**A REPORT**

**On**

**SOS GAME PROJECT**

**By**

**Mthiyane Ntuthuko**

**&**

**Mholi Mncube**

# **Table of Content**

1 Introduction ……………………………………………………………………………………………………………………. 3

* 1. Project Background and Requirements ……………………………………………………………………… 3

2 Project Management ………………………………………………………………………………………………………. 3

2.1 Project Plan ………………………………………………………………………………………………………………. 3

2.2 Team Work ……………………………………………………………………………………………………………….. 3

3 Design …………………………………………………………………………………………………………………………….. 3

3.1 Low-fidelity prototype ………………………………………………………………………………………………. 4

3.2 High-fidelity prototype ……………………………………………………………………………………………… 4

3.3 Final User Interface Design ……………………………………………………………………………………….. 5

3.4 Messaging Protocol …………………………………………………………………………………………………… 5

4 Implementations …………………………………………………………………………………………………………….. 5

5 Testing ……………………………………………………………………………………………………………………………. 6

5.1 Correctness ………………………………………………………………………………………………………......... 6

5.2 User testing ………….…………………………………………………………………………………………………. 6

6 Enhancements ………………………………………………………………………………………….…………………….. 7

7 Future Work ……………………………………………………………………………………………………………………. 7

8 Conclusion ………………………………………………………………………………………………………………………. 8

9 Appendices …………………………………………………………………………………………………………………….. 8

# **1 Introduction**

This documentation was written to provide a few details about what the project was about, the objectives of the project, how the project was managed and how each task was divided. This documentation also includes details of the designs of the project, and how these designs were implemented. It also contains the types of testing that was done in the project, user’s response about it, limitations and other additional enhancements that can be added.

## **1.1 Project Background and Requirements**

The Ten Eleven Games company development team was approached by the company to develop a version of the game called SOS, this is a two player game which is played over the internet. The team was required to develop the SOS Game Client which interacts with the server, enable two users to connect tom the server which allows them to play against each other and also design a suitable Graphical User Interface (GUI) for the game.

# **2 Project management**

## **2.1 Project Plan**

A detailed project plan indicating when tasks will get done and who will complete them was drawn at the beginning. Each deliverable was broken down into a number of detailed task then each team member was given their specific responsibility/ task and when that specific task had to be completed, these tasks of each member were estimated to take about one to two days as most of them were more of research like. The times at which the team would meet were also set beforehand. After all the responsibilities were split up, they are jotted down into a scheduling table detailing the tasks. The team did not just stick to one plan, as a Scrum like methodology was used, a few changes were allowed to be made on the plan.

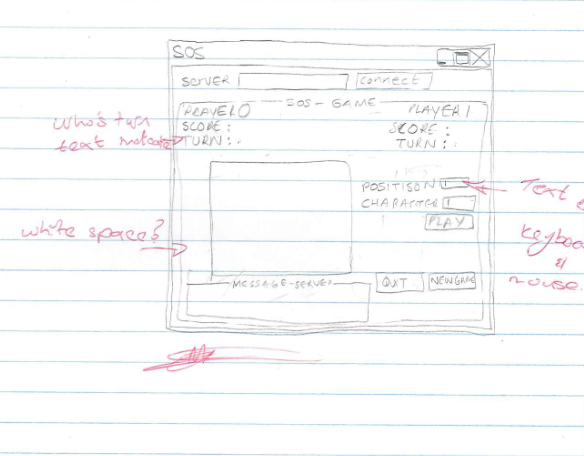
## **2.2 Team work**

Both Extreme Programming (XP) and Scrum agile methodologies were used in this project. The team met at the beginning of the week for Sprint Planning Meetings, afterward each member would go out to do their particular research. The team would meet again two days later to discuss what each member has found on their research then it was decided how it would be implemented. When it came to coding, the Extreme Programming (XP) methodology was used. One member would be the Driver (Would write up the code) and the other member be the Navigator (Look at the code as it’s written and question if and provide alternative ways). The roles were alternated one weekly basis, this week this member would be the driver and the following week another member would be the driver. All of the code was written in the presence of both members, this was done so both team members would understand code well and that both members would be about to learn from each other.

