Class: CMPT 370

Professor: Christopher Dutchyn

Tutorial Leader: Jonathon Baxter

ROBOSPORT 370
DESIGN DOCUMENT

Group B1: Adam Ronellenfitsch, Dylan Prefontaine, Evan Snook, Matthew Frisky, and Wynston Ramsay

Table of Contents

Architecture

Model-View-Controller is the architecture chosen for our system. Using Model-View-Controller allows us to separate data and user interface so that data handling is not affected by the user interface. This data can easily be changed and moved without the need to change the interface using the controller. The model represents the knowledge or data contained in our system. The views are the visual representation of the system, mainly the graphical interface. The controller is the link between the user and the system, it works with the model and views to control the user input.

The main reason we decided to use Model-View-Controller for our system is that it allows you to easily separate the different components of the system. This increases the ease of re-use for classes, particularly in the model aspect of the system. It also allows for easier maintenance and testing as each component in the system can be separated and independently tested or altered without affecting the overall system. It is a great way to keep our code functional and maintainable as having everything separate allows you to see exactly what each piece of the system is supposed to be doing.

Another reason we chose to use Model-View-Controller is that it is the architecture that we as a team are most familiar with and felt we would be the most comfortable using in our system. This plays a major role in our decision as being more familiar with Model-View-Controller. It lets us implement our system more easily and effectively than if we were using another unfamiliar architecture. We also feel as though Model-View-Controller is very fitting for our system as it is highly UI based and enables us to easily handle the interaction between the UI with the data through the controller.

Definitions

Global Direction

Global Direction refers to the direction on the board where 0 is always to the right, 1 is to the bottom right and continues around the board to 5 being the top right.

Robot Librarian

The Robot Librarian is a system for storing and providing access to a collection of robot script for the project. The purpose of this system is to store robot scrips, including metadata, to facilitate viewing of the collection by various criteria, to enable download of a robot script for execution in a match and to accept updates of statistics based on completion of a match.

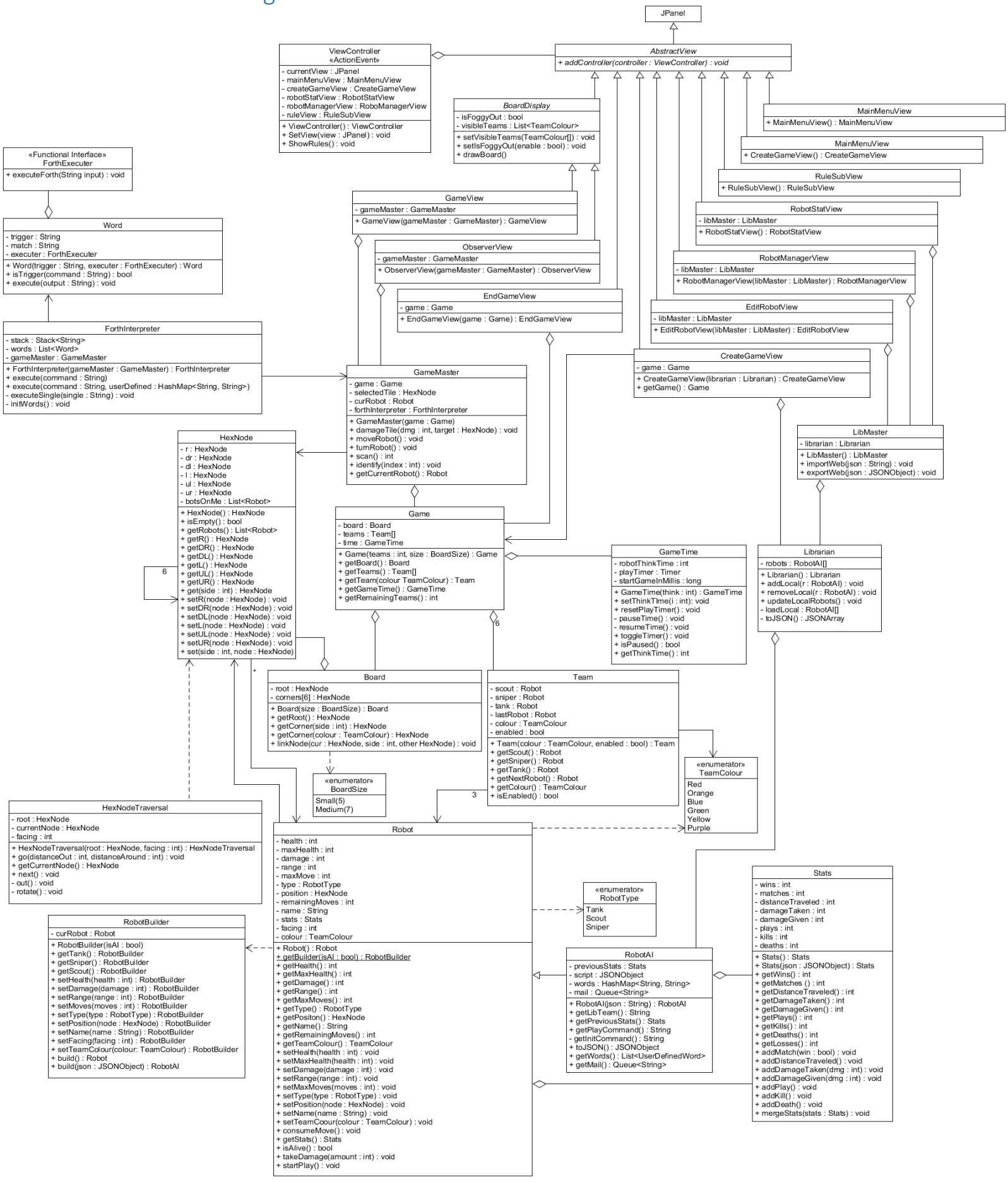
JSON

We will be using an external library to handle parsing the JSON objects that we will receive and send to the Robot Librarian. The library we will be using is the JSON.simple library available through maven at com.googlecode.json-simple.

ForthExecutor

ForthExecutor is a functional interface that a Word will use to execute a command.

