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. 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.

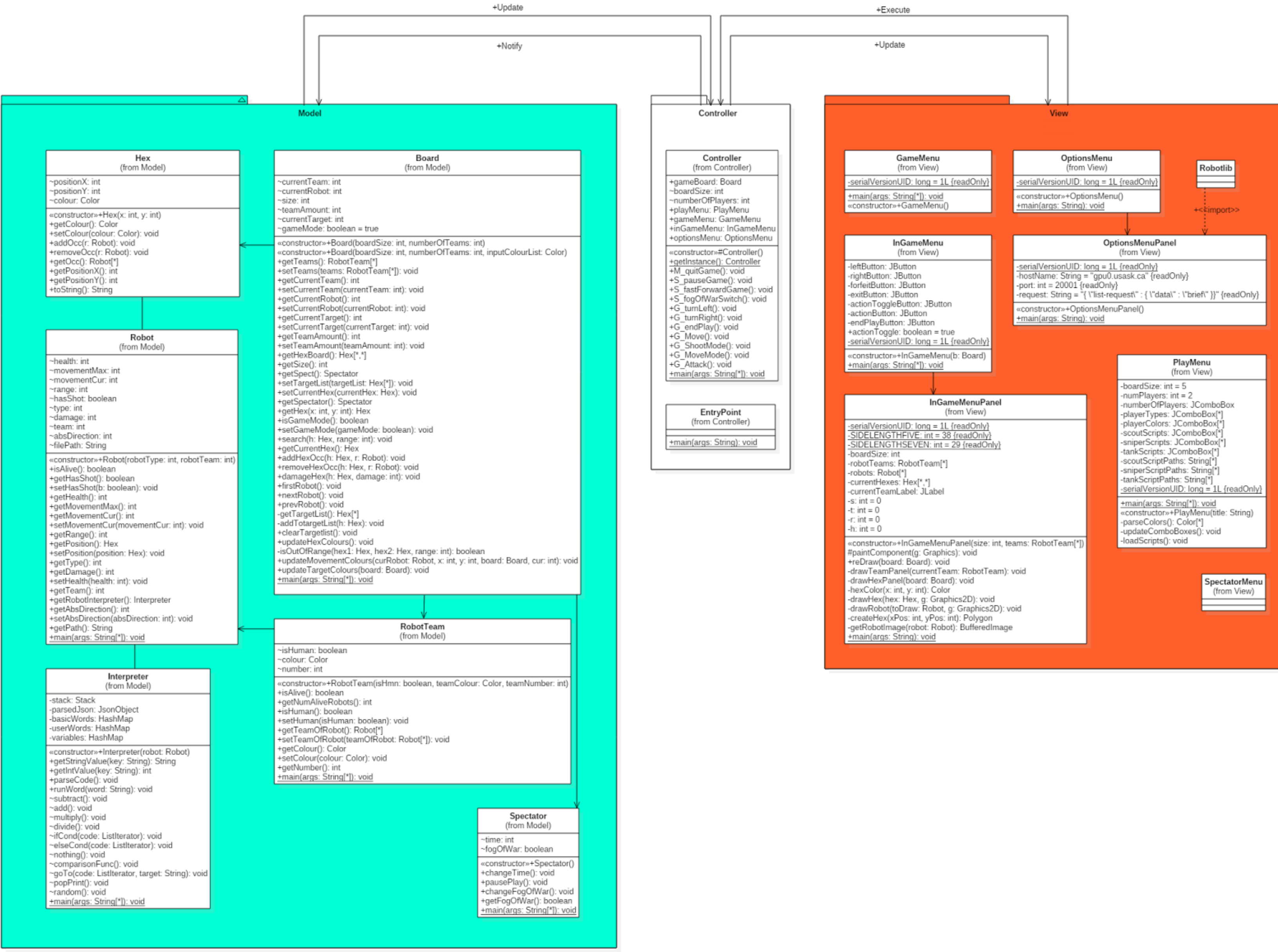
Upon entry, the options menu will present the user with a list of importable robots, as well as various stats of theirs. Users may import their selected robots by pressing the “Import” button on them. This will save the robot’s script to the program’s local directory which can be found in the folder labelled “Model”. These scripts may be accessed through the play menu and assigned to AI player’s robots when the game is created.

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).

Provide information on robots/their format/importing/exporting:

Discuss limitations and restrictions of our system

Maintenance Document


+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
Board
(from Model)
~currentTeam: int
~currentRobot: int
~size: int
~teamAmount: int
~currentTarget: 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
+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
+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
+updateTargetColours(board: Board): void
+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
+setPosition(position: Hex): void
+getType(): int
+getDamage(): int
+setHealth(health: int): void
+getTeam(): int
+getRobotInterpreter(): Interpreter
+getAbsDirection(): int
+setAbsDirection(absDirection: int): void
+getPath(): String
+main(args: String[*]): void
RobotTeam
(from Model)
~isHuman: boolean
~colour: Color
~number: int
«constructor»+RobotTeam(isHmn: boolean, teamColour: Color, teamNumber: int)
+isAlive(): boolean
+getNumAliveRobots(): int
+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
~ifCond(code: ListIterator): void
~elseCond(code: ListIterator): void
~nothing(): void
~comparisonFunc(): void
~goTo(code: ListIterator, target: String): void
~popPrint(): void
~random(): 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
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)

Architecture of our system (Including new UML)

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

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

Designs that were altered/removed during construction

Remaining bugs