# **3 Design**

## **3.1 Low-fidelity Prototype**

The low-fidelity prototype was the first design the game’s graphical user interface that was put together by the team. It was written on paper just to give a basic idea of how the user interface would look like. It was then taken to the team leader to assess it and add a few comments on it (texts in red are the team leader’s comments)



## **3.2 High-Fidelity Prototype**

The high-fidelity prototype is just the same as the low-fidelity prototype the only difference is that it is now coded and a few enhancements have been added and a few changes have been made to it. Python and PyQt4 was used to code up this design including the final design.



## **3.3 Final User Interface Design**

This is the final graphical user interface design of the game, it is the end product of the high-fidelity prototype. A few changes were made on the high-fidelity prototype based on user’s response to it, these changes were made to make the game more user friendly and appealing.

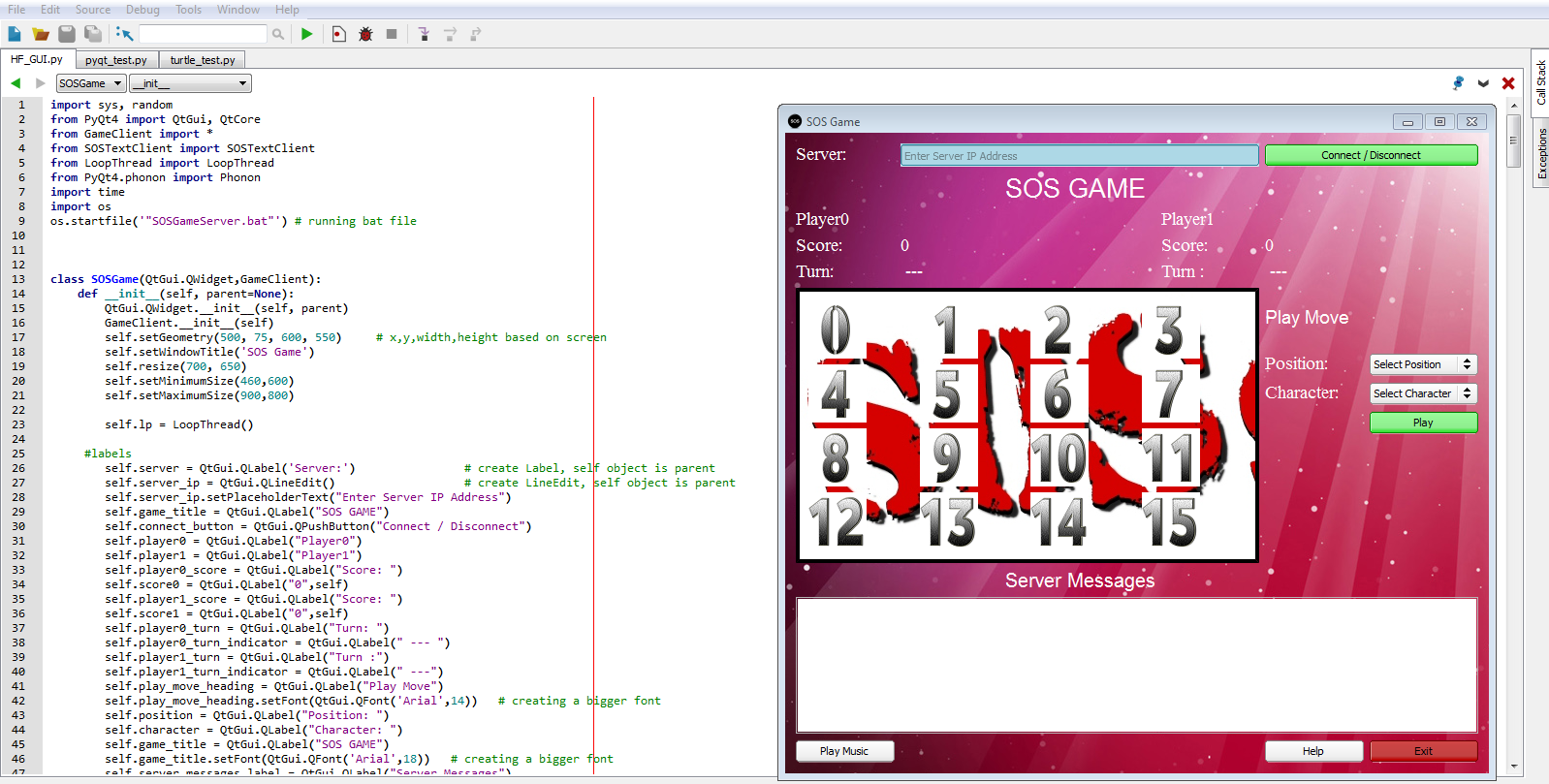


## **3.4 Messaging Protocol**

The team had to create a game client and a thread which connected to a game server that had already been designed for them. The thread is similar to a play loop, and it connected the graphical user interface and the game server. The game client was responsible for sending messages (user inputs) to the server which evaluated these inputs then sent a response back to the thread, the thread then sent the messages from the server to the game client, basically the thread acted as a middle man between the game client and game server. The game client had the responsibility of sending the appropriate server messages (when it’s a player’s turn to play, invalid moves made, etc.) to the user in a more user friendly manner. The game server was also responsible for connecting the machines of the players (users) playing at that particular time and sending the appropriate messages to each user at appropriate times. Before the server connects to the machines one user has to first run a bat file than both of then the other has to enter the opponents Internet Protocol (IP) Address so the game client can connect to the game server.

# **4 Implementations**

The team then implemented the design into developing the system, all the ideas were put together and the code for the system was written. The first version of the GUI is the high-fidelity prototype above, it was then changed numerous times based on users feed-back. This was done to make the system more user friendly and appealing. The final code for the system is attached below (left of the screenshot) and on the right of the screenshot is the final GUI of the system which is the end product of the code. The game can run on operation systems Windows 7 and above (later versions) and doesn’t run on a lot of memory, it can run on most machines.



# **5 Testing**

## **5.1 Correctness**

This section of documentation focuses on the methods that were used for rectification of errors that might have been in counted during the coding of the software, what scenarios were used, why they were chosen and what were the results.

The correctness process is a 5 step procedure

