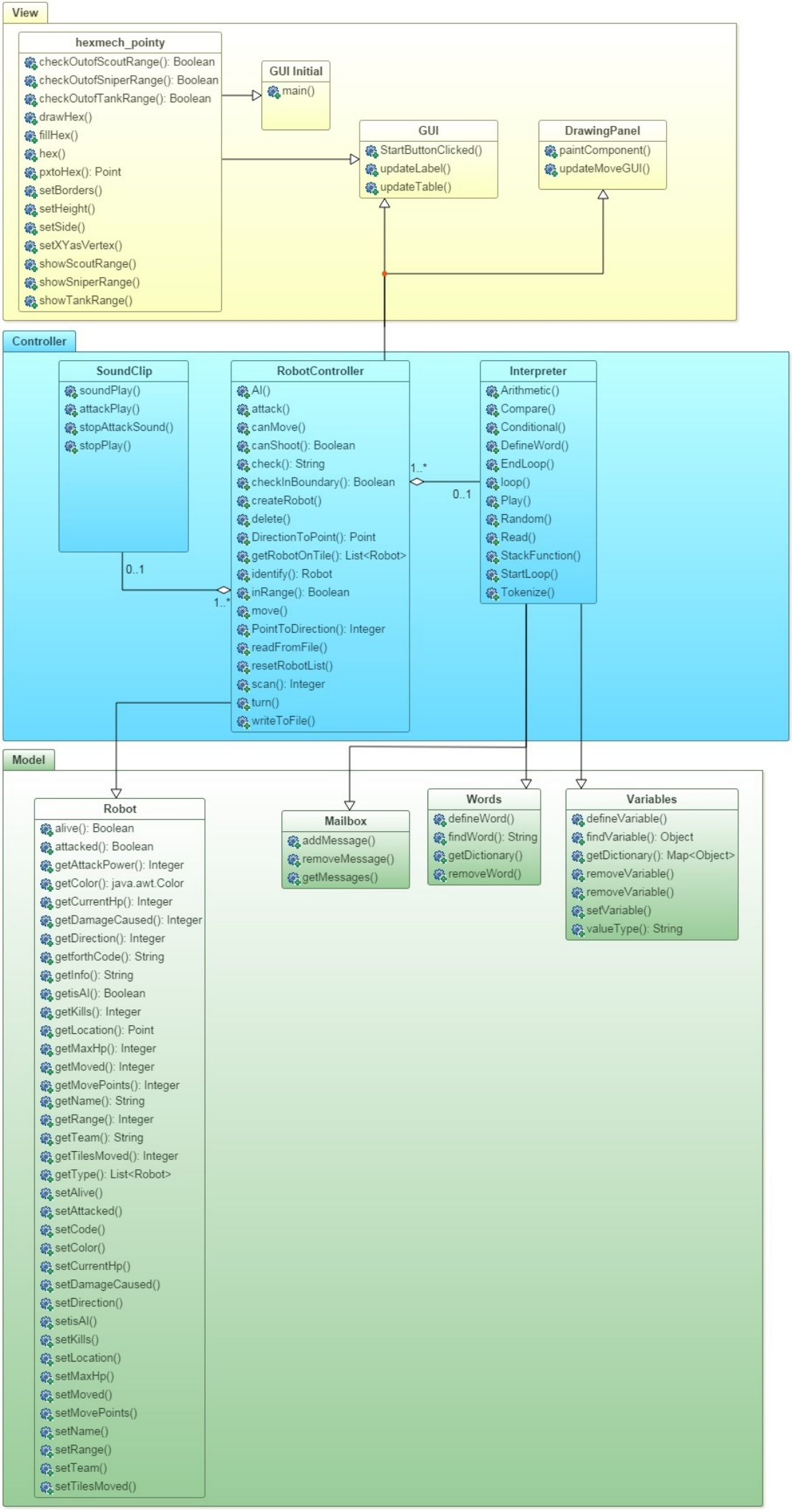
Programmer's Manual

Project Name:	RobotGo
Team:	CMPT370-B3
Team Member:	
	Cheng, Gong
	Hounjet, Carlin
	Lan, Shaoxiong
	Xie, Joey
	Yue, YiRui

