# Design Document

**Project Name:** RobotGo

**Team:** CMPT370-B3

**Team Member:** Cheng, Gong
Hounjet, Carlin
Lan, Shaoxiong
Xie, Joey
Yue, YiRui

**Date:** Oct 23, 2016

# Table of Contents

# Part 1 Architecture

Our project ended up being a combination of Model View Controller with some aspects of independent components integrated.

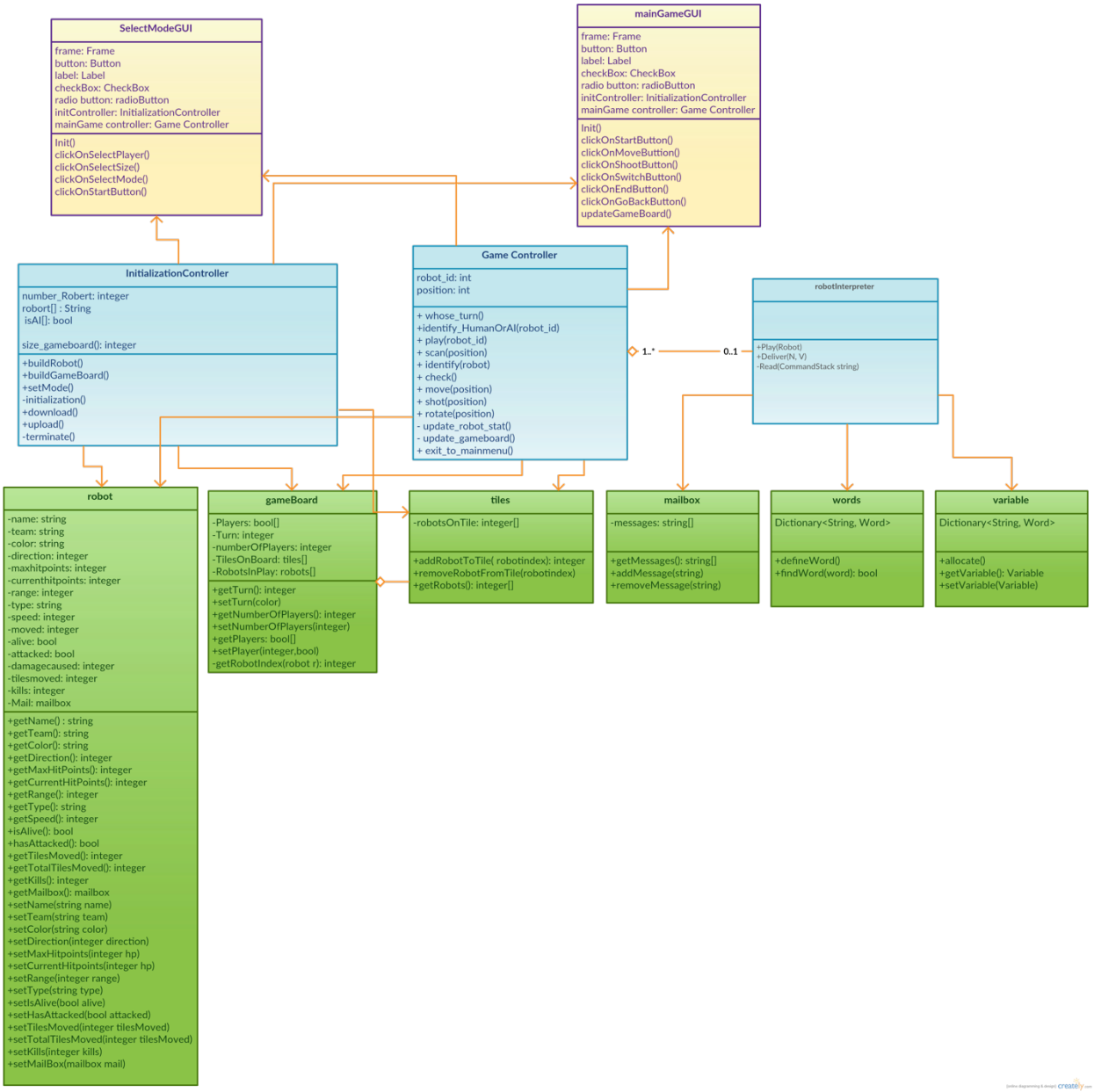
This integration of different architecture was necessary to separate the user's different interactions between the Setup Menu and the Main Game, as well as to incorporate the Robot Interpreter as an internal component. We thought of all three of these components as separate instances of MVC where aspects of a data-flow system are used to connect the Setup Menu and the Main Game through initialization of the rules, the teams, tiles, robots and the Gameboard while the robot interpreter interacts with the main game through event-based function calls.