• Detect the error

• Locate the error

• Design ways to repair

• Repair the error in system

• Re-test the system

## **5.2 User Testing**

Focuses on how well end-users were able to communicate with the system, i.e. is making use of it and if they in counted any problems what were those problems.

There is no better way to test a system other than using it. So we had two end-users connect on different computers and execute the system, since it was able to connect over the local area network. First step is to connect to the localhost network which is the LAN that enables the system to communicate to other system on the other machine. Subsequently the system once connected displays messages to the users on the system so users follow these messages as instructions and then are able to use the system.

Once the users finished using the system. There was a survey for each of the end-users to fill out and give feedback on the system, usability etc. Once the surveys are completed the survey is later analysed taken into consideration and implemented on the system.

Some of the common problems that came across the both users was the fact that they didn’t understand the user interface of the command prompt, MS-DOS it’s a very unique user interface and not easy to familiarize yourself with.

The system final graphical user interface was able to eradicate most of the confusion end-users had.

# **6 Enhancements**

This is where the innovation and creativity is displayed, features added to the existing software to monopolise the user and make their experience with the software memorable.

On the system an assistance button was added as an enhancement to help assists the user step by step to get the system running well, then also use it. The feature guides the user so they don’t waste time trying figure out how to use it. The other enhancement was leisure, a music option was added to have your own selected song playing in the background while you use the system to give full experience of the system.

# **7 Future Work**

New version of the software not yet implemented but focus on making the software more easy to use. Such as voice command to take actions and voice prompt for certain instructions.

The new version of the system shall have voice command instructions so the user does not have to reach most of the time but listen to the command the system gives to use the system. Other enhancement that can still be added are a celebration animation to the winner to indicate success and a pleasant way to tell the other end-user they have lost. Also a database to save highest obtained score any other who wishes to use the system they know what score they need to achieve to be on the top of the score board.

# **8 Conclusion**

This report was written to give detailed information of the processes that the team went through in designing and developing the system. Also to give the limitations and specifications of the system as well as its advantages. The report is for the company to use it as a reference if at some later stage it decides to develop an enhanced version of the SOS game, with a different development team, or takes on a similar project.