UML Class Diagram



This UML diagram outlines the overall structure of our system and their relationships. Following the UML diagram is a description of every class. In each description, we discuss their responsibilities and collaborators in detail with the use of a Class Responsibility Collaborator (CRC) card. There are also descriptions of each attribute and method for those classes. Note that accessor and mutator functions do not have a description given that they are trivial; accessors return the value of an attribute and mutators change the value of an attribute. We also make use of three enumerator classes: TeamColour to denote a team's colour (RED, BLUE, YELLOW etc.), RobotType to denote the type of robot (SCOUT, SNIPER, TANK) and BoardSize to denote the size of the board (SMALL for a board size of 5 and MEDIUM for a board size of 7).

Class Writeups

AbstractView

CRC

AbstractView	
Responsibilities Act as an outline for all the views	Collaborators extends JPanel All the view will extend this

Description / Overview

Gives each view a basic interface to allow the controller to listen to the input within the view.

Method Overview

public abstract void addController(ViewController controller)

Method Writeups

public abstract void addController(ViewController controller)

Adds the controller as a listener to this action.

Board

CRC

Board	
Responsibilities To maintain a model of the board.	Collaborators HexNode and BoardSize help to accomplish the board's task of representing a hexagonal board of size 5 or 7.

Description / Overview

The Board's main purpose is to represent a model of an abstract data type that is made up of HexNodes. In our MVC architecture, the board is a model so it is only used to store data. A board is used

to hold a series of HexNodes to form a graph to represent the game board. There are two size options for the length of each side that are determined by the enumerator BoardSize; one size will be a length of 5 hexagon tiles and the other, a length of 7. All the tiles will be linked together with the known size to form a Board model.

Instance Variables

ROOT

Data Type: HexNode<T>
Contains a private reference to the center HexNode.

CORNERS

Data Type: HexNode<T>[6]
A private array of HexNode's to represent the six corners of the Board that will be used to determine Team starting positions.

Method Overview

public Board(BoardSize size)
public HexNode getRoot()
public HexNode getCorner(int direction)
public void linkNode(HexNode cur, int side, HexNode other)

Method Writeups

public Board(BoardSize size)

This is the constructor for the Board, it takes a BoardSize as input take is determined by the user. The constructor will then create a root HexNode. It uses this reference to iteratively create HexNodes that will expand from the root until the size condition is satisfied. Once the Board is created, it identifies and stores the corner HexNodes into an array of six.

public void linkNode(HexNode cur, int side, HexNode other)

The linkNode method is used when initializing the Board to link together HexNodes. As arguments, it receives two HexNodes, "cur" and "other", as well as an integer called side. "Cur" represents the current node that is being linked with the "other" HexNode. The Board constructor repeatedly links nodes together until the board is complete.

BoardDisplay

CRC

BoardDisplay	
Responsibilities Create a visual for the Board model and display it to the user.	Collaborators BoardDisplay requires a GameView to be active so that the board can be rendered. It also requires there to be a complete Board model. It inherits from the AbstractView class.

Description / Overview

BoardDisplay is a class that simply displays the model of the Board in the view. BoardDisplay is the parent of GameView and represents one of our views in our model-view-controller architecture. The board is to draw all the elements of the game state and redraws when told to. The board state can change from the game being played or which an observer changing the fog visibility settings. While observing, they can either see the entire board, or select a team's colour to view the game from their perspective.

Instance Variables

isFoggyOut

Data Type: *boolean*

If you are observing a game, you can set this to false to show the entire board. If set to true it will display the teams point of view based on *visibleTeams*.

visibleTeams

Data Type: *List<TeamColour>*

A list of teams whose perspective is visible in the fog.

Method Overview

public abstract void setVisibleTeams(TeamColour … colours)
public void setFoggyOut(bool enableFogMachine)
public abstract void drawBoard()

Method Writeups

public abstract void setVisibleTeams(TeamColour … colours)

Set the list of teams that are visible in the fog.

public abstract void drawBoard()

Draws a hexagonal board to be displayed in the GameView. It draws the entire game state; which includes teams, robots, etc. If the user is in observer mode, then it will draw based on whether fog of war is selected and which teams are selected.

CreateGameView

CRC

CreateGameView	
Responsibilities Display the CreateGame screen to the user.	Collaborators Relies on the ViewController to be activated and deactivated. Extends AbstractView like all views.

Description / Overview

The CreateGameView is a view that is to show a Create Game screen to the user. It is activated by the ViewController when the Create Game button in the MainMenuView is triggered. It presents an interface that helps the user determine how they want to set up their game. It includes the following

things: changing the number of teams, who controls each team, the board size, robot think time, and which robots are on each team.

Instance Variables

GAME

Data Type: Game

A reference to the model of the game so that ViewController can access it.

Method Overview

public CreateGameView()
public Game getGame()

Method Writeups

public CreateGameView()

This is the constructor for the CreateGameView class. It creates the Create Game interface and displays it to the user. It contains and event to handle the "Start" button which will notify the ViewController. It also creates UI elements for the user to interact with to create settings for the game they are creating.

EditRobotView

CRC

EditRobotView	
Responsibilities Display the interface to edit Robots	Collaborators AbstractView to control the users input on the Interface. LibMaster to access robots and their code.

Description / Overview

A User Interface that contains a list of robots. The selected robot's information is displayed in the boxes which can be changed. There is a selection box with the list of robots to be edited and text boxes to edit the robots name, team, and code. There are four buttons available to save changes, cancel changes, create a new robot, or go back to the main menu.

Instance Variables

LIBMASTER

Data Type: LibMaster

Contains a private reference to the libMaster that contains the librarian to help access the robots and their code

Method Overview

public EditRobotView(LibMaster libMaster)

Method Writeups

public EditRobotView(LibMaster libMaster)

Constructs a new EditRobotView using a LibMaster that will be used to get and set the robot's information.

EndGameView

CRC

EndGameView	
Responsibilities Display the end game screen with information about the match	Collaborators AbstractView to control the users input on the Interface. Game to access information from the match to be displayed

Description / Overview

The user interface that appears once a game has ended. It contains match statistics for individual teams as well as 3 buttons. The New Game Button will bring the user to the Create Game user interface. The Main Menu Button brings the user to the Main Menu, and the Quit Button will terminate the game.

