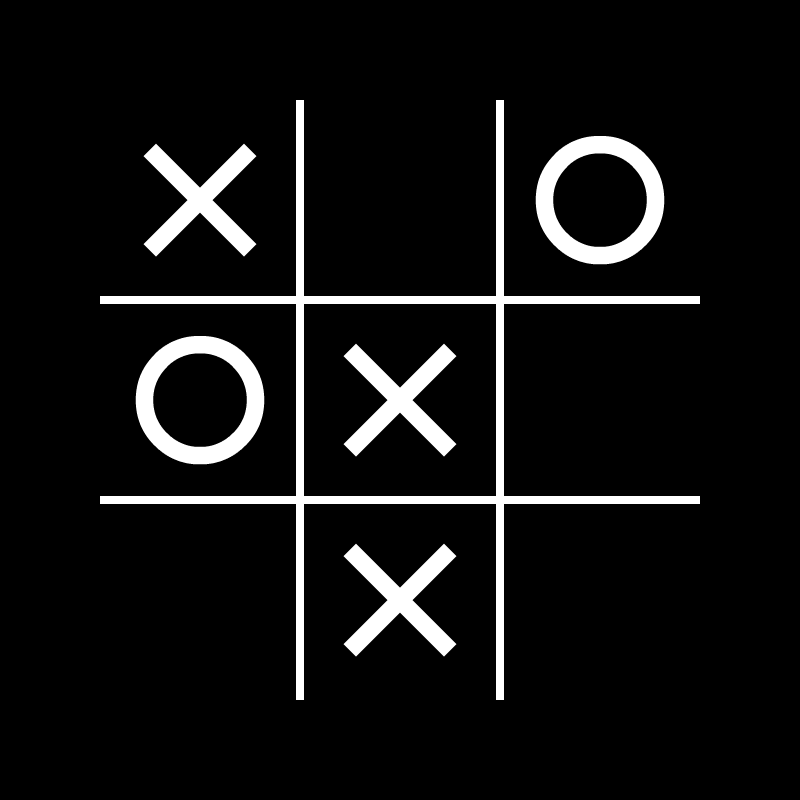
Madikizela Will | Computer Science | June 22, 2020

Final Report

oxo game



**Table of Content**

[Introduction](#introduction1)…………………………………………………………………………………

[Project Management](#ProjectManagement1)…………………………………………………………………….

[System Design](#SystemDesign1)………………………………………………………………………………

[Implementation](#Implementation1)……………………………………………………………………………

[Testing](#Testing1)………………………………………………………………………………………….

[System Correctness](#SystemCorrectness1)………………………………………………………………………

[Enhancements](#Enhancements1)……………………………………………………………………………..

[Future Work](#FutureWork1)………………………………………………………………………………..

[Conclusion](#Conclusion1)…………………………………………………………………………………..

[Appendices](#Appendices1)………………………………………………………………………………….

**Introduction**

As a developer in the game development company known as Ten Eleven Game. I was approached by the company to develop a game called OXO Game, also known as tic tac toe game.

The game was expected to allow two users to connect to the server and allocate each player a character randomly i.e. O and X. Design the program-user Interface that is easy to use and understand, also state the interface conversions to the user at the beginning of the game and display the board. Should produce valid moves only and report invalid moves. The two players take turns putting their characters in the 3×3 grid board. The player who first gets three characters in a row (vertically, horizontally or diagonally) wins the game, and the other player loses the game.

I was required to develop a python-based program that will allow users to connect to the server and get their characters. The communication messages from the server to the player display to the screen while the player is playing.