# **9 Appendices**

This section focuses on the code of the system, basically the structure of the code of the system. The code below is the system itself.

import sys, random

from PyQt4 import QtGui, QtCore

from GameClient import \*

from SOSTextClient import SOSTextClient

from LoopThread import LoopThread

from PyQt4.phonon import Phonon

import time

import os

os.startfile('"SOSGameServer.bat"') # running bat file

class SOSGame(QtGui.QWidget,GameClient):

def \_\_init\_\_(self, parent=None):

QtGui.QWidget.\_\_init\_\_(self, parent)

GameClient.\_\_init\_\_(self)

self.setGeometry(500, 75, 600, 550) # x,y,width,height based on screen

self.setWindowTitle('SOS Game')

self.resize(700, 650)

self.setMinimumSize(460,600)

self.setMaximumSize(900,800)

self.lp = LoopThread()

#labels

self.server = QtGui.QLabel('Server:') # create Label, self object is parent

self.server\_ip = QtGui.QLineEdit() # create LineEdit, self object is parent

self.server\_ip.setPlaceholderText("Enter Server IP Address")

self.game\_title = QtGui.QLabel("SOS GAME")

self.connect\_button = QtGui.QPushButton("Connect / Disconnect")

self.player0 = QtGui.QLabel("Player0")

self.player1 = QtGui.QLabel("Player1")

self.player0\_score = QtGui.QLabel("Score: ")

self.score0 = QtGui.QLabel("0",self)

self.player1\_score = QtGui.QLabel("Score: ")

self.score1 = QtGui.QLabel("0",self)

self.player0\_turn = QtGui.QLabel("Turn: ")

self.player0\_turn\_indicator = QtGui.QLabel(" --- ")

self.player1\_turn = QtGui.QLabel("Turn :")

self.player1\_turn\_indicator = QtGui.QLabel(" ---")

self.play\_move\_heading = QtGui.QLabel("Play Move")

self.play\_move\_heading.setFont(QtGui.QFont('Arial',14)) # creating a bigger font

self.position = QtGui.QLabel("Position: ")

self.character = QtGui.QLabel("Character: ")

self.game\_title = QtGui.QLabel("SOS GAME")

self.game\_title.setFont(QtGui.QFont('Arial',18)) # creating a bigger font

self.server\_messages\_label = QtGui.QLabel("Server Messages")

self.server\_messages\_label.setFont(QtGui.QFont('Arial',15)) # creating a bigger font

#Combo Boxes

self.position\_list = QtGui.QComboBox()

self.character\_list = QtGui.QComboBox()

self.position\_list.addItems(["Select Position","0","1","2","3","4","5","6","7","8","9","10","11","12","13","14","15"]) # Adding numbers onto the position combo\_box

self.character\_list.addItems(["Select Character","S","O"]) # Adding characters onto the character combo\_box

#Button

self.play\_button = QtGui.QPushButton("Play")

self.exit\_button = QtGui.QPushButton("Exit")

self.instructions = QtGui.QPushButton("Help")

self.music\_button = QtGui.QPushButton('Play Music')

# Pictures of numbers on the grid

self.no\_0 = QtGui.QPixmap('numbers/0.png')

self.no\_1 = QtGui.QPixmap('numbers/1.png')

self.no\_2 = QtGui.QPixmap('numbers/2.png')

self.no\_3 = QtGui.QPixmap('numbers/3.png')

self.no\_4 = QtGui.QPixmap('numbers/4.png')

self.no\_5 = QtGui.QPixmap('numbers/5.png')

self.no\_6 = QtGui.QPixmap('numbers/6.png')

self.no\_7 = QtGui.QPixmap('numbers/7.png')

self.no\_8 = QtGui.QPixmap('numbers/8.png')

self.no\_9 = QtGui.QPixmap('numbers/9.png')

self.no\_10 = QtGui.QPixmap('numbers/10.png')

self.no\_11 = QtGui.QPixmap('numbers/11.png')

self.no\_12 = QtGui.QPixmap('numbers/12.png')

self.no\_13 = QtGui.QPixmap('numbers/13.png')

self.no\_14 = QtGui.QPixmap('numbers/14.png')

self.no\_15 = QtGui.QPixmap('numbers/15.png')

# Pictures of numbers labels

self.no\_0\_label = QtGui.QLabel()

self.no\_1\_label = QtGui.QLabel()

self.no\_2\_label = QtGui.QLabel()

self.no\_3\_label = QtGui.QLabel()

self.no\_4\_label = QtGui.QLabel()

self.no\_5\_label = QtGui.QLabel()

self.no\_6\_label = QtGui.QLabel()

self.no\_7\_label = QtGui.QLabel()

self.no\_8\_label = QtGui.QLabel()

self.no\_9\_label = QtGui.QLabel()

self.no\_10\_label = QtGui.QLabel()

self.no\_11\_label = QtGui.QLabel()

self.no\_12\_label = QtGui.QLabel()

self.no\_13\_label = QtGui.QLabel()

self.no\_14\_label = QtGui.QLabel()

self.no\_15\_label = QtGui.QLabel()

self.no\_0\_label.setPixmap(self.no\_0)

self.no\_1\_label.setPixmap(self.no\_1)

self.no\_2\_label.setPixmap(self.no\_2)

self.no\_3\_label.setPixmap(self.no\_3)

self.no\_4\_label.setPixmap(self.no\_4)

self.no\_5\_label.setPixmap(self.no\_5)

self.no\_6\_label.setPixmap(self.no\_6)

self.no\_7\_label.setPixmap(self.no\_7)

self.no\_8\_label.setPixmap(self.no\_8)

self.no\_9\_label.setPixmap(self.no\_9)

self.no\_10\_label.setPixmap(self.no\_10)

self.no\_11\_label.setPixmap(self.no\_11)

self.no\_12\_label.setPixmap(self.no\_12)

self.no\_13\_label.setPixmap(self.no\_13)

self.no\_14\_label.setPixmap(self.no\_14)

self.no\_15\_label.setPixmap(self.no\_15)

# Adding pictures on the grid

self.board\_grid = QtGui.QGridLayout()

self.board\_grid.addWidget(self.no\_0\_label,0,0)

self.board\_grid.addWidget(self.no\_1\_label,0,1)

self.board\_grid.addWidget(self.no\_2\_label,0,2)

self.board\_grid.addWidget(self.no\_3\_label,0,3)

self.board\_grid.addWidget(self.no\_4\_label,1,0)

self.board\_grid.addWidget(self.no\_5\_label,1,1)

self.board\_grid.addWidget(self.no\_6\_label,1,2)

self.board\_grid.addWidget(self.no\_7\_label,1,3)

self.board\_grid.addWidget(self.no\_8\_label,2,0)

self.board\_grid.addWidget(self.no\_9\_label,2,1)

self.board\_grid.addWidget(self.no\_10\_label,2,2)

self.board\_grid.addWidget(self.no\_11\_label,2,3)

self.board\_grid.addWidget(self.no\_12\_label,3,0)

self.board\_grid.addWidget(self.no\_13\_label,3,1)

self.board\_grid.addWidget(self.no\_14\_label,3,2)

self.board\_grid.addWidget(self.no\_15\_label,3,3)

self.groupBox = QtGui.QGroupBox()

self.groupBox.setLayout(self.board\_grid) # Adding the boarder around

self.groupBox.setStyleSheet("QGroupBox {background-color: lightblue padding 6px; border:4px solid; border-style: inset; background-image: url(images/sos\_b.jpg); width}")

#music player features

self.media = Phonon.MediaObject(self)

self.media.stateChanged.connect(self.handleStateChanged)

self.audio = Phonon.AudioOutput(Phonon.VideoCategory, self)

Phonon.createPath(self.media, self.audio)

self.music\_button.clicked.connect(self.stopButton)

self.sources = []

# Text\_Boxes

self.server\_messages = QtGui.QListWidget(self)

# Main Layout

self.grid = QtGui.QGridLayout()

self.grid.addWidget(self.server,0,0)

self.grid.addWidget(self.server\_ip,0,1,1,3) # setting position and size

self.grid.addWidget(self.connect\_button,0,4,1,2)

self.grid.addWidget(self.game\_title,1,2)

self.grid.addWidget(self.player0,2,0)

self.grid.addWidget(self.player1,2,3)

self.grid.addWidget(self.player0\_score,3,0)

self.grid.addWidget(self.score0,3,1)

self.grid.addWidget(self.player1\_score,3,3)

self.grid.addWidget(self.score1,3,4)

self.grid.addWidget(self.player0\_turn,4,0)

self.grid.addWidget(self.player0\_turn\_indicator,4,1)

self.grid.addWidget(self.player1\_turn,4,3)

self.grid.addWidget(self.player1\_turn\_indicator,4,4)

self.grid.addWidget(self.groupBox,5,0,6,4)

self.grid.addWidget(self.play\_move\_heading,5,4)

self.grid.addWidget(self.position,6,4)

self.grid.addWidget(self.position\_list,6,5)

self.grid.addWidget(self.character,7,4)

self.grid.addWidget(self.character\_list,7,5)

self.grid.addWidget(self.play\_button,8,5)

self.grid.addWidget(self.server\_messages\_label,11,2)

self.grid.addWidget(self.server\_messages,12,0,1,6)

self.grid.addWidget(self.instructions,13,4)

self.grid.addWidget(self.exit\_button,13,5)

self.grid.addWidget(self.music\_button,13,0)

self.setLayout(self.grid)

#Customization of the gui

#Changing labels fonts

self.server.setFont(QtGui.QFont('Times',14,3)) # create Label, self object is parent

self.player0.setFont(QtGui.QFont('Times',14,3))

self.player1.setFont(QtGui.QFont('Times',14,3))

self.player0\_score.setFont(QtGui.QFont('Times',14,3))

self.player1\_score.setFont(QtGui.QFont('Times',14,3))

self.score0.setFont(QtGui.QFont('Times',14,3))

self.score1.setFont(QtGui.QFont('Times',14,3))

self.player0\_turn.setFont(QtGui.QFont('Times',14,3))

self.player1\_turn.setFont(QtGui.QFont('Times',14,3))

self.player0\_turn\_indicator.setFont(QtGui.QFont('Times',14,3))

self.player1\_turn\_indicator.setFont(QtGui.QFont('Times',14,3))

self.play\_move\_heading.setFont(QtGui.QFont('Arial',14)) # creating a bigger font

self.position.setFont(QtGui.QFont('Times',14,3))

self.character.setFont(QtGui.QFont('Times',14,3))

self.game\_title.setFont(QtGui.QFont('Arial',20,5)) # creating a bigger font

self.server\_messages\_label.setFont(QtGui.QFont('Arial',15)) # creating a bigger font

# Changing labels colour and message boxes

self.server.setStyleSheet("QLabel { color: white;}")

self.player0.setStyleSheet("QLabel { color: white;}")

self.player1.setStyleSheet("QLabel { color: white;}")

self.player0\_score.setStyleSheet("QLabel { color: white;}")

self.player1\_score.setStyleSheet("QLabel { color: white;}")

self.score0.setStyleSheet("QLabel { color: white;}")

self.score1.setStyleSheet("QLabel { color: white;}")

self.player0\_turn.setStyleSheet("QLabel { color: white;}")

self.player1\_turn.setStyleSheet("QLabel { color: white;}")

self.player0\_turn\_indicator.setStyleSheet("QLabel { color: white;}")

self.player1\_turn\_indicator.setStyleSheet("QLabel { color: white;}")

self.play\_move\_heading.setStyleSheet("QLabel { color: white;}")

self.position.setStyleSheet("QLabel { color: white;}")

self.character.setStyleSheet("QLabel { color: white;}")

self.game\_title.setStyleSheet("QLabel { color: white;}")

self.server\_messages\_label.setStyleSheet("QLabel { color: white;}")

self.server\_messages.setStyleSheet("QTextEdit { color: rgb(50, 50, 50); font-size: 11px; background-color: lightblue; }")

self.server\_ip.setStyleSheet("QLineEdit { color: rgb(50, 50, 50); font-size: 11px; background-color: lightblue;}")

#Styling 1 comboboxs

QtGui.QApplication.setStyle(QtGui.QStyleFactory.create('cleanlooks')) #this changes the theme from windows 8 to more smooth good looking comboboxs

#backgroundColor and picture

self.picture = QtGui.QPalette(self)

self.picture.setBrush(QtGui.QPalette.Background,QtGui.QBrush(QtGui.QPixmap('images/inclined-lines-abstract-wide-wallpaper-1600x900-004.jpg')))

self.setPalette(self.picture)

#setting up window icon

self.setWindowIcon(QtGui.QIcon('images/sos1.png'))

#pushbuttons color

self.connect\_button.setStyleSheet("QPushButton { background-color: lightgreen;}") #green button indication to proceed

self.play\_button.setStyleSheet("QPushButton { background-color: lightgreen;}")

self.exit\_button.setStyleSheet("QPushButton { background-color: #c34242;}") #red button to indicate stop or exit

#connecting buttons to user interface

self.connect(self.connect\_button,QtCore.SIGNAL('clicked()'),self.connect\_server)

self.connect(self.play\_button,QtCore.SIGNAL('clicked()'),self.play\_move)

self.connect(self.exit\_button,QtCore.SIGNAL('clicked()'),self.exit\_game)

self.connect(self.instructions,QtCore.SIGNAL('clicked()'),self.instruction\_clicked)

self.lp = LoopThread() # create thread

self.lp.update\_label\_signal.connect(self.loop\_thread\_slot) # connect signals to slots

def connect\_server(self):

self.server\_messages.addItem('Trying to connect, please wait.')

self.server\_ip = self.server\_ip.displayText()

ip = self.server\_ip #this is used to make name of ip to be accessible to both gui and thread class

self.lp.connect(ip)

self.lp.start() # thread started

def loop\_thread\_slot(self, msg): # slot which handles signal from thread

if msg[0:8] == 'new game':

self.server\_messages.clear() # clears the message board when new game is played

self.score0.setText("0") # resetting the score when a new game is started

self.score1.setText("0")

self.player = msg[9:10] # getting which player is player one and two

self.server\_messages.addItem("New game has been started, you are player {}.".format(msg[9:10]))

# Resetting the board for a new game

self.no\_0\_label.setPixmap(self.no\_0)

self.no\_1\_label.setPixmap(self.no\_1)

self.no\_2\_label.setPixmap(self.no\_2)

self.no\_3\_label.setPixmap(self.no\_3)

self.no\_4\_label.setPixmap(self.no\_4)

self.no\_5\_label.setPixmap(self.no\_5)

self.no\_6\_label.setPixmap(self.no\_6)

self.no\_7\_label.setPixmap(self.no\_7)

self.no\_8\_label.setPixmap(self.no\_8)

self.no\_9\_label.setPixmap(self.no\_9)

self.no\_10\_label.setPixmap(self.no\_10)

self.no\_11\_label.setPixmap(self.no\_11)

self.no\_12\_label.setPixmap(self.no\_12)

self.no\_13\_label.setPixmap(self.no\_13)

self.no\_14\_label.setPixmap(self.no\_14)

self.no\_15\_label.setPixmap(self.no\_15)

elif msg[:] =='your move':

self.message = msg

# Indicating who's turn is it to play

if self.player == '0':

self.player0\_turn\_indicator.setText("Your Move")

self.player1\_turn\_indicator.setText("Wait")

else:

self.player0\_turn\_indicator.setText("Wait")

self.player1\_turn\_indicator.setText("Your Move")

elif msg[:] == 'opponents move':

# Indicating who's turn is it to play

if self.player == '0':

self.player0\_turn\_indicator.setText("Wait")

self.player1\_turn\_indicator.setText("Your Move")

else:

self.player0\_turn\_indicator.setText("Your Move")

self.player1\_turn\_indicator.setText("Wait")

elif msg[0:5] =='valid': # If the move is valid do the following

selected\_move\_list = msg.split(',')

position = selected\_move\_list[1]

character = QtGui.QPixmap('numbers/' + selected\_move\_list[2] + '.png')

self.Player0\_score = selected\_move\_list[3]

self.Player1\_score = selected\_move\_list[4]

# Updating the scores

self.score0.setText(self.Player0\_score)

self.score1.setText(self.Player1\_score)

# if the users chooses these positin replace the number with the chosen character

if position == '0':

self.no\_0\_label.setPixmap(character)

elif position =='1':

self.no\_1\_label.setPixmap(character)

elif position =='2':

self.no\_2\_label.setPixmap(character)

elif position =='3':

self.no\_3\_label.setPixmap(character)

elif position =='4':

self.no\_4\_label.setPixmap(character)

elif position =='5':

self.no\_5\_label.setPixmap(character)

elif position =='6':

self.no\_6\_label.setPixmap(character)

elif position =='7':

self.no\_7\_label.setPixmap(character)

elif position =='8':

self.no\_8\_label.setPixmap(character)

elif position =='9':

self.no\_9\_label.setPixmap(character)

elif position =='10':

self.no\_10\_label.setPixmap(character)

elif position =='11':

self.no\_11\_label.setPixmap(character)

elif position =='12':

self.no\_12\_label.setPixmap(character)

elif position =='13':

self.no\_13\_label.setPixmap(character)

elif position=='14':

self.no\_14\_label.setPixmap(character)

elif position =='15':

self.no\_15\_label.setPixmap(character)

elif msg == 'invalid move': # If the selected move is invalid

self.server\_messages.addItem("Invalid move, try agian.")

elif msg[0:9] == 'game over': # This is when the board is full

winner\_list = msg.split(',') # Spliting the message into a list

winner = winner\_list[1] # Game winner

p0\_final\_score = winner\_list[2]

p1\_final\_score = winner\_list[3] # Final scores

if p0\_final\_score == p1\_final\_score:

QtGui.QMessageBox.information(self, "Game over","Winner: Drawn Battle (Level Score) \nPlayer 0 Final Score: {} \nPlayer 1 Final Score: {}".format(p0\_final\_score,p1\_final\_score))

else:

QtGui.QMessageBox.information(self, "Game over","Winner: Player {} \nPlayer 0 Final Score: {} \nPlayer 1 Final Score: {}".format(winner,p0\_final\_score,p1\_final\_score))

elif msg == 'play again':

reply = QtGui.QMessageBox.question(None,'Play Again','Do you want to play again?',QtGui.QMessageBox.Yes, QtGui.QMessageBox.No)

if reply == QtGui.QMessageBox.Yes:

x = 'y'

else:

x = 'n'

self.lp.move(x)

elif msg == 'exit game':

self.server\_messages.addItem("The game has been ended either by you or the opponent.")

def instruction\_clicked(self): # Game instructions

instruction = QtGui.QMessageBox.information(self,"Instructions",'Connection \n \n 1.Connect to the server by entering your oppenent\'s IP ADDRESS then press the Connect/Disconnect button.\n2.Check the Server Message box to see if you are connected.\n3.Look for your player number on the Server Messages box below..\n4.If you want to quit game,press the exit button. \n \n How The Game Works \n \nPlayers take turns to add either an "S" or an "O" to any square, with no requirement to use the same letter each turn. The object of the game is for each player to attempt to create the straight sequence S-O-S among connected squares (either diagonally, horizontally, or vertically), and to create as many such sequences as they can. If a player succeeds in creating an SOS, that player immediately takes another turn, and continues to do so until no SOS can be created on their turn. Otherwise turns alternate between players after each move.')

def exit\_game(self):

# adding buttons to ask user if they want to quit or not

reply = QtGui.QMessageBox.question(None,'WARNING !!','Are you sure you want exit?',QtGui.QMessageBox.Yes, QtGui.QMessageBox.No)

if reply == QtGui.QMessageBox.Yes:

self.server\_messages.addItem("The oppenent has exited the game, run the game again to play new opponent")

self.close()

def play\_move(self):

x = self.position\_list.currentText() + ',' + self.character\_list.currentText() # position and character

try:

sms = self.position\_list.currentText() + ',' + self.character\_list.currentText()

if self.message == 'your move':

self.lp.move(sms) # sending to the loop to send to the thread

else:

self.server\_messages.addItem('The move cannot be played because it is not your move yet, Please wait.')

except:

self.server\_messages.addItem('Error: The game is not connected, first connect to server.') # Error if the users players before the connection is established

def addFiles(self):

files = QtGui.QFileDialog.getOpenFileNames(self, "Select Music Files",

QtGui.QDesktopServices.storageLocation(QtGui.QDesktopServices.MusicLocation))

if not files:

return

index = len(self.sources)

for string in files:

self.sources.append(Phonon.MediaSource(string))

if self.sources:

self.metaInformationResolver.setCurrentSource(self.sources[index])

def stopButton(self):

if self.media.state() == Phonon.PlayingState: #This is to keep track of media state

self.media.stop() #stop any media if being executed

else:

path = QtGui.QFileDialog.getOpenFileName(self, self.music\_button.text()) #opens directry on local disk for person to select a file

if path:

self.media.setCurrentSource(Phonon.MediaSource(path)) # sets the media file selected so it is executed

self.media.play() #executes the media file

def handleStateChanged(self, newstate, oldstate):

if newstate == Phonon.PlayingState:

self.music\_button.setText('Stop') #if the play file button is clicked it changes state to allow user to stop media being played

elif (newstate != Phonon.LoadingState and

newstate != Phonon.BufferingState):

self.music\_button.setText('Choose File')

if newstate == Phonon.ErrorState:

source = self.media.currentSource().fileName()

print ('ERROR: could not play:', source.toLocal8Bit().data())

print (' %s' % self.media.errorString().toLocal8Bit().data())

app = QtGui.QApplication(sys.argv)

abs\_widget = SOSGame()

abs\_widget.show()

sys.exit(app.exec\_())