Instance Variables

GAME

Data Type: Game

Contains a private reference to a game that will be used to fetch statistics from.

Method Overview

public EndGameView(game : Game)

Method Writeups

public EndGameView(game : Game)

Constructs a EndGameView screen using a Game to fill in fields that need some statistics from the game.

ForthInterpreter

CRC

ForthInterpreter	
Responsibilities Take input to command Robots in the game	Collaborators GameMaster to modify and get information about the game

Description / Overview

The ForthInterpreter will be taking in input from Robots as Strings and executing the commands on its Stack using its Words while gathering its information from the GameMaster.

Instance Variables

STACK

Data Type: *Stack<String>*
The Stack that this interpreter will work on.

WORDS

Data Type: *List<Word>*
A list of built-in Forth words to be interpreted.

GAMEMASTER

Data Type: GameMaster
The GameMaster that this interpreter will be using to command robots.

Method Overview

public ForthInterpreter(GameMaster gameMaster)
public void execute(String fullCommand)
public void execute(String fullCommand, HashMap<String, String> userDefined)
public executeSingle(String singleCommand)
public void initWords()

Method Writeups

public ForthInterpreter(GameMaster gameMaster)
Create a forth interpreter with a reference to the current GameMaster so it can modify the game.

Public void execute(String fullCommand)
Execute a string that contains multiple commands.

public void execute(String fullCommand, HashMap<String, String> userDefinedWord)
Execute a string that contains multiple commands and may contain a user defined word.

public void executeSingle(String singleCommand)
Execute a single command.

public void initWords()
Initialize all the pre-defined Forth words

Game

CRC

Game	
Responsibilities Holds all the data needed to play the game.	Collaborators Holds the Team, GameTime, Board, and GameMaster.

Description / Overview

The Game holds three things: the game board, an array of six teams, and the GameTime attribute. Game can provide the GameMaster with a reference to the game board being used as well as the list of teams, a specific team based on a colour, the GameTime object, and the number of remaining teams.

Instance Variables

BOARD

Data Type: Board
Board is a reference to the Board object that is currently being used.

TEAMS

Data Type: Team*[]*
Teams is an array of six Teams in the order Red, Orange, Blue, Green, Yellow, Purple

TIME

Data Type: GameTime
Time holds the GameTime object relative to this game.

Method Overview

public Game(int teamCount, BoardSize size)
public Board getBoard()
public Team[] getTeams()
public Team getTeam(TeamColour colour)
public GameTime getGameTime()
public int getRemainingTeams()

Method Writeups

public Game(int teamCount, BoardSize size)
Creates the GameTime, Board, and six teams for the game and enables or disables them accordingly.

public Team getTeam(TeamColour colour)
This method will return the team with the corresponding colour.

public int getRemainingTeams()
This method returns the number of teams that are enabled.

GameMaster

CRC

GameMaster	
Responsibilities Hold a game board and control the robots	Collaborators Sends commands to the ForthInterpreter Controls the state of the Game

Description / Overview

GameMaster is the controller of the overall game mechanics such as: controlling robots, (turning, shooting, firing, scanning and identifying) and communicates with the ForthInterpreter to execute commands for the RobotAI.

Instance Variables

GAME

Data Type: Game

Holds a reference to the current game state.

SELECTEDTILE

Data Type: HexNode<T>

The tile that the human player has selected.

CURROBOT

Data Type: Robot

The robot who is currently making their play.

FORTHINTERPRETER

Data Type: ForthInterpreter

The forth interpreter to control the RobotAI in the current Game.

Method Overview

public GameMaster(Game game)
public void damageTile(int damage, HexNode<T> node)
public void moveRobot()
public void turnRobot()
public int scan()
public void identify(int index)
public void getCurrentRobot()

Method Writeups

public GameMaster(Game game)

The constructor for the GameMaster takes in the Game it will be controlling.

public void damageTile(int damage, HexNode<T> node)

DamageTile takes in an integer for how much damage to do and a HexNode for which tile to damage. Damage will be the robots damage and HexNode will be the selectedTile.

public void moveRobot()

MoveRobot moves the robot one tile in the direction that he is facing if it can

public Team getCurrentTeam()

GetCurrentTeam will ask the curRobot for his colour attribute.

public void turnRobot()

Rotate the robot one unit clockwise.

public int scan()

Scan returns the number of robots currently in view of the current robot.

public void identify(int index)

Identify will take will push TeamColour, how far the robot is away and the direction the robot is in, as well as the robots remaining health.

GameTime

CRC

GameTime	
Responsibilities Keep track of the current game time. End players turn if they run out of time.	Collaborators Game holds GameTime

Description / Overview

The GameTime class is a part of the Model in our architecture and is a class that simply keeps track of all aspects of time for a game. GameTime controls how long a robot takes to complete a turn, as well as how long a human player has to complete a turn. GameTime can change the think time for a robot, pause the timer, reset the timer, and keep track of the overall time for the game.

Instance Variables

ROBOTTHINKTIME

Data Type: *Integer*

How long a robot takes to act in seconds.

PLAYTIMER

Data Type: *Timer*

The timer that will end a player's turn if they run out of time.

STARTGAMEINMILLIS

Data Type: *Long*

Tags the beginning of a game with the currentTimeInMillis to be used to calculate the total game time when the game ends.

Method Overview

public GameTime GameTime(int think)
public void setThinkTime(int i)
public void resetPlayTimer()
public void toggleTimer()
public bool isPaused()
public int getThinkTime()
private void pauseTime()
private void resumeTime()

Method Writeups

public GameTime GameTime(int think)

Sets the robot think time and instantiates it's other fields.

public void setThinkTime(int i)

This method sets the think time for the robot to some integer I.

public void resetPlayTimer()

This method resets the play timer back to 0.

public void toggleTimer()

This method switches toggles the timer being paused or running. If the timer is paused it will resume and if it is running it will pause.

public bool isPaused()

This method checks whether the timer is paused or running.

public int getThinkTime()

This method returns the current think time for the robot.

private void pauseTime()

This method pauses the timer, allowing it to be resumed at the same number at any time.

public void resumeTime()

This method resumes the timer from the same time it was paused.