Date: Dec 08, 2016

Table of Contents


View
hexmech_pointy
checkOutofScoutRange(): Boolean
checkOutofSniperRange(): Boolean
checkOutofTankRange(): Boolean
drawHex()
fillHex()
hex()
pxtoHex(): Point
setBorders()
setHeight()
setSide()
setXYasVertex()
showScoutRange()
showSniperRange()
showTankRange()
GUI Initial
main()
GUI
StartButtonClicked()
updateLabel()
updateTable()
DrawingPanel
paintComponent()
updateMoveGUI()
Controller
SoundClip
soundPlay()
attackPlay()
stopAttackSound()
stopPlay()
0..1
1..*
RobotController
AI()
attack()
canMove()
canShoot(): Boolean
check(): String
checkInBoundary(): Boolean
createRobot()
delete()
DirectionToPoint(): Point
getRobotOnTile(): List<Robot>
identify(): Robot
inRange(): Boolean
move()
PointToDirection(): Integer
readFromFile()
resetRobotList()
scan(): Integer
turn()
writeToFile()
1..*
0..1
Interpreter
Arithmetic()
Compare()
Conditional()
DefineWord()
EndLoop()
loop()
Play()
Random()
Read()
StackFunction()
StartLoop()
Tokenize()
Model
Robot
alive(): Boolean
attacked(): Boolean
getAttackPower(): Integer
getColor(): java.awt.Color
getCurrentHp(): Integer
getDamageCaused(): Integer
getDirection(): Integer
getforthCode(): String
getInfo(): String
getisAI(): Boolean
getKills(): Integer
getLocation(): Point
getMaxHp(): Integer
getMoved(): Integer
getMovePoints(): Integer
getName(): String
getRange(): Integer
getTeam(): String
getTilesMoved(): Integer
getType(): List<Robot>
setAlive()
setAttacked()
setCode()
setColor()
setCurrentHp()
setDamageCaused()
setDirection()
setisAI()
setKills()
setLocation()
setMaxHp()
setMoved()
setMovePoints()
setName()
setRange()
setTeam()
setTilesMoved()
Mailbox
addMessage()
removeMessage()
getMessages()
Words
defineWord()
findWord(): String
getDictionary()
removeWord()
Variables
defineVariable()
findVariable(): Object
getDictionary(): Map<Object>
removeVariable()
removeVariable()
setVariable()
valueType(): String

1.2 Final 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 GUI initial and the controller, 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 GUI initial and the robotController through initialization of the rules, the teams, tiles, robots and the gameboard while the robot interpreter and sound class interacts with the controller class through event-based function calls.

Having an overarching independent components architecture with class interactions based on MVC allows us to break up our system in necessary ways. It was necessary to have two separate GUI 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 based script to run our Robot's AI. We still kept the style of interaction 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.

2 Maintainability Information

2.1 Interpreter

This class is a bit tricky to understand, as it is full of stack manipulation and can be difficult to interpret. We have done our best to explain the logic through comments but it is still pretty dense. The gist of it is, the Read function takes in a robot's forth script and information, it then stores that information in in the Words class and feeds the forth script string into Play using a function call with the script as it's parameter. Play then uses the Tokenize function to break the script down into space “ “ delimited tokens that are stored in a stack named CommandStack. This command stack is then read through by Play in order, with different functions being called for different forth words using a case break to implement the functionality of all system defined words while storing values and strings on the Data stack for retrieval by system words and their functions. At this point new words and variables can be defined in the script, they are then stored and retrieved from the Words and Variables model class. One of the words is “play”, which Read then passes back into the Play function to begin the robot's behaviour. To do this, Play tokenizes and interprets the words contained within the robot's“play” word. Once the Command stack is empty, the robot's behaviour is concluded and Play ends.

A difficult to understand part of the interpreter is how loops work in the forth based

robot language. The idea is that when a loop is started by the “do” word, the beginning and end of the loop is stored in the loopCount stack. This stack allows us to have nested loops, as only the top two values of the stack are used until the loop is ended, at which point they are popped off and the inner loop's end and index can now be accessed. While a loop is active (checked by whether or not the loopCount stack is empty or not), words are stored in the LoopStack. When a loop reaches it's end, the words are pushed back onto the Command stack to be interpreted again, creating the desired repeated behaviour.

If a programmer wishes to define a new system word, they can just create a new case break that checks for that word and can then implement whatever behaviour they desire within this case or with a new function that this case will call.

2.2 GUI

If a programmer wishes to update the layout or how the game board and robots are implemented, here is where they will do so. This is the class where you will find how the game is set up using the information from GUI initial, how the current team's stats are displayed, and how turns are handled using the “switch” button.

2.3 Hexmech_pointy

This class is where you will find how the tiles (pointy hexagons) are created, drawn, and populated with the visual representation of our robots. If a programmer wishes to update the graphics of the game board (say, to change to robot's pictures or add animation) which was decidedly simple for our implement ion as our team had little to no experience with graphical systems in Java). This class is easy to understand but an essential companion to the main GUI.

2.4 RobotController

This class has all of the logic used to run the rules of the game and to implement the behaviour of the robots both human and AI. It receives commands from the actionlister and from the Interpreter and handles them using the game logic described in the user manual.

3 How to compile and run our system

The main class of our system is GUI_Initial. All that is required to run our game is to download all the files into an eclipse workspace and compile it as a java application with the GUI_Initial class marked as the main class. We used Eclipse for quick test driven development and bug testing.