPlayamunt(p: int): void
+getSpect(): Spectator
+setTargetList(targetList: Hex[*]): void
+setCurrentHex(currentHex: Hex): void
+getSpectator(): Spectator
+getHex(x: int, y: int): Hex
+isGameMode(): boolean
+setGameMode(gameMode: boolean): void
+search(h: Hex, range: int): void
+addToMailbox(rt: RobotTeam): void
+getCurrentHex(): Hex
+addHexOcc(h: Hex, r: Robot): void
+removeHexOcc(h: Hex, r: Robot): void
+damageHex(h: Hex, damage: int): void
+firstRobot(): void
+nextRobot(): void
+prevRobot(): void
~getTargetList(): Hex[*]
-addTotargetList(h: Hex): void
+clearTargetlist(): void
+updateHexColours(): void
-isOutOfRange(hex1: Hex, hex2: Hex, range: int): boolean
+updateMovementColours(curRobot: Robot, x: int, y: int, board: Board, cur: int): void
+initializeScripts(isHuman: Boolean[*], scoutPaths: String[*], sniperPaths: String[*], tankPaths: String[*]): void
+updateTargetColours(board: Board): void
+getHexWithDistanceAndRange(start: Hex, range: int, direction: int): Hex
+main(args: String[*]): void
Robot
(from Model)
~health: int
~movementMax: int
~movementCur: int
~range: int
~hasShot: boolean
~type: int
~damage: int
~team: int
~absDirection: int
~filePath: String
«constructor»+Robot(robotType: int, robotTeam: int)
+isAlive(): boolean
+getHasShot(): boolean
+setHasShot(b: boolean): void
+getHealth(): int
+getMovementMax(): int
+getMovementCur(): int
+setMovementCur(movementCur: int): void
+getRange(): int
+getPosition(): Hex
+getMailBoxdata(mailboxdata: Robot[*]): void
+setPosition(position: Hex): void
+getType(): int
+getDamage(): int
+setHealth(health: int): void
+getTeam(): int
+getRobotInterpreter(): Interpreter
+getAbsDirection(): int
+setAbsDirection(absDirection: int): void
+getPath(): String
+getColourString(): String
+getStringType(): String
+main(args: String[*]): void
RobotTeam
(from Model)
~isHuman: boolean
~colour: Color
~number: int
~turnNumber: Integer[*] {collection="LinkedList"}
«constructor»+RobotTeam(isHmn: boolean, teamColour: Color, teamNumber: int)
+isAlive(): boolean
+getNumAliveRobots(): int
+addMailbox(r: Robot, play: int, numberofteam: int): void
+isHuman(): boolean
+setHuman(isHuman: boolean): void
+getTeamOfRobot(): Robot[*]
+setTeamOfRobot(teamOfRobot: Robot[*]): void
+getColour(): Color
+setColour(colour: Color): void
+getNumber(): int
+main(args: String[*]): void
Interpreter
(from Model)
-stack: Stack
-parsedJson: JsonObject
-basicWords: HashMap
-userWords: HashMap
-variables: HashMap
«constructor»+Interpreter(robot: Robot)
+getStringValue(key: String): String
+getIntValue(key: String): int
+parseCode(): void
+runWord(word: String): void
~subtract(): void
~add(): void
~multiply(): void
~divide(): void
~drop(): void
~dup(): void
~swap(): void
~rot(): void
~ifCond(codeIterator: ListIterator): void
~elseCond(codeIterator: ListIterator): void
~equalTo(): void
~notEqualTo(): void
~lessThan(): void
~lessThanEqual(): void
~greaterThan(): void
~greaterThanEqual(): void
~and(): void
~or(): void
~invert(): void
~health(): void
~healthLeft(): void
~moves(): void
~movesLeft(): void
~attack(): void
~range(): void
~team(): void
~type(): void
~move(): void
~turn(): void
~shoot(): void
~scan(): void
~identify(): void
~check(): void
~send(): void
~message(): void
~receive(): void
~goTo(codeIterator: ListIterator, target: String): void
~goBackTo(codeIterator: ListIterator, target: String): void
~popPrint(): void
~randomPush(): void
~nothing(): void
+main(args: String[*]): void
Spectator
(from Model)
~time: int
~fogOfWar: boolean
«constructor»+Spectator()
+changeTime(): void
+pausePlay(): void
+changeFogOfWar(): void
+getFogOfWar(): boolean
+main(args: String[*]): void



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.

For the system to support robot scripts, as previously mentioned, .json files are used to store their data and behaviors. This format has the advantage of storing all information in an organized and clear way, though making use of this requires our system to have parsing functionality. To achieve this, we use Google's GSON library, which contains functionality for parsing the file format into objects which our system can handle. This parsing takes place in various places, such as importing robots or using the scripts in-game. Details on this library may be found at: https://sites.google.com/site/gson/gson-user-guide.