GameView

CRC

GameView	
Responsibilities GameView is responsible for displaying the game screen to the user.	Collaborators GameView collaborates with GameMaster and inherits from BoardDisplay to display the game to the user.

Description / Overview

GameView in our model-view-controller architecture is to represent a view that will display the game state to the user. The game to be displayed contains the hexagon shaped board which GameView will inherit from the BoardDisplay class. The GameView ill also contain buttons to shoot, move, end turn, exit match and to view the rules. At the top of the view it will indicate which team's turn it is. The view exists only when a game is created and the game still exists; once a game is finished the view will be replaced by the end game screen. The view should automatically be redrawn when a change in the game has been made to display the most recent version of the model.

Instance Variables

GAMEMASTER

Data Type: GameMaster

A reference to the controller of the current game which is called.

Method Overview

public GameView(GameMaster gameMaster)

Method Writeups

public GameView(GameMaster gameMaster)

GameView is the constructor for the class the draws the initial game in a frame to be shown to the user whether they are a player or an observer. It listens events triggered via buttons and tile selection in the view; and communicates with the game controller by having a reference to it when created. The controller (gameMaster) will handle all events when fired and GameView will redraw whenever changes to the Game Model are made.

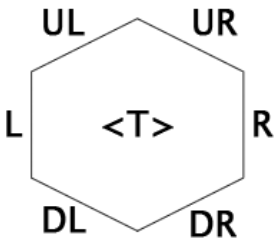
HexNode<T>

CRC

HexNode	
Responsibilities Holds data and has six references to other nodes.	Collaborators HexNode Generic type T

Description / Overview

HexNode is a node that depicts a point on our board which will be displayed as a graph. Each HexNode has a reference to its neighboring nodes and a value of type *T* which is generic to the class. The HexNode is visualized as a Pointy Orientation Hexagon (see below).



Instance Variables

R

Data Type: *HexNode<T>*
A reference to the node to the right of this node.

DR

Data Type: *HexNode<T>*
A reference to the node to the bottom-right of this node.

DL

Data Type: *HexNode<T>*
A reference to the node to the bottom-left of this node.

L

Data Type: *HexNode<T>*
A reference to the node to the left of this node.

UL

Data Type: *HexNode<T>*
A reference to the node to the top-left of this node.

UR

Data Type: *HexNode<T>*
A reference to the node to the top-right of this node.

VALUE

Data Type: *T*

The value of this node.

Method Overview

public HexNode<T> ()
public void setR()
public void setDR()
public void setDL()
public void setL()
public void setUL()
public void setUR()
public void set(int side, HexNode<T> node)
public HexNode<T> getR()
public HexNode<T> getDR()
public HexNode<T> getDL()
public HexNode<T> getL()
public HexNode<T> getUL()
public HexNode<T> getUR()
public HexNode<T> get(int side)
public boolean isEmpty()
public T getValue()

Method Writeups

public void set(int side, HexNode<T> node)

Set the side of this node to a specified node. (see Global Direction)

public HexNode<T> get(int side)

Get a node on a specified side of this node. (see Global Direction)

public boolean isEmpty()

Is the value of this node null

HexNodeTraversal

CRC

HexNodeTraversal	
Responsibilities Traverses through a structure of HexNodes connected by links to one another without modifying their data.	Collaborators HexNode

Description / Overview

HexNodeTraversal is used to iterate through the graph-like structure of the HexNodes. It keeps track of the root HexNode (where the traverse began), the current HexNode that is being traversed, and which linked node it is facing (see Global Direction). It provides movement through the Hexagon board by moving from HexNode to HexNode, layer by layer while preventing the traverse from leaving the boundaries. It begins from the center and starts from the global position of 0 and works around that layer, then moves to the next layer until the entire graph has been traversed. It is an inside-out approach at traversing the Board. First layer will just be the middle so it will have the node 0. The second layer will have 0-5, the third layer will have 0-11, the fourth 0-17, the fifth 0-23, and in the case the board size is 7 the sixth layer will have 0-35.

Instance Variables

ROOT

Data Type: HexNode<T>

The root contains reference a HexNode that is the starting point of the traversal.

CURRENTNODE

Data Type: HexNode<T>

CurrentNode is the CurrentNode that the traversal is on.

FACING

Data Type: *Integer*

Facing is an integer that contains the direction to start the traversal according to global direction.

Method Overview

public HexNodeTraversal(HexNode root, int facing)
public void go(int distanceOut, int distanceAround)
public HexNode getCurrentNode();
public void next();
private void out();
private void rotate();

Method Writeups

public HexNodeTraversal(HexNode root, int facing)

This is the constructor for the HexNodeTraversal class. It simply creates a HexNodeTraversal object, initially sets the root HexNode and the direction to star the traversal.

public void go(int distanceOut, int distanceAround)

Go is a method that traverses to a hexnode. Distance out being which layer it is on and distance around being the index of the node on that layer.

public void next();

Moves the current node to the next node in the traversal.

private void out();

Moves the currentNode one layer outwards from the current position.

private void rotate();

Turns the facing direction one more to the right (Adds 1). It needs to check that it is facing a valid HexNode and that it does not rotate more than 6 times (0-5), it should reset back to zero in this instance.

LibMaster

CRC

LibMaster	
Responsibilities Importing and exporting to and from the moodle library	Collaborators LibMaster collaborates with the edit robots and the manage robots views. LibMaster also talks with a Librarian object to help manage the robots.

Description / Overview

LibMaster provides a way to import and export robots to and from the website. It uses the Librarian to hold all the robots that will be able to be used in a game.

Instance Variables

LIBRARIAN

Data Type: *Librarian*

The librarian attribute holds a member of the Librarian class.