We focused on thinking of the main components as MVC for three main advantages. First, the main purpose of MVC is separation of concerns, which allows us to keep logic from becoming muddled to allow ourselves and the system a straightforward understanding of all interactions. The second advantage of MVC comes from having clear representations and storage of objects in the Model that are connected via the the game rules and logic contained in the Controller. This works best for playing a game via our Interface. Thirdly we are all familiar with it from having worked with it in previous classes. This gives us an advantage in designing and documenting our system.

Having an overarching independent components architecture allows us to break up our system in necessary ways. It was necessary to have two separate controllers for the game setup and the game play because the interfaces and information interacted with and viewed are very much distinct. This allowed us to remove unnecessary connections and use data-flow to streamline the process of initialization. It was also very necessary to separate the robot interpreter from the controllers because it contains a state machine for reading and writing Forth script to run our Robot's AI. We still kept the style consistent with MVC despite it being an interpreter and technically doesn't interact directly with the view. It does however require a controller and models and behaves much in the same way. The only difference being that it interacts with the view via event-based function calls to the MainGameController class.

# Part 2 Class and UML Class Diagram

## 2.1 UML Class Diagram




- View: the yellow part of the diagram
- Controller: the blue part of the diagram
- Model: the green part of the diagram

## 2.2 InitializationController

### 2.2.1 Class-responsibility-collaboration card

| InitializationController | |
|---|---|
| Responsibilities | Collaborators |
| • Build the robots<br>• Build Gameboard and tiles<br>• Start the game<br>• Terminate(shut down) the game<br>• Select Mode<br>• Select robots(Human/AI)<br>• Download from and to JSON file | • slectMode Interface class<br>• gameboard model class<br>• tiles model class<br>• robot model class |

### 2.2.2 Initialization Controller Class

| InitializationController |
|---|
| int number_Robert<br>string robort[];<br>Boolean isAI[];<br>Int size_gameboard(); |
| +buildRobot(robot[]);<br>+buildGameBoard(size_gameboard);<br>+setMode(isAI[]);<br>-initialization();<br>+download(robot_id);<br>+upload();<br>-terminate(); |

### 2.2.3 Function explanations

buildRobot(robot[])

- build robots and pass their statics to Robot model.
- Pre: after the player choose the robot number.
- Post: the statics has been built and stored temporarily.
- Return: robots script

buildGameBoard(size_gameboard())

- take the size of the board and build the board correspondingly.
- Pre: after the player choose the game board size.
- Post: game board has been built.

- Return: nothing

setMode(isAI[])
- take the input of which robot is Human and which robot is AI, put these data into robot model.
- pre: after the player choose the robot or AI.
- post: corresponding data has been put in robot model.
- Return: Boolean, true if the data was put in robot model successfully, false otherwise

initialization()
- take the scrip from the model and start the game
- pre: the robot scripts have been built in the model and downloaded successfully. The game board has been built successfully
- Post: the game starts
- Return Boolean, true if the game starts successfully, false otherwise

download(robot_id)
- download the script from model
- pre: script was in model object
- post: the script was downloaded successfully
- return: Boolean, true downloaded if downloaded successfully, false otherwise

upload()
- upload the script from model
- pre: data was successfully subtracted
- post: the script was downloaded successfully
- return: Boolean, true if uploaded successfully, false otherwise

terminate()
- shut down the game
- pre: terminate key was pressed.
- Post: the program terminates
- Return: nothing