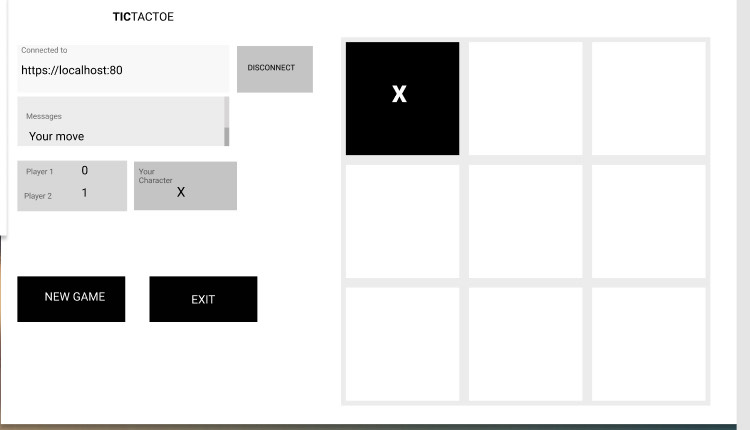
**Project Management**

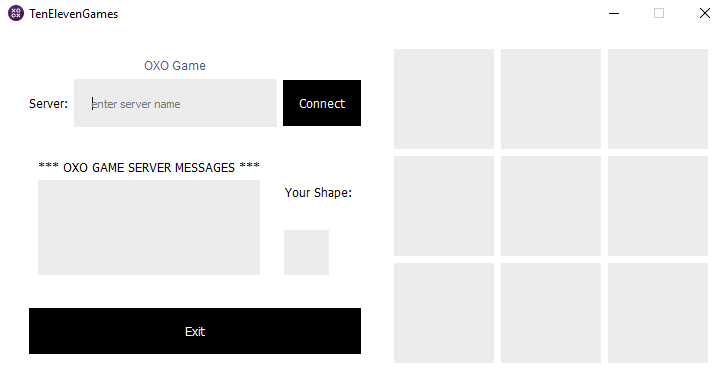
Work Plan

The project’s work was broken down into 6 weekly assignments. The Project plan was created that servers as the guideline on creating timelines and establishing deadlines for each phase of the project. The schedule plan that was designed in the project plan was used to factor different times-related and provide a good idea on what to be done next. That gave me a lot of time to plan and submit every work that was required in time.

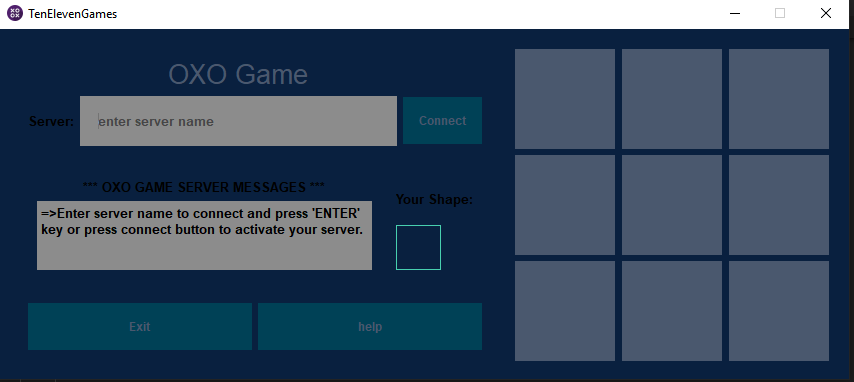
**System Design**

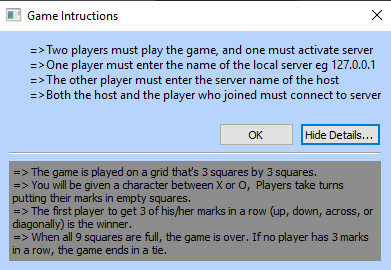
Low Fidelity Prototype – the first idea I had when designing the GUI of the game

* This was designed by [*figma*](https://www.figma.com/) (it’s not a functioning program)

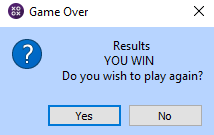
High Fidelity Prototype 

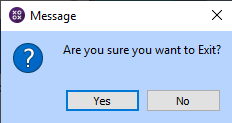
Final Design – with all the implementations made to the game



**Implementation**

I have implemented the help button that will allow users not familiar with how to connect to the server, get assistance, and guide them on how to play the game with the game rules as well. This is the message that will pop up when the help button is clicked.