Method Overview

public void import(String json)
public void export(String json)

Method Writeups

public void import(String json)

Import will take a robot from the website and add it to the list in the librarian.

public void export(String json)

Export will take a robot from our system and add it to the website

Librarian

CRC

Librarian	
Responsibilities to manage robots that are stored within the system in JSON	Collaborators RobotAI is used access different pieces of data about robots. LibMaster uses Librarian to access robots so that it can upload and download them.

Description / Overview

The Librarian has an array of all the robots in the system and has procedures to add, remove, update, and load in the robots to the array. It can also create a JSON file for the robots to have their information saved.

Instance Variables

ROBOTS

Data Type: *RobotAI[]*

An array of robots to be accessed.

Method Overview

public Librarian()
public void addLocal(RobotAI robot)
public void removelLocal(RobotAI robot)
public void updateLocalRobots()
private RobotAI[] loadLocal()
private JSONArray toJSON()

Method Writeups

public Libarian()

This Method constructs a New Librarian

public void addLocal(RobotAI robot)

This method adds a robot to the array of robots

public void removelLocal(RobotAI robot)

This method removes a robot from the list of robots

public void updateLocalRobots()

This method updates the robots saved on the local machine

private RobotAI[] loadLocal()

This method loads a robot from the local machine

public JSONArray toJSONArray()

This method turns all the robot's information into a JSON formatted file

MainMenuView

CRC

MainMenuView	
Responsibilities Provide an interface between the human and the system.	Collaborator ViewController to notify it when to change views

Description / Overview

Main menu view is the main hub for all views. From here you have five buttons: create game, rules, stats, robot manager, and quit. The create game, stats, robot manager buttons will send a message to the view controller telling it to change to each respective view. The rules button will open a rules window, displaying the rules. The quit button will shut down the program.

Method Overview

public MainMenuView()

Method Writeups

public MainMenuView()

The constructor for the main menu view will set up all the five buttons and set the title text.

ObserverView

CRC

ObserverView	
Responsibilities To display the Observer interface when a game is created and the user is an observer.	Collaborators It inherits from BoardDisplay and AbstractView. It is activated and deactivated through the ViewController. The game provides the information necessary to perform its task.

Description / Overview

The ObserverView class in our model-view-controller architecture is a view. It is activated by the ViewController when a game is created and the user is an Observer. Its purpose is to create a board and display that in the interface since both involve a game being played. The Observer view is different from the GameView since it does not allow the user to interact with the board directly. Instead they are given viewing options; things like changing the robot think time which speeds up or slows down the game. It has buttons to display rules of the game or to exit the match. Fog of war checkboxes are available as well so the user can choose which teams perspectives they can watch the game from. It is also redrawn whenever changes are made to the model or if the fog of war is changed. Once a game is finished the ViewController will deactivate the ObserverView.

Instance Variables

GAMEMASTER

Data Type: GameMaster

The gameMaster variable has the purpose of creating a link between ObserverView and the GameMaster Controller. It is useful for when events are triggered in the view, so it can then send the events to the GameMaster to be handled.

Method Overview

public ObserverView(GameMaster gameMaster)

Method Writeups

public ObserverView(GameMaster gameMaster)

Create the ObserverView to display the options that an Observer can modify.

RoboManagerView

CRC

RoboManagerView	
Responsibilities RoboManagerView is responsible for displaying a list of robots and importing/exporting robots.	Collaborators RoboManagerView collaborates with AbstractView and LibMaster.

Description / Overview

RoboManagerView is part of the View in our architecture and it displays two lists of robots, one list of local robots, and one list of online robots. It also displays five buttons, which allow you to edit the local robots, import/export new robots, and delete existing robots. It also has a back button which takes you back to the previous view.

Instance Variables

LIBMASTER

Data Type: *LibMaster*

Contains a private reference to LibMaster that will be used to fetch the lists of robots from.

Method Overview

public RoboManagerView(LibMaster libMaster*)*

Method Writeups

public RoboManagerView(LibMaster libMaster)

This method creates a frame which displays five buttons (edit, import, export, delete, and back) and two lists of robots.

Robot

CRC

Robot	
Responsibilities Encompasses the condition of a robot and its Stats throughout a game.	Collaborators RobotBuilder to build a robot Stats to hold this robots statistics

Description / Overview

The Robot class contains the stats of a specific robot in a game. A robot must be one of the three types being a Scout, Sniper or Tank, each with their own unique properties. This class is responsible to keep track of and control the state of this robot and does not affect any other models directly as this is done through GameMaster.

Instance Variables

MAXHEALTH

Data Type: *Integer*

The amount of health this unit will spawn with at the start of the game.

HEALTH

Data Type: *Integer*

The amount of health this unit has left.

DAMAGE

Data Type: *Integer*

The amount of damage this Robot deals when they attack a tile.

RANGE

Data Type: *Integer*

The maximum radius in which this robot can see and shoot.

MAXMOVE

Data Type: *Integer*

The maximum amount of times this Robot can move in a play.

REMAININGMOVES

Data Type: *Integer*

The remaining moves this Robot has for this play.

TYPE

Data Type: *RobotType*

The type of robot this unit is (Scout, Sniper or Tank)

POSITION

Data Type: HexNode<T>

The current node that this Robot resides on.

NAME

Data Type: *String*

A name for this robot. The name is displayed below the robot on the game board.

STATS

Data Type: Stats

A collection of this robot's stats specific to this game. (Kills, Damage Dealt etc.)

FACING

Data Type: *Integer*

The Global Direction that this robot is facing on the board.

TEAMCOLOUR

Data Type: *TeamColour*

The colour of the team that this robot belongs to.