## 2.3 Game Controller Class

### 2.3.1 Class-Responsibility-Collaborator card

| Game Controller | |
|---|---|
| Responsibilities | Collaborators |
| • Identify the turn & round<br>• Identify if it's a human robot or AI<br>• Play function<br>• Move a robot<br>• Shot a robot<br>• Rotate a robot<br>• Update the robot statistics in robot model<br>• Update the game board if needed<br>• Exit to the main menu | • Gameboard Interface class<br>• Robot model class<br>• Words model class<br>• Mailboxes model class |

### 2.3.2 Game Controller Class

| Game Controller |
|---|
| robot_id: int<br>position: int |
| + whose_turn()<br>+identify_HumanOrAI(robot_id)<br>+ play(robot_id)<br>+ move(position)<br>+ shot(position)<br>+ rotate(position)<br>- update_robot_stat()<br>- update_gameboard()<br>+ exit_to_mainmenu()<br>+ scan()<br>+ check()<br>+ identify() |

### 2.3.3 Function explanations

whose_turn()

- identify which robot should be in the battle field for a specific turn.
- pre: the start button is clicked on the interface of 'selecting robot'

- post: identify which robot should be played in the current turn
- return: the robot_id of the robot

identify_HumanOrAI(robot_id)

- identify the robot that is going to be played should belong to a human team or AI team
- pre: after the turn is identified
- post: identify the robot team
- return: nothing

play(robot_id)

- if the robot is played by the human, call functions move/attack/rotate directly inside the game controller to finish play; if the robot is played by AI, call interpreter class to translate
- pre: the turn and team of a robot has already been identified
- post: pass the robot to interpreter if it's AI, or just call functions move/attack/rotate directly inside.
- return: nothing

rotate()

- turn the direction of a robot
- pre: the function play is called. The robot already chooses its destination to move to.
- post: the robot rotates to the desired direction, update the gameboard interface
- return: nothing

move()

- move a robot on the game board
- pre: the function play is called. The robot has at least one move point
- post: the robot moves to the desired destination, update the gameboard interface
- return: nothing

attack()

- attack a robot if in range
- pre: the function play is called. The robot doesn't use its attack point
- post: the robot finishes it attack action, update the gameboard interface in needed.
- return: nothing

update_robot_stat()

- keep track the statistics of robots in each turn. Collaborate with robot model.
- pre: each time a robot rotates, moves or shoots.
- post: change the robot statistics in robot model.
- return: nothing

update_gameboard()

- update the game board interface in each turn. Collaborate with game board interface.
- pre: turn change, robot dead or
- post: change the robot statistics in robot model. Update the game board interface.
- return: nothing

exit_to_mainmenu()
- exit to the main menu on clicking the exit button
- pre: on clicking the exit button.
- post: exit to the main menu.
- return: nothing

scan()
- scan the game board, get the enemy robot position on the game board.
- pre: it's the AI' s term
- post: get the position of enemy robot
- return: nothing

check()
- check if the adjacent space of a given direction is occupied or not
- pre: given a specific direction
- post: pushes a string describing the adjacent space in that direction
- return: nothing

identify()
- identify the team color, range, direction and remaining health of a given target.
- pre: after scanning and given a specific robot.
- post: identify the team color, range, direction and health of the given robot.
- return: nothing

## 2.4 RobotTranslator Class

2.4.1 Class-Responsibility-Collaborator card

| **RobotTranslator** | |
|---|---|
| Responsibilities | Collaborators |
| • Read Robot scripts and make function calls<br>• Translate returned information into Forth to be pushed onto the stack<br>• Receive and store mail | • Game Controller class<br>• Variables model class<br>• Words model class<br>• MailBoxes model class |