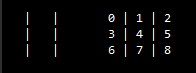
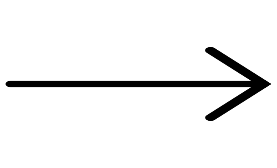
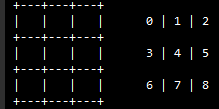
This is the pop-up message that indicates the results stating the winner or loser or weather it’s a tie.

When the player exits the game, pressing the exit button, a pop-up message will appear asking the user if they are sure to exit the game. This saves a lot of work avoiding mistakes and unexpected misuse of the buttons.

**Testing**

**Text Based Client**

One of the major concepts as a developer is to satisfy the user or the client, since we are developing a system for the company, we need to evaluate our program before we procced. On the text-based client, the major problem was the board design, the program was working as expected without errors. To solve the problem, I added borders (outer lines) to make it clearer.

**GUI Prototype**

In design, although there are several factors that have a significant influence on the outcome, colors play a major function to attract users. Color is the easiest and the most important aspect of engaging the user with the product in design. To overcome the problem, I changed the color of gui from white to dim blue. The user interface was well laid out and everything was easy to access and placed in an appropriate position.

**System Correctness**

The system was tested in many ways to make the program bug free. Different scenarios were, mostly major scenarios that could course major problems in the program, make it to crush or have logical errors that will not be pleasant for the user.

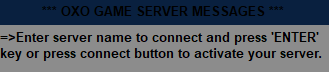
**Enter wrong server name** – the program responded accordingly by displaying the error message in the Text screen.

**Press the board play button before connecting to the server** – before connecting to server and getting your icon, the program does not allow the user to press the board play button (the buttons are disabled)

**Pressed connect button multiple times** – after connecting to server the program disables the connect button, disallowing the users to press the button multiple times

**Edit the text edit that displays messages from the server** – the Text box could not be edited, at all.

**Enhancements**

Connect the keyboard to the game – I have linked the connect button to the keyboard, “ENTER” button. Displayed a line of instruction to the user that will assist the user understand how to use it.

Disable the play board buttons – when it’s the opponent’s turn to play.

Play sound – the program can play sound when you place the character to the game board button, also play different sounds when the player WINS or LOSES.

**Future Work**

Artificial Intelligence can be used for this game in order to play only single player, in other words the server that was given should be able to allow the user to play against the computer.

Enable users to change the color of the game to the color of their needs.

Collect the score when players play multiple times

**Conclusion**

In the conclusion of the project, I would like to say that Python is a fun programming language and while creating a project like this, it has not just been a good experience, but it also helped in the development of my creativity and logical thinking and also enhanced my programming skills, also gave me the taste of the really world problems. I would be more than happy to work on the other projects in Python because it’s just amazing to work with Python. The program is working, and it’s also bug-free. The Project was handled nice and smooth although it was super challenging, I managed to do all the tusks in time.