The system may require maintenance over time due to Java updates or other reasons. All of the above mentioned classes are the most likely to break over time due to their intricacy. For finding bugs that aren't visual, the controller would be a good place to begin hunting as it initializes the rest of the system. From there, the board would be a good place to check next as it handles all the data of the model. Mishandling of said data would be a likely cause of a non-visual bug. For visual bugs, the view would be the obvious place to look. In-game errors would likely be found in the in-game menu or its respective panel, while bugs in other menus may be found in their classes.

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. The Robot Team have also added the mailbox system, the mailbox stores the location of all the robot each of the robot on the team sees, and shares it with each of the robot on 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. The firstRobot method will set the currentHex to the first robot in the target list. The clearTargetList method will remove all hex's from the target list. The addToTargetList method will take in a hex to append to the target list. The updateHexColours method updates all of the hex's colours to either gray or white based on the range of the current robot team's robots. The isOutOfRange method takes in two hexs and a range value and determines if they are out of range. The updateMovementColours method takes in info for a robot and then sets the hex that the robot is facing to the colour green. The updateTargetColours method takes the targetList of the board and makes every hex in that list yellow. It also makes the currently selected hex red.

- **Interpreter:**

    The Interpreter class also met dramatic changes. The separate word classes ended up not being implemented. The basic structure was changed to have 3 hash maps (for the base words, user defined word, and variables), a stack for the actual information processing. The forth code is initially parsed, categorising all words as either userWords or variables. All the code used in the definition of the userWords are saved as a string, and will be parsed further when running the word. A play is started by calling runWord("play").

During a runWord, the forth code for that particular word is iterated through. All basic words are mapped to their proper function in Java, and each userDefined word calls a new instance of runWord for that word. The allows for resolution of functions within other functions. For all variables found during a runWord, the variable's name is pushed to the stack. The forth interpreter is mostly working, however the loop words are currently not implemented as well as the "identify!" word.

- **View:**
  - **In-Game Menu:**

    The in-game menu went through a variety of changes throughout construction. The most notable change is the splitting of the menu into two separate classes: the frame and the panel. While many of the fields in the original design still exist, albeit split among the two new classes, additional fields and private methods were implemented to assist with the functionality. For instance, the panel class contains many functions to support drawing the board, such as drawHex(), drawRobot(), and other similar methods.

    These are all required to support the menu's reDraw() function that was specified in the original design. The frame class contains the listeners for each button, as well as the panels for displaying the current team or the currently targeted hex. While the implementation did evolve compared to the design, it's original purpose stayed true and it's only public function (reDraw) still exists as the primary function.

  - **Game Menu:**

    The game menu remained almost exactly the same as in our design, with no real changes being implemented other than visual differences. This was expected, as the only purpose of this menu is to serve as a hub for the rest of the system.
  - **Spectator Menu:**

    Unfortunately, the spectator menu never made it into construction, and had to be cut. However, had it been implemented, it would likely have remained similar to it's original design, as the buttons were intended to be the only real functionality of it anyways.
  - **Options Menu:**

    The options menu also remained quite similar to the original design, though there are differences to be noted. Most notably, the options menu was split into two classes: the frame and panel. The panel handles the displaying of the importable robots, which are downloaded and parsed from the robot librarian server. Each robot is displayed along with an import button which saves the script to the system when pressed. The frame contains the overall layout for displaying the panel, as well as the back button,

- **Play Menu:**

  The play menu had minor changes made to it during construction. The most upfront being that combo boxes were added to support selection for robot scripts for each player, along with the combo boxes for team types and colors already specified in design. Other than that, along with listeners for required buttons, nothing major changed from design and it's original purpose stayed true: to create the board based on the user's selection options.

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.

1. On start up of the system in the EntryPoint.class there is a possibility of two instances of the system occurring. Both instances close when the exit button is pressed within the system and when the window is closed. If the play button or options button is selected only the one instance will go to the next menu while the other stays on the GameMenu frame.

2. On startup the system loads in multiple windows before the GameMenu appears. While this is not so much a user effecting error it is a graphical bug and could be optimized so that the system cleanly presents the first window to be seen without loading the other windows before it.

3. When the user selects the Play menu button and is brought to the play menu if they decide to exit by pressing the back button a new instance of the Main menu is created along with the current window. This bug is able to duplicate over and over again by continuously going to the play menu and backing out of the play menu.

4. When starting a game the robots appear to be in their starting locations but when another robot tries to target another robot which hasn't moved yet the shooting robot is unable to target the robot that hasn't moved yet.

5. When starting the game the first robot does not have the icon to see which way it is facing. Instead the directional button has to be pressed in order for the green hexagon that signals the direction the robot is facing to appear.