2.4.2 RobotTranslator Class

| RobotTranslator |
|---|
| +Play(Robot)<br>+Deliver(N, V)<br>-Read(CommandStack string)<br>-Deliver(N, V)<br>-Play(Robot)<br>-Arithmetic(value, value, operator)<br>-Define(string)<br>-Compare(value, value, operator)<br>-DeclareVar(Type, value)<br>-Print(String)<br>-Random()<br>-FinalAction(String)<br>-Action(String) |

2.4.3 Function explanations

Deliver(N, V)

- Adds Forth script to save message (V) from teammate (N) into a robot's mailbox to command stack string.
- pre: get the
- post: the message is sent to a specific teammate
- return: nothing

Play(Robot)

- Calls Status() from game controller and translates results into Forth script that sets variables for the robot about to play and stores it in a string for the command stack.
- pre: it's the robot's turn
- post:Adds the robot's script to the string.
- Returns: String

Read(CommandStack string)

- Parce tokens from CommandStack string (everything delimited by a space) and populates the CommandStack linked list.
- While there exists items on the CommandStack linked list, pop of Tokens and read them.
- If token is a literal word, push it onto the DataStack. If it's any other kind of word, search the dictionary to identify it and execute it. Words can create loops, if/then statements, do arithmetic, create function calls to Main Game controller, or end the Read function and return a function call
- Returns: function call

Arithmetic(value, value, operator)
Takes in values and an operator from an arithmetic word and completes the operation
Returns: value

Define(string)
if Read() decides we're defining a word if it searches for a word and it's already defined or not from the return value of the search. If it isn't then we break up the forth code and define a word.
Return: Boolean

Compare(value, value, operator)
Checks out what kind of comparison is called and figures it out
Return: Boolean

DeclareVar(Type, value)
Checks what kind of variable to declare, searches if it exists and sets it or creates it

Print(String)
Takes in string off the Data stack after "." word is called

Random()
Returns random number
Return:Int

FinalAction(String)
if Read() dicovers an action that ends play, this function figures out which kind and clears the stacks then sends off the appropriate function call

Action(String)
If Read() discovers an action that requires a function call and return value, then it calls that function and translates it into Forth code to push onto data stack

2.4.4 loop description:
Loops
To handle loops we will need specially defined words to handle their logic

Do
pushes end and start of a loop onto loop stack

I
checks to see if there is already a non zero value stored in loop variable (i), if there isn't then it stores the first pop from the loop stack. If there is, it creates a command to fill another forth defined variable (j, k, l,...) from the loop stack and that becomes the one it returns until Loop() removes it. The programmer will have to just remember the depth of the loop that they want the index value from.

Loop
Adds 1 to value of the integer stored in it's loop level's I
Pops top two values from the loop stack, pushing the second one followed by I

Leave
Set I to equal second pop from loop stack, pop until pop=loop then run that

## 2.5 selectModeGUI Class

2.5.1 Class-Responsibility-Collaborator card

| selectModeGUI | |
|---|---|
| Responsibilities | Collaborator |
| • show the selectMode interface to user<br>• wait for user's action to select player number<br>• wait for user's action to select game board size<br>• wait for user's action to select human/AI mode for each player | • selectMode Controller class<br>• mainGame Controller class |

2.5.2 selectModeGUI Class

| selectModeGUI |
|---|
| Frame<br>Button<br>Label<br>Checkbox<br>Radio button<br>selectMode controller<br>mainGame controller |
| Init()<br>clickOnSelectPlayer()<br>clickOnSelectSize()<br>clickOnSelectMode()<br>clickOnStartButton() |

2.5.3 Function explanations

Init()

- initialize and show the select mode interface to user, waiting for their action
- pre: the game program is open
- post: initialize and show the select mode interface to user
- return: nothing

clickOnSelectPlayer()

- when user clicks the button to select player number, then send message to selectMode controller to set the number of players