Method Overview

public Robot()
public static RobotBuilder getBuilder(boolean isAI)
public int getHealth()
public int getMaxHealth()
public int getDamage()
public int getRange()
public int getMaxMoves()
public int getRemainingMoves()
public RobotType getType()
public HexNode getPosition()()
public TeamColour getTeamColour()()
public String getName()
public void setHealth(int health)
public void setDamage(int damage)
public void setRange(int range)
public void setMoves(int moves)
public void setType(RobotType type)
public void setPosition(HexNode node)
public void setName(String name)
public void setTeamColour(TeamColour colour)
public Stats getStats()
public void consumeMove()
public boolean isAlive()
public void takeDamage(int amount)
public void startPlay()

Method Writeups

public static RobotBuidler getBuilder(boolean isAI)

Gives us an instance of RobotBuilder which is a helper class to create robots.

public void consumeMove()

Decrements the amount of moves this Robot has in its turn.

public boolean isAlive()

Checks if this robot is still alive.

public void takeDamage (int amount)

Reduces this Robots health by amount and updates the stats of the robot with the amount of damage it took and if it died from the damage.

public void startPlay ()

Update the Robot's stats by incrementing the play count and reset the robot's remaining moves.

RobotAI

CRC

RobotAI	
Responsibilities Holds the information about a Robot that will be controlled by a script.	Collaborators Stats for its previous statistics Librarian stores this robot on the local machine

Description / Overview

RobotAI is an extension of the Robot class that contains extra information about this robot including the Team that coded it, its previous stats and the script that created it.

Instance Variables

PRESVIOUSSTATS

Data Type: Stats

A combination of this Robots stats from all the matches it has played.

SCRIPT

Data Type: JSONObject

The JSON script that instantiates the robot.

WORDS

Data Type: *HashMap<String, String>*

A list of words that this robots script has declared for itself to use.

MAIL

Data Type: *Queue<String>*

A mailbox that can hold up to 6 entries for use by the forth interpreter to

Method Overview

public RobotAI(String json)
public String getLibTeam()
public Stats getPreviousStats()
public String getPlayCommand()
public String getInitCommand()
public JSONObject toJSON()
public HashMap<String, String> getWords()
public Queue<Mail> getMail()

Method Writeups

public RobotAI(String json)
Create a RobotAI object with its corresponding JSON string

public String getLibTeam()
Get the team that created this robot.

public Stats getPreviousStats()
Get the statistics saved about this robot

public String getPlayCommand()
Get the forth script that that will execute this robots play

private String getInitCommand()
Get the command that instantiates this robots words.

public JSONObject toJSON()
Get a JSONObject representation of this Robot

public HashMap<String, String>getWords()
Get the map of defined words for this robot.

RobotBuilder

CRC

RobotBuilder	
Responsibilities RobotBuilder is responsible for setting all the variables for a robot.	Collaborators RobotBuilder collaborates with Robot.

Description / Overview

RobotBuilder is part of the Model in our architecture, it gives us an easy interface for creating robots and modifying single aspects about them without needing a gargantuan constructor for our Robot class.

Instance Variables

Method Overview

public RobotBuilder getTank()
public RobotBuilder getSniper()
public RobotBuilder getScout()
public RobotBuilder setHealth(int health)
public RobotBuilder setDamage(int damage)
public RobotBuilder setRange(int range)
public RobotBuilder setMoves(int moves)
public RobotBuilder setType(RobotType type)
public RobotBuilder setPosition(HexNode node)
public RobotBuilder setName(String name)
public RobotBuilder setFacing(int facing)
public RobotBuilder setTeamColour(TeamColour colour)
public Robot build();
public Robot build(JSONObject json)

Method Writeups

public Robot getTank()

This method sets the variables health and maxHealth, damage, range, maxMoves, and type to the default values for a tank robot.

public Robot getSniper()

This method sets the variables health and maxHealth, damage, range, maxMoves, and type to the default values for a sniper robot.

public Robot getScout()

This method sets the variables health and maxHealth, damage, range, maxMoves, and type to the default values for a scout robot.

public Robot build()

This method returns the current Robot with the new changes made to the robot's variables.

public RobotAI build(JSONObject json)

This method returns the RobotAI with the new changes made to the robot's variables.

RobotStatView

CRC

RobotStatView	
Responsibilities To display the statistics of the local RobotAIs.	Collaborators AbstractView to control the users input on the Interface. LibMaster to access robots and their statistics.

Description / Overview

A User Interface that contains a list of robots to be selected, options to sort the robots, and the currently selected robot's statistics. This Interface will be controlled by a ViewController and uses the Librarian to access robots and their information. There is a selection box to choose a robot, a textbox to view the RobotAI stats, a box to sort the robots in the selection box, and 3 buttons to upload the statistics, reset the statistics, and go back to the main menu.

Instance Variables

LIBMASTER

Data Type: *LibMaster*

Contains a private reference to the libMaster that contains the librarian to help access the robots

Method Overview

public RobotStatView(libMaster : LibMaster)

Method Writeups

public RobotStatView(LibMaster)

Constructs a RobotStatView Screen using Libmaster to access the Statistics.

RuleSubView

CRC

RuleSubView	
Responsibilities RuleSubView is responsible for displaying the rules for the game.	Collaborators RuleSubView collaborates with AbstractView.

Description / Overview

RuleSubView simply displays a frame with the rules for the game, as well as a back button which closes the frame. The frame can be opened either from the main menu or by the observer or player while the game is running.

Instance Variables

Method Overview

public RuleSubView()

Method Writeups

public RuleSubView()

RuleSubView is the constructor that creates a frame to be shown to the user on top of the current frame. It displays a text containing the rules, as well as a button to close the frame.

Stats

CRC

Stats	
Responsibilities The purpose of the Stats class is to keep track of all the different stats for each Robot.	Collaborators Stats collaborates with Robot.

Description / Overview

The Stats class is a part of the Model in our architecture, and is a class that keeps track of all stats for each Robot. Stats holds the wins, matches, distance traveled, damage taken, damage given, plays made, kills, and deaths for each Robot. Stats can also increment each of these and determine the number of losses for the Robot.

Instance Variables

WINS

Data Type: *Integer*

Wins is an Integer that keeps track of how many times a robot has won a game.

MATCHES

Data Type: *Integer*

Matches is an Integer that keeps track of how many matches a robot has played.

DISTANCETRAVELED

Data Type: *Integer*

DistanceTraveled is an Integer that keeps track of how many spaces a robot has traveled.

DAMAGETAKEN

Data Type: *Integer*

DamageTaken is an Integer that keeps track of how much damage a robot has taken.