**Appendices**

OXO\_Game.py

1. # main gui class
2. # Will Madikizela
4. **import** sys
5. **from** PyQt5.Qt **import** Qt
6. **from** PyQt5.QtWidgets **import**\*
7. **from** PyQt5.QtCore **import**\*
8. **from** PyQt5.QtGui **import**\*
9. **from** time **import**\*
10. **from** PyQt5.QtMultimedia **import**\*
11. **from** GameClient **import**\*
13. **class** LoopThread(QThread):
14. # create a signal
15. msg\_signal = pyqtSignal(str)
16. **def** \_\_init\_\_ (self):
17. QThread.\_\_init\_\_(self)
18. **def** run(self):
19. **while** True:
20. msg = OXO.receive\_message()
21. # emit signal
22. **if** len(msg):self.msg\_signal.emit(msg)
24. **class** OxoGame(QWidget, GameClient): # inherits from QWidgets and gameclient
25. # parent difines parent widget
26. **def** \_\_init\_\_ (self, parent = None):
27. # super class constructor
28. QWidget.\_\_init\_\_(self, parent)
29. GameClient.\_\_init\_\_(self)
30. # window name
31. self.setWindowTitle("TenElevenGames")
32. # backgroung color of the window
33. self.setPalette(QPalette(QColor("#09203F")))
34. # x, y, widgth and height
35. self.setGeometry(270, 117, 500, 350)
36. # game icon
37. self.setWindowIcon(QIcon("gameIcon.png"))
38. # Disable maximizing window
39. self.setFixedSize(850, 350)
40. # default font
41. self.setFont(QFont("arial", 10, weight = QFont.Bold))

44. self.sounds = dict(play=QSound("play.wav"),
45. win=QSound("win.wav"),
46. lose=QSound("lose.wav"))
48. # Declare variables
49. # heading label
50. self.heading  = QLabel("OXO Game")
51. # set font of the heading label
52. self.heading.setFont(QFont("Simplifica", 20, 10))
53. # set color of the heading label to greyish blue
54. self.heading.setStyleSheet("color: #4A586E ")
55. # set the heading label to center
56. self.heading.setAlignment(Qt.AlignCenter)
58. # server label
59. self.server\_input\_label = QLabel("Server:")
60. # server input
61. self.server\_url\_input = QLineEdit()
62. # reurn pressed to allow the user to press 'Enter' key to connect
63. self.server\_url\_input.returnPressed.connect(self.connect)
64. # allow a button to clear all the text at once
65. self.server\_url\_input.setClearButtonEnabled(True)
66. # place holder in the input for server for more intructions
67. self.server\_url\_input.setPlaceholderText("enter server name")
68. # button to connect to server
69. self.connect\_button = QPushButton("Connect")
70. # set Object Name for connect name
71. self.connect\_button.setObjectName("connect")
72. # connect the button to report action when pressed
73. self.connect\_button.clicked.connect(self.connect)

76. # labels indicating messages from server
77. self.messages\_from\_server\_label = QLabel("\*\*\*OXO GAME SERVER MESSAGES\*\*\*")
78. # align the message label to centre
79. self.messages\_from\_server\_label.setAlignment(Qt.AlignCenter)
80. # messages from the server
81. self.messages\_from\_server = QTextEdit("")
82. # disable editing the textedit
83. self.messages\_from\_server.setReadOnly(True)
85. # label indicating the players character
86. self.character\_label = QLabel("Your Shape:")
87. # align the label to centre
88. self.character\_label.setAlignment(Qt.AlignCenter)
90. # character of the player
91. # x character
92. self.o = QIcon("nought.png")
93. # o character
94. self.x = QIcon("cross.png")
95. # blank character
96. self.null = QIcon("blank.gif")
98. # the button to place the character
99. self.character = QPushButton()
100. # oject name to specify the character button
101. self.character.setObjectName("character")
102. # set text to Fasle to allow icon to be inserted to the button
103. self.character.setText("")
104. # set the size of the icon to 45 heigth and 45 widgth
105. self.character.setIconSize(QSize(45, 45))
106. # set the size of the button to 45 height and 45 widgth
107. self.character.setFixedSize(45, 45)
108. self.character.setEnabled(True)
110. # button to exit the game
111. self.exit\_button = QPushButton("Exit")
112. # help button
113. self.help\_button = QPushButton("help")
114. # set Object for help button
115. self.help\_button.setObjectName("help")
116. # set ObjectName for exit button
117. self.exit\_button.setObjectName("exit")
118. # connect the button to report action when pressed
119. self.exit\_button.clicked.connect(self.closeEvent)
120. self.help\_button.clicked.connect(self.help\_button\_clicked)
122. # board buttons
123. self.board = QGridLayout()
124. # customize horizontal spacing between buttons to 7px
125. self.board.setHorizontalSpacing(7)
126. # customize vertical spacing between buttons to 7px
127. self.board.setVerticalSpacing(7)
128. self.counter = 0
129. # iterate at a range of 3 to allow 3 columns
130. **for** self.column **in** range(3):
131. # iterate at a range of 3 to allow 3 rows
132. **for** self.row **in** range(3):
133. # set board button
134. self.board\_play\_button = QToolButton()
135. # Fixed size of the button to 100 height and 100 widgth
136. self.board\_play\_button.setFixedSize(100, 100)
137. # Fixed size of the icon in the button to 150 height and 150 widgth
138. self.board\_play\_button.setIconSize(QSize(150, 150))
139. # set text to Fasle to allow icon to be inserted to the button
140. self.board\_play\_button.setText("")
141. # track each button by giving it, its object name
142. self.board\_play\_button.setObjectName(str(self.counter))
143. self.board.addWidget(self.board\_play\_button, self.column, self.row)
144. # increment counter
145. self.counter += 1
146. # create widget for the board game
147. self.board\_widget = QWidget()
148. self.board\_widget.setLayout(self.board)
149. # get all the buttons in the board widget
150. self.allButtons = self.board\_widget.findChildren(QToolButton)
151. # iterate each button
152. **for** button **in** self.allButtons:
153. # connect each button to report feedback when pressed
154. button.clicked.connect(self.buttons)