- pre: when user clicks the button to select player number
- post: send message to selectMode controller to set the number of players
- return: nothing

clickOnSelectSize()

- when user clicks the button to select game board size, then send message to selectMode controller to set the game board size
- pre: when user clicks the button to select game board size
- post: send message to selectMode controller to set the game board size
- return: nothing

clickOnSelectMode()

- when user clicks the button to select Human/AI mode for each player, then send message to selectMode controller to set the Human/AI mode for each player
- pre: when user clicks the button to select Human/AI mode for each player
- post: send message to selectMode controller to set the Human/AI mode for each player
- return: nothing

clickOnSubmitButton()

- when user clicks on the submit button, then send message to selectMode controller to close selectMode interface and game controller to open and initialize the game interface
- pre: when user clicks on the submit button
- post: send message to selectMode controller to close selectMode interface and game controller to open and initialize the game interface
- return: nothing

## 2.6 mainGameGUI Class

2.6.1 Class-Responsibility-Collaborator card

| mainGameGUI | |
|---|---|
| Responsibilities | Collaborator |
| • show the main game interface to user<br>• wait for user's action to click Start button<br>• wait for user's action to click tiles<br>• wait for user's action to click switch button<br>• wait for user's action to click end or goBack button<br>• update the game board in time | • selectMode Controller class<br>• mainGame Controller class |

2.6.2 mainGameGUI Class

| mainGameGUI |
|---|
| Frame<br>Button<br>Label<br>Checkbox<br>Radio button<br>selectMode controller<br>mainGame controller |
| Init()<br>clickOnStartButton()<br>clickOnMoveButtion()<br>clickOnShootButton()<br>clickOnSwitchButton()<br>clickOnEndButton()<br>clickOnGoBackButton()<br>updateGameBoard() |

2.6.3 Function explanations

Init()

- initialize and show the main game interface to user, waiting for their action
- pre: when user clicks on the submit button in selectMode interface
- post: initialize and show the main game interface to user

- return: nothing

clickOnStartButton()
- when user clicks on the start button, then send message to mainGame controller to begin the game
- pre: when user clicks on the start button in mainGame interface
- post: send message to mainGame controller to begin the game
- return: nothing

clickOnMoveButton()
- when user clicks on the move button, then send message to mainGame controller to move the robot
- pre: when user clicks on the move button
- post: send message to mainGame controller to move the robot
- return: nothing

clickOnShootButton()
- when user clicks on the shoot button, then send message to mainGame controller to make current robot shoot and reduce the health of enemys
- pre: when user clicks on the shoot button
- post: send message to mainGame controller to make current robot shoot and reduce the health of enemys
- return: nothing

clickOnSwitchButton()
- when user clicks on the switch button, then send message to mainGame controller to end the behavior of current robot and begin the behavior of robot of next team
- pre: when user clicks on the switch button
- post: send message to mainGame controller to end the behavior of current robot and begin the behavior of robot of next team
- return: nothing

clickOnEndButton()
- when user clicks on the end button, then send message to mainGame controller to end the game and close the mainGame interface
- pre: when user clicks on the end button
- post: send message to mainGame controller to end the game and close the mainGame interface
- return: nothing

clickOnGoBackButton()

- when user clicks on the goBack button, then send message to mainGame controller to end the current game, close the mainGame interface and send message to SelectMode controller to open the selectMode interface
- pre: when user clicks on the goBack button
- post: the game is end and return to the selectMode interface
- return: nothing

updateGameBoard()

- when the user clicks any button, the interface should update its gameboard based on mainGame controller
- pre: when user does any action on the mainGame GUI
- post: the gameBoard should be updated based on mainGame controller
- return: nothing

## 2.7. GameBoard Model

2.7.1 Class-Responsibility-Collaborator card

| GameBoard | |
|---|---|
| Responsibilities | Collaborator |
| • To store information on whose turn it is, which players are human or AI, the board size and the number of players.<br>• It will also have an array of the tile<br>• class and an array of all the robots that are in play. | • Tile<br>• Robot |