DAMAGEGIVEN

Data Type: *Integer*

DamageGiven is an Integer that keeps track of how much damage a robot has given to other robots.

PLAYS

Data Type: *Integer*

Plays is an Integer that keeps track of how many plays a robot has made.

KILLS

Data Type: *Integer*

Kills is an Integer that keeps track of how many robots a robot has killed.

DEATHS

Data Type: *Integer*

Deaths keeps track of how many robots a robot has killed.

Method Overview

public Stats()
public Stats()
public int getWins()
public int getMatches()
public int getDistanceTraveled()
public int getDamageTaken()
public int getDamageGiven()
public int getPlays()
public int getKills()
public int getDeaths()
public int getLosses()
public void addMatch(boolean win)
public void addDistanceTraveled()
public void addDamageTaken(int damage)
public void addDamageGiven(int damage)
public void addPlay()
public void addKill()
public void addDeath()
public void mergeStats(Stats other)

Method Writeups

public void addMatch(boolean win)

This method will increment the number of matches played by the robot by one.

public void addDistanceTraveled()

This method will increment the distance traveled.

public void addDamageTaken(int damage)
This method will increment the total damage taken by the robot by the Integer damage.

public void addDamageGiven(int damage)
This method will increment the total damage given by the robot by the Integer damage.

public void addPlay()
This method increments the number of plays for the robot by one.

public void addKill()
This method increments the number of kills for the robot by one.

public void addDeath()
This method increments the number of deaths for the robot by one.

public void mergeStats(Stats other)
This method merges the current set of variables in Stats with another set of variables Stats other.

Team

CRC

Team	
Responsibilities Keep track of team information, such as: references to robots on the team, what colour the team is, which robot is next in line to play, and if the team is active.	Collaborators The Team class collaborates with the Game.

Description / Overview

The Team class is going to keep information about a specific team. A few of these things would be: which robot on that team is next to play, keeping in account which robots are dead and which robot has recently moved. As well as if the team still has any robots alive making the team active or inactive.

Instance Variables

SCOUT

Data Type: Robot
This holds a reference to a robot with its type attribute Scout.

TANK

Data Type: Robot
This holds a reference to a robot with its type attribute Tank.

SNIPER

Data Type: Robot
This holds a reference to a robot with its type attribute Sniper.

LASTROBOT

Data Type: Robot

This holds the reference to the robot that has most recently played.

COLOUR

Data Type: *TeamColour*

This holds an enumerator for the colour this team represents. (i.e. Red, Orange, Blue, etc.)

ENABLED

Data Type: *boolean*

This holds a true value when it has remaining live robots, and holds a false value when all robots on the team are dead or if the team is not currently in the game.

Method Overview

public Team Team(TeamColour colour, boolean enabled)
public Robot getScout()
public Robot getTank()
public Robot getSniper()
public Robot getNextRobot()
public TeamColour getColour()
public boolean IsEnabled()

Method Write-ups:

public Team Team(TeamColour colour, boolean enabled)

Constructor for the Team class. Assigning a colour to Colour attribute, enabled to Enabled attribute. Each robot will be assigned through the Robot Builder.

public Robot getScout()

Returns the reference of the Robot with the type attribute Scout.

public Robot getTank()

Returns the reference of the Robot with the type attribute Tank.

private Robot getSniper()

Returns the reference of the Robot with the type attribute Sniper.

public Robot getNextRobot()

By checking the lastRobot attribute this function will return the robot who is next in line to play.

public TeamColour getColour()

Returns the teams colour.

public boolean isEnabled()

Checks to see if the team is enabled in order to know if it should play or not.

ViewController

CRC

ViewController	
Responsibilities Change between views and keep references to ones that can be reused	Collaborators Everything that extends AbstractView

Description / Overview

The ViewController is our main controller that will handle changing between views in the system. ViewController is also responsible for creating the GameMaster which is a sub controller in GameView and ObserverView.

Instance Variables

CURRENTVIEW

Data Type: *JPanel*

A reference to the view that is currently being displayed.

MAINMENUVIEW

Data Type: *MainMenuView*

A reference to the Main Menu View to avoid re-instantiating it later.

CREATEGAMEVIEW

Data Type: *CreateGameView*

A reference to the Create Game View to avoid re-instantiating it later.

ROBOTSTATVIEW

Data Type: *RobotStatView*

A reference to the RobotStatView to avoid re-instantiating it later.

ROBOTMANAGERVIEW

Data Type: *RobotManagerView*

A reference to the RobotManagerView to avoid re-instantiating it later.

RULEVIEW

Data Type: *RuleSubView*

A reference to the RobotManagerView to avoid re-instantiating it later.

Method Overview

public ViewController()
public void setView(JPanel view)
public void showRules()

Method Writeups

public ViewController()

Create the view controller and instantiate all of its fields.

public setView(JPanel view)

Set the current view to the specified view

public void showRules()

Display the Rule view above the current view. The RulesSubView is unique as it appears above the current view and does not move the user to a new screen.

Word

CRC

Word	
Responsibilities Contains a trigger and an executer for a Forth command	Collaborators ForthInterpreter holds all the words

Description / Overview

A word contains a string that will trigger the executor for the word.

Instance Variables

TRIGGER

Data Type: *String*

A regular expression that will trigger this word.

MATCH

Data Type: String

The string that the regular expression pulls from the trigger

EXECUTOR

Data Type: ForthExecutor

A function interface that will execute the forth command on the stack or through the GameMaster

Method Overview

public Word(String trigger, ForthExecutor executor)
public boolean isTrigger(String command)
public void execute(String input)

Method Writeups

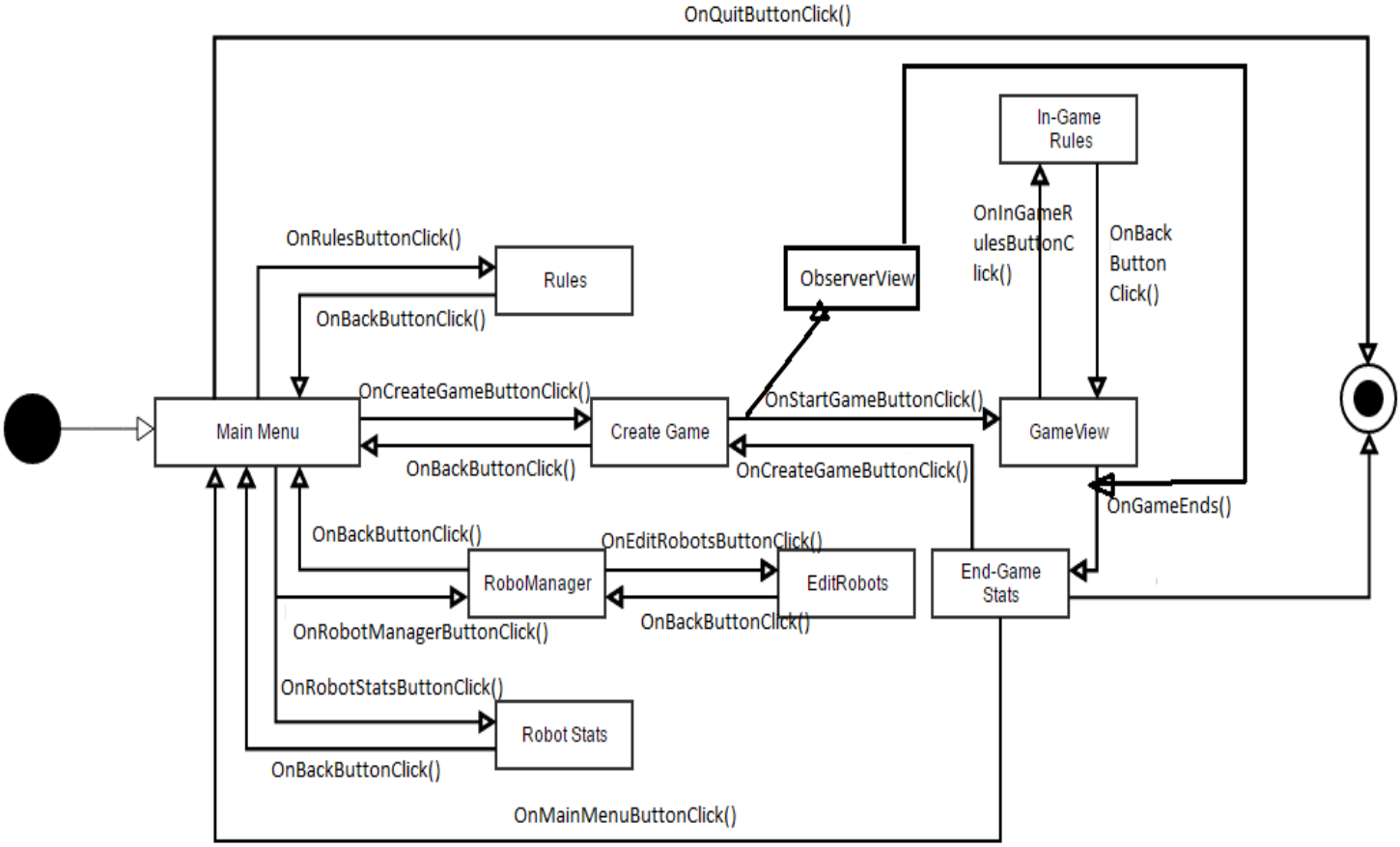
public Word(String trigger, ForthExecutor executor)

Creates a new Word and instantiates its fields.

public boolean isTrigger (String command)
Checks if the string is a trigger for the Word and sets the match if it is.

public void execute(String input)
Executes the executor corresponding to this word.

State Transition Diagram



The Program will enter the Main Menu state where it will contain 5 buttons. OnQuitButtonClick() will immediately terminate the program. OnRulesButtonClick() will bring up a rules interface. From the rules interface the only event is OnBackButtonClick() which will bring the user back to the main menu. OnRobotStatsButtonClick() brings up the user interface, and from there OnBackButtonClick() will bring the user back to the main menu. OnRoboManagerButtonClick() the robot management user interface will be brought up where the user can manage their robot OnEditRobotButtonClick() and return to the managers OnBackButtonClick() and then to the main menu with another OnBackButtonClick() event. OnCreateGameButtonClick() the interface to enter data that will be used to create a game. from here OnBackButtonClick() will bring the user back to the main menu, or OnStartGameButtonClick() will take the data and create a game view interface or an observer view interface for the user. From game view OnInGameRulesButtonClick() brings up rules and then OnBackButtonClick() brings the user back into the game. OnGameEnds() happens when there is only one color of robots remaining from either game view and

OnMainMenuButtonClick() will bring the user back to the main menu, or OnQuitButtonClick() will terminate the game.

System Finalization

Networking was the major dispute, although there are many upsides to providing networking to the game, we ultimately decided to not include it. If we were to include it, the game would have to function differently depending on how many players there are which adds complexity. One benefit of having networking is that when multiple human players are in the same match, they cannot see their opponents point of view. Since we limited the game to one device, we decided to compensate for the lack of networking with a turn delay; After any "play" has ended and the next play is a human's, then the board will blackout and a dialog will pop-up. It will just be a button to simply confirm they are the indicated player so that no one gains extra intel about the other teams. Animation was another could-have option that resulted in us not including it in our game. We wanted to focus on nailing the primary functionalities of the game and potentially "buggy" animations would take away from the simplicity that our interface and game is meant for. Sound was the last could-have feature, again we decided not to include it for the same reason as animation; we believed it would take away from the game if it was not perfectly timed with the actions in the game and focusing on functionality was more important. Lastly, stats were a required feature in the game but we decided to store all the stats locally and expand on the stats from the required ones. We believe having a local copy of all the robots provides a nice way to analyze data and allows the user to pick and choose robots based on their previous games (which serves as a difficulty changer). In the requirements document we did not provide an interface that delayed turn transitions for multiple human players; this will be an interface we must add in our implementation. Other than that, the system will function as explained in the requirements document and will be built on the foundations we explained throughout this document.