157. """ empty variables that will hold the decision and the shape of the user """
158. self.decision = None
159. self.shape = None
161. self.loop\_thread = LoopThread()
162. self.loop\_thread.msg\_signal.connect(self.handle\_message)
164. ##################### LAY OUT MANAGEMENT ###########################
166. # input info grid
167. self.server\_heading\_grid = QGridLayout()
168. # heading
169. self.server\_heading\_grid.addWidget(self.heading, 0, 1, 1, 1)
170. # server input label
171. self.server\_heading\_grid.addWidget(self.server\_input\_label, 1, 0)
172. # server input
173. self.server\_heading\_grid.addWidget(self.server\_url\_input, 1, 1)
174. # connect button
175. self.server\_heading\_grid.addWidget(self.connect\_button, 1, 2)
176. self.server\_heading\_grid\_widget = QWidget()
177. self.server\_heading\_grid\_widget.setLayout(self.server\_heading\_grid)
179. # grid layout for player information
180. self.detail\_character\_grid = QGridLayout()
181. # label for messages from server
182. self.detail\_character\_grid.addWidget(self.messages\_from\_server\_label, 2, 0)
183. # messages from server
184. self.detail\_character\_grid.addWidget(self.messages\_from\_server, 3, 0)
185. self.detail\_character\_grid\_widget = QWidget()
186. self.detail\_character\_grid\_widget.setLayout(self.detail\_character\_grid)
188. # character layout information
189. self.character\_layout = QGridLayout()
190. # character display label
191. self.character\_layout.addWidget(self.character\_label, 0, 0)
192. # character of the player or user
193. self.character\_layout.addWidget(self.character, 1, 0)
194. self.character\_layout\_widget  = QWidget()
195. self.character\_layout\_widget.setLayout(self.character\_layout)
197. # layout for all the details of the user or player
198. self.detail = QHBoxLayout()
199. # layout for player information
200. self.detail.addWidget(self.detail\_character\_grid\_widget)
201. # character layout information
202. self.detail.addWidget(self.character\_layout\_widget)
203. self.detail\_widget = QWidget()
204. self.detail\_widget.setLayout(self.detail)
206. # the exit layout
207. self.button\_horizontalBox = QHBoxLayout()
208. # exit button
209. self.button\_horizontalBox.addWidget(self.exit\_button)
210. self.button\_horizontalBox.addWidget(self.help\_button)
211. self.button\_horizontalBox\_widget = QWidget()
212. self.button\_horizontalBox\_widget.setLayout(self.button\_horizontalBox)