2.7.2 GameBoard Class

| GameBoard |
|---|
| -bool[] Players<br>-integer Turn<br>-integer numberOfPlayers<br>-tiles[] TilesOnBoard<br>-robots[] RobotsInPlay<br>-integer getRobotIndex(robot r) |
| +integer getTurn()<br>+void setTurn(integer turn)<br>+integer getNumberOfPlayers()<br>+void setNumberOfPlayers(integer number)<br>+bool getPlayers(integer index)<br>+void setPlayer(bool human,integer index)<br>+robot getRobot(integer index)<br>+tile getTile(integer index) |

2.7.3 Variable explanations.

- The array of booleans Players stores information on whether each player is a human or an AI. True for a human, false for an AI.
- The turn variable records whose turn it currently is, with a value of 0 to 1,2 or 5. The numberOfPlayers holds the number of players.
- The tilesOnBoard variable is an array of tiles that holds the instances of the tile class used in this game.

- The robotsInPlay variable holds an array of robots that holds the instances of the robot class used in this game.
- GetRobotIndex returns the index the robot holds in the RobotsInPlay array.

2.7.4 Function explanations

These functions are the accessor and mutator methods of the variables described above.

## 2.8 Robot Model

2.8.1 Class-Responsibility-Collaborator card

| Robot | |
|---|---|
| Responsibilities | Collaborator |
| • To store the name, team, color, direction, max hitpoints, current hit points, attack, speed, used movement points.<br>• To check whether the robot is alive or dead.<br>• To store the statistics of the robot over the game: damage taken, damage caused, tiles moved and kills. | • Mailbox<br>• Initialization controller<br>• Game controller |

2.8.2 Robot Model Class

| Robot Model |
|---|
| -bool[] Players<br>-string name<br>-string team<br>-string color<br>-integer direction<br>-integer maxhitpoints<br>-integer currenthitpoints<br>-integer range<br>-string type<br>-interger speed<br>-integer moved<br>-bool alive<br>-bool attacked<br>-integer damagecaused<br>-integer tilesmoved<br>-integer kills |

+string getName()
+string getTeam()
+string getColor()
+integer getDirection()
+integer getMaxHitPoints()
+integer getCurrentHitPoints()
+integer getRange()
+string getType()
+integer getSpeed()
+bool isAlive()
+bool hasAttacked()
+integer getTilesMoved()
+integer getTotalTilesMoved()
+integer getKills()
+mailbox getMailbox()
+void setName(string name)
+void setTeam(string team)
+void setColor(string color)
+void setDirection(integer direction)
+void setMaxHitpoints(integer hp)
+void setCurrentHitpoints(integer hp)
+void setRange(integer range)
+void setType(string type)
+void setIsAlive(bool alive)
+void setHasAttacked(bool attacked)
+void setTilesMoved(integer tilesMoved)
+void setTotalTilesMoved(integer tilesMoved)
+void setKills(integer kills)
+void setMailBox(mailbox mail)

2.8.3 Variable explanations.

- The variable name holds the name of the robot, team holds the name of the robot's team, color holds the the team's color for this game.
- The direction of the robot is represented by an integer with a value of zero to five.
- Maxhitpoints and currenthitpoints hold the maximum and current hit points of the robot. Range holds the range of the robot. Tanks and scouts have a range of one, snipers have a range of two.
- Type holds the robot type which is a string of value "sniper", "tank" or "scout".
- Speed is the amount of tiles a robot may move on each turn, a scout can move three, a sniper can move two and a tank can move one.

- Moved stores the number of tiles the robot has already moved this turn, this can not be higher than speed.
- Alive is a boolean that is true when the robot is alive and false when the robot is dead. Attacked is a boolean that is true when the robot has already attacked this turn and false when it has not. Damagedcasued stores the amount of damage thisrobot has caused this game.
- Tilesmoved holds the amount of tiles the robot has moved this game. Kills is the number of robots killed by this robot in this game.
- Mail holds an instance of the mailbox class, this will be the mailbox for this robot.

2.8.4 Function explanations

These functions are the accessor and mutator methods of the variables described above.

## 2.9 Tile Model Class

2.9.1 Class-Responsibility-Collaborator card

| **Tile Model** | |
|---|---|
| Responsibilities | Collaborator |
| To record which robots are on top of it. | • Mailbox<br>• Initialization controller<br>• Gameboard |

2.9.2 Robot Model Class

| Tile Model |
|---|
| -integer[] robotsOnTile |
| +integer addRobotToTile(integer robotindex)<br>+void removeRobotFromTile(integer robotindex)<br>+integer[] getRobots() |

2.9.3 Variable explanations.
RobotsOnTile is an array that stores the indexes of all the robots that are on top of it.

2.9.4 Function explanations
RobotsOnTile is an array that stores the indexes of all the robots that are on top of it.
AddRobotToTile adds the robot robotsOnTile and removeRobotFromTile removes it.
GetRobots returns the RobotsOnTile array.

## 2.10 Mailbox Model Class

2.10.1 Class-Responsibility-Collaborator card

| Mailbox Model | |
|---|---|
| Responsibilities | Collaborator |
| • To store messages so that AI controlled robots may communicate with each other. | • Robot Model |

2.10.2 Robot Model Class

| Mailbox Model |
|---|
| -string[] messages |
| +string[] getMessages()<br>+void addMessage(string)<br>+void removeMessage(string) |

2.10.3 Variable explanations.
RobotsOnTile is an array that stores the indexes of all the robots that are on top of it.

2.10.4 Function explanations
GetMessages will return this array, addMessage will add a message to the array while removeMessage will remove it.

**2.11 Words Model Class**

2.11.1 Class-Responsibility-Collaborator card

| Words Model | |
|---|---|
| Responsibilities | Collaborator |
| • Define words in dictionary<br>• Search the dictionary for words | • Robot interpreter |

2.12.2 Robot Model Class

| Words Model |
|---|
| -Dictionary<String, Words> |
| +defineWord(Name<string>, script<string>)<br>+findWord(String): bool |

2.11.3 Function explanations
findWord(string)
find searches for a word from the dictionary, throws exception if not found

defineWordcName<string>, script<string>)
Adds it to a dictionary of words. Throws exception if already defined

## 2.12 Variables Model Class

2.12.1 Class-Responsibility-Collaborator card

| Variables Model | |
|---|---|
| Responsibilities | Collaborator |
| • Retrieve values of variables<br>• Set values of variables | • Robot interpreter |

2.12.2 Robot Model Class

| Words Model |
|---|
| -Dictionary<String, Words> |
| +Allocate(Name, Value)<br>+Get(name)<br>+Set(Name, Value) |

2.12.3 Function explanations
Allocate(Name, Value):
Allocates Variable of given name and value or throws exception if it exists

Get(name)
retrieves variable value of given name, throws exception if it doesn't exist

Set(Name, Value)
Stores or allocates variable of given name and value, throws exception if it doesn't exist

## Part 3 Changes Made to Requirement Document

The biggest change from our requirements is that the robot librarian is no longer a part of our system.

Originally we thought that we had to implement our own robot librarian but that was completely unnecessary. Instead of doing the librarian functions ourselves our system will send requests to the librarian with JSON.

The second change is in our play action. In our requirements we had split up the play action into the smaller actions: move, shoot and turn but it seems this was problematic according to the feedback we received so we have changed it back into one action.

The third change is that originally we were going to allow the player to choose the robots from the library that he would play with, but now only the robots controlled by the AI will send statistics to the librarian.

The forth change is that we added the interpreter to the system, which was not mentioned in the requirements document but is clearly necessary.

One thing that we were considering were turn timers, but we decided against this as it was unnecessary.