

Entities and their method's description:

Model

GameBoard:

The GameBoard is a model entity in our system. It keeps track of all the information pertaining to the game board. The GameBoard varies in size depending on the amount of teams in play. If two or three teams are playing, the game board will be size 5. If six teams are playing, the game board will be size 7. The methods **getSize()** and **setSize()** will determine the correct board size and set up the game based on the correct board. The GameBoard also retains any statistics created during the game in progress. Examples of this would be a robot damaging another robot, or a robot getting damaged. These damages (both given and taken) need to be recorded and added to the robot library's statistics after the game. The method **updateImmediateStatistics(RobotID, Stat, Value)** handles the ongoing statistics in the game by taking in the Robot Identifier, which is unique to each robot, what statistic it should change, as well as how much it should change it by. The related method **getImmediateStatistics()** can be used to grab any ongoing statistic in the game. The method **getAllRobots()** returns a list of all robots playing in the game. Lastly, the method **isValid(BoardPosition)** is a boolean that checks if a specific board position is valid.

Robot:

The Robot is a model entity in our system. It holds of all the information pertaining to the robots of the system. Each robot (Sniper, Tank, Scout) have a unique *health*, *range*, and *damage*, all of which are expressed as member variables. Each robot will either be controlled by a human player, or an AI. An AI refers to the Script created to dictate the robots moving and shooting behaviours. To see what is controlling a robot's playing, run **isAI()**. To acquire that robots specific AI script, run **getAI()**. Each Robot will be unique and thus has a way to uniquely identify itself: the *identifier* member. Lastly, when a Team of Robots are not being controlled by human players, but rather AI, they have the ability to communicate with each other. This communication will be achieved through accessing the *mailbox* member, which is a list of the messages the robot has received. The *variables* member is a map indexed by the string name of the variable, keyed to the value. This is used by the NpcController's ForthInterpreter to store the robot's user (AI) defined variables and their values.

BoardPosition:

BoardPosition is a model entity in our system. BoardPosition is responsible for knowing the position represented on the game board, as well as how it's current position correlates to another position on the gameboard. The **turn(Direction)** method changes the direction of the

board position given a specific point of direction, number 1-5. The method **advance()** then moves the position it is currently in by one in the direction it is currently facing. Next, the method **directionTo(BoardPosition other)** will be used on another board position to determine the direction required to change the current position to the *other* position, usually used in conjunction with **distanceTo**. The related method **distanceTo(BoardPosition other)** is the distance between the current position and the position *other*. Another method **immediateDirection(BoardPosition other)** returns the immediate neighborhood direction towards the position *other* (if ambiguous, decide pseudorandomly). Finally, the **getRelative(int direction, int distance)** method returns a new BoardPosition, which has the position relative to the current position dictated by the direction and distance argument.

Team:

The Team entity is a model within our system. A Team consists of three Robots (a Sniper, Tank and Scout) expressed as a list of these robots in the member variable robot. Each Team will be unique and thus has a way to uniquely identify itself: the *identifier* member. When a game is created, each team that is part of the game is assigned a random unique color to be played, achieved by **setColor(color)**. Color dictates both starting position on the board and their respective turn orders. Also, the colors available will be dictated by the number of players in the game, so that all teams start in the correct starting positions. **getColor()** returns the team color, used in board positioning and turn sequences.

Robot Library Socket:

The RobotLibrarySocket is a model entity in our system. The RobotLibrarySocket is to act as the half-way-house between the rest of our system and the robot library. To update the statistics in the robot library, it will run the **updateStatistics(Gameboard)** method. This will take place at the end of every game played. The GameBoard contains the statistics from the game played and thus is passed into the method to be used. To request a list of all of the Teams or Robots contained in the robot library the **enumerateTeams()** as well as the **enumerateRobots()** methods can be used, respectively. It may be of necessity that only one specific Robot or Team be accessed from the robot library, in which case the methods **retrieveRobotFromLibrary(RobotID)** and **retrieveTeamFromLibrary(TeamID)** can be used. It should be noted that the Robot Library only sends and receives JSON files, so it is the Robot Library Socket's job to create and translate these files in its requests.

View

GameDisplay:

The GameDisplayer is a view entity in our system. This entity takes care of all the visuals during the playing of a game. Using the method **displayGame()** it will show the gameboard, and any teams, or robots necessary in their appropriate positions as well as directions. With the **close()** method it will exit the screen of the game.

SetupDisplay:

The SetupDisplay is a view entity in our system. This entity takes care of all the visuals for the GUI's during the setup of the game. This entity will display a GUI with the **displaySetup()** method, which brings forth a particular GUI dependant on the specific button clicked by the user. The **close()** method then exits the GUI needed.

Controller

Referee:

The Referee is a controller entity in the system. This entity manages the game while it is being played. The **startGame(GameBoard)** method will begin the playing of the game, allowing the red team to begin their first round. The (hopefully infrequently used) method **abandonGame()** will stop the game in progress. This will be useful for testing as well as error handling. The **updateGameBoard()** method calls the GameDisplayer entity to update the game display. This will take place after a change in state of the game has occurred, such as a movement, or a death of a Robot. Since the RobotController needs to acquire the GameBoard, the Referee has the method **getGameBoard()**, which returns the GameBoard. Finally, the Referee also handles the case of when a Team decides to leave the game. The **teamQuit(Team)** method will remove said Team from the game. The game, given that there is still more than one Team on the board, will continue to play as normal, minus the quitting Team's robots.

Game_INITIALIZER:

The Game_INITIALIZER is a controller entity in the system. **initializeGame()** will return a Gameboard. All the information given by the user during the SetupDisplay will be passed to the Game_INITIALIZER, it will run **initializeGame()** with this information and create a Gameboard to be used for the playing of the game.

Note: The AI scripts for the NPC Robots, if any, are only loaded into the Forth interpreter after all the Teams and their corresponding Robots have been selected for the game and the game is about about to begin.

RobotController:

The RobotController is a controller entity in the system. It manages the actions of the robots during the game. Since the robots have the capabilities to play, quit, scan, shoot, and send

a message to other robots, the RobotController then has to manage these. **play()** is called when it is a specific robots' turn to do actions, and all of it's actions are done inside play. The **quit()** method sends a message to the RobotController to remove the team that that robot is on from the game board. **scan()** is used to see what robots, both enemy and team, are within visual range of the playing robot. The robots that are in the range are then visible to that robot and displayed on the board. **shoot(int distance, int direction)** is used to shoot a space. This is done by passing in the distance from the robot to the space wanted to shoot on, as well as the direction between the robot and the space wanted to shoot on. NPC robots have the ability to talk to one another, when a robot wishes to send a message it uses **send(Robot, String)**. This method, given the specific robot it wants to send a message to, and the string containing a message, will add the string to that robot's mailbox. RobotController has a *myRobot* variable, containing a Robot so it knows which robot to take these actions on. It also has a *myReferee*, containing the Referee, in order to call it's methods when needed, Such as **teamQuit(Team)** when a robot initiates **quit()**.

Script:

The Script is a model entity in our system. This entity contains the AI logic for the NPC robots. This logic is held as a Map of words, and gets run in **play()**.

ForthInterpreter:

The ForthInterpreter is a controller in our system. This entity, given a Script, and after translating the Script, executes the given code.

PcController

A PcController, as mentioned in the UML, describes a human controlled player that interacts with the system.

NpcController

A NpcController, as mentioned in the UML, describes a non-human interaction with the system.