215. vbox = QVBoxLayout()
216. vbox.addWidget(self.server\_heading\_grid\_widget)
217. vbox.addWidget(self.detail\_widget)
218. vbox.addWidget(self.button\_horizontalBox\_widget)
219. vbox\_widget  = QWidget()
220. vbox\_widget.setLayout(vbox)
222. ##############################  MAIN LAYOUT  #################################
224. self.main\_layout = QHBoxLayout()
225. self.main\_layout.addWidget(vbox\_widget)
226. self.main\_layout.addWidget(self.board\_widget)
227. self.main\_layout\_widget = QWidget()
228. self.setLayout(self.main\_layout)
230. # disable the grid button
231. self.board\_widget.setEnabled(False)
233. **def** clear\_board(self):
234. # clears the board game to allow NewGame option
235. **for** button **in** self.allButtons: # iterate each button
236. button.setText("")
237. button.setIcon(QIcon())
238. button.setEnabled(True)
240. **def** closeEvent(self):
241. # generates a question when exit button is clicked
242. reply = QMessageBox.question(
243. self, "Message",
244. "Are you sure you want to Exit?",
245. QMessageBox.Yes | QMessageBox.No
246. )
247. **if** reply == QMessageBox.Yes:
248. self.exit()
249. **else**:
250. **pass**
252. **def** keyPressEnter(self, event):
253. # close application from Escape key
254. **if** event.key() == Qt.key\_Escape:
255. self.close()
257. # fuction for connect button
258. **def** connect(self):
259. **try**:
260. self.connect\_to\_server(self.server\_url\_input.text())
261. # disables the connect button after connected succesfully
262. self.connect\_button.setEnabled(False)
263. self.messages\_from\_server.insertPlainText("=>Successfully connected to the server.\n")
264. **except**:self.messages\_from\_server.insertPlainText('=>unable to connect to the server!\n')
265. **else**:self.loop\_thread.start()
267. # fuction for game board buttons
268. **def** buttons(self):
269. self.button = self.sender()
270. self.sounds["play"].play()
271. # sends the infomation of the clicked button to server
272. self.send\_message(self.button.objectName())
274. **def** input\_play\_again(self,decision):
275. # pop up message after the Game is over, asking if the users are willing to play again
276. self.user\_response = QMessageBox.question(
277. self,
278. "Game Over",
279. "                 Results \n              "+ decision +"\nDo you wish to play again?",
280. QMessageBox.Yes|QMessageBox.No
281. )
283. # assign each button to a variable to anable the server to communicate
284. **if** self.user\_response == QMessageBox.Yes:
285. self.feedback = 'y'
286. **else**:
287. self.feedback = 'n'
289. **def** handle\_message(self, msg):
291. # indicates the new game is about to start
292. **if** msg[:msg.find(",")] == "new game":
294. # disable the game board
295. self.board\_widget.setEnabled(False)
297. # gets the shape from the message
298. self.shape = msg[-1]
299. **if** self.shape == 'O':
300. self.character.setIcon(self.o)
301. **else**:
302. self.character.setIcon(self.x)
304. # a statement indicating the player's character
305. self.messages\_from\_server.insertPlainText("=>The game is about to begin your character is " + self.shape + "\n")
306. self.messages\_from\_server.moveCursor(QTextCursor.End)
308. # indicates it’s the clients move
309. **elif** msg == "your move":
311. # Enable the game board
312. self.board\_widget.setEnabled(True)
314. # let the player know it’s their time to move
315. self.messages\_from\_server.insertPlainText("=>It's your turn to move\n")
316. self.messages\_from\_server.moveCursor(QTextCursor.End)
318. # indicates it’s the opponents move
319. **elif** msg == "opponents move":
321. # disable the game board
322. self.board\_widget.setEnabled(False)
324. # let the player know, the opponent player is about to move
325. self.messages\_from\_server.insertPlainText("=>Waiting for the opponent to move...\n")
326. self.messages\_from\_server.moveCursor(QTextCursor.End)
328. # position chosen is valid
329. **elif** msg[:msg.find(",")] == "valid move":
331. # get position from player
332. self.position = msg[-1]
334. # locate the button that is clicked
335. self.clicked\_button = self.board\_widget.findChild(QToolButton, str(self.position))
337. # Place an Icon to the clicked button
338. # msg[-1], is the shape
339. **if** msg[-3]== "X":
340. self.clicked\_button.setIcon(self.x)
341. **elif** msg[-3] == "O":
342. self.clicked\_button.setIcon(self.o)
344. # position chosen is invalid
345. **elif** msg == "invalid move":
347. # position unavailable on the board
348. self.messages\_from\_server.insertPlainText("=>You can't go there. Try again.\n")
349. self.messages\_from\_server.moveCursor(QTextCursor.End)
351. # indicates that the game is over
352. **elif** msg[:msg.find(",")] == "game over":
354. # disable the game board
355. self.board\_widget.setEnabled(False)
357. # get results from server log for the winner
358. self.winner = msg[-1] # X
360. # checks which character wins
361. **if** self.shape == self.winner:
363. # set decision, to let know the user won
364. self.decision = "YOU WIN"
366. # play sound that indicates that the play won
367. self.sounds["win"].play()
369. # print the decision to the text that shows messages from server
370. self.messages\_from\_server.insertPlainText("=>Game Over!\n=>Thank you for playing," + self.decision + "\n")
371. self.messages\_from\_server.moveCursor(QTextCursor.End)
373. # if there is no winner - it's a Tie
374. **elif** self.winner == "T" :
376. # set decision to tie
377. self.decision = "IT'S A TIE"
379. # print the decision to the text that shows messages from server
380. self.messages\_from\_server.insertPlainText("=>Game Over!\n=>Thank you for playing,It's a Tie :) \n")
381. self.messages\_from\_server.moveCursor(QTextCursor.End)
383. # checks which character looses
384. **else**:
386. # play sound that indicates that the play lost
387. self.sounds["lose"].play()
389. # set decision, to let know the user lost
390. self.decision = "YOU LOST"
392. # print the decision to the text that shows messages from server
393. self.messages\_from\_server.insertPlainText("=>Game Over!\n=>Thank you for playing," + self.decision + "\n")
394. self.messages\_from\_server.moveCursor(QTextCursor.End)
396. # see if the player wants to play again
397. **elif** msg == "play again":
399. # show the pop\_message, ask the users if they are willing to play again
400. self.input\_play\_again(self.decision)
402. # if the user wants to play again
403. **if** self.feedback == 'y':
404. **try**:self.send\_message(self.feedback)
405. **except**:self.server\_connection\_lost()
406. **else**:self.clear\_board()
408. # if the user does not want to play again
409. **else**:
410. self.feedback == 'n'
411. **try**:self.send\_message(self.feedback)
412. **except**:self.server\_connection\_lost()
414. # terminate the game
415. **elif** msg == "exit game":
417. # message displayed if the other player exit game or is taking too long to play
418. self.messages\_from\_server.insertPlainText("=>One of the players does not wish to continue\n")
419. self.messages\_from\_server.moveCursor(QTextCursor.End)
420. self.close()
421. # passage time
422. sleep(5)
424. # when the opponent disconnects from the game
425. **def** player\_lost\_connection(self):
426. # displays the text to the server message text box
427. self.messages\_from\_server.insertPlainText("=>Looks like your opponent disconnected\n")
428. # closes the server
429. self.socket.close()
430. self.messages\_from\_server.moveCursor(QTextCursor.End)
431. # clears the board
432. self.clear\_board()
433. self.socket = socket(AF\_INET, SOCK\_STREAM)
434. self.board\_widget.setEnabled(False) # disable the board
435. self.connect\_button.setEnabled(False) # disable the connect button
437. # when the host closes the server
438. **def** server\_connection\_lost(self):
439. # displays the text to the server text box
440. self.messages\_from\_server.insertPlainText("=>server disconnected!\n")
441. # closes the server
442. self.socket.close()
443. self.messages\_from\_server.moveCursor(QTextCursor.End)
444. # clears the board
445. self.clear\_board()
446. self.socket = socket(AF\_INET, SOCK\_STREAM)
447. self.board\_widget.setEnabled(False) # disable the board
448. self.connect\_button.setEnabled(False) # disable the connect button
450. # gives the user all the details they need to connect to server and how to play game
451. **def** help\_button\_clicked(self):
452. # create a pop-up dialog
453. message = Message()
454. # set title of the window
455. message.setWindowTitle("Game Instructions")
456. # the first text that will appear when the window shows up
457. message.setText("""=>Two players must play the game, and one must activate server
458. =>One player must enter the name of the local server eg 127.0.0.1
459. =>The other player must enter the server name of the host
460. =>Both the host and the player who joined must connect to server""")
461. # message in the button for more details about the game
462. message.setDetailedText("""=> The game is played on a grid that's 3 squares by 3 squares.
463. => You will be given a character between X or O,  Players take turns putting their marks in empty squares.
464. => The first player to get 3 of his/her marks in a row (up, down, across, or diagonally) is the winner.
465. => When all 9 squares are full, the game is over. If no player has 3 marks in a row, the game ends in a tie.""")
466. message.exec\_()
468. **def** play\_loop(self):
469. **while** True:
470. msg = self.receive\_message()
471. **if** len(msg): self.handle\_message(msg)
472. **else**:
473. self.player\_lost\_connection()
474. **break**
476. # fuction for exit button
477. **def** exit(self):
478. self.close()
480. # Global styles
481. stylesheet = """
482. #connect {
483. background: #004156;
484. border: none;
485. padding: 16px;
486. color: #4A586E;
487. font-size: 12px;
488. font-weight: bold;
489. }
490. #exit {
491. background: #004156;
492. border: none;
493. padding: 16px;
494. color: #4A586E;
495. font-size: 12px;
496. font-weight: bold;
497. }
498. #help{
499. background: #004156;
500. border: none;
501. padding: 16px;
502. color: #4A586E;
503. font-size: 12px;
504. font-weight: bold;
505. }
506. #help::hover {
507. border: 1px solid #48CFAF;
508. color: #01142F;
509. }
511. #character {
512. background: #09203F;
513. border: 1px solid #48CFAF;
514. padding: 16px;
515. }
516. #exit::hover {
517. border: 1px solid #48CFAF;
518. color: #01142F;
519. }
520. #connect::hover {
521. border: 1px solid #48CFAF;
522. color: #01142F;
523. }
524. QToolButton {
525. background: #4A586E;
526. padding: 16px;
527. outline: none;
528. border: none;
530. }
531. QToolButton:pressed {
532. background: #D5D5D5;
533. padding: 16px;
534. outline: none;
535. }
536. QToolButton::hover{
537. border: 1px solid #48CFAF;
538. color: #01142F;
540. }
541. QToolBuuton:checked {
542. background: #ECECEC;
543. padding: 16px;
544. outline: none;
546. }
547. QTextEdit {
548. height: 32px;
549. background: #8C8C8C;
550. height: 32px ;
551. outline: none;
552. border: none;
553. }
554. QLineEdit {
555. background: #8C8C8C;
556. padding: 16px;
557. outline: none;
558. border: none;
559. }
560. QMessageBox {
561. background: #B7D4FF;
562. text-align: center;
563. color: red;
564. outline: none;
566. }
567. """
568. app = QApplication(sys.argv)
569. OXO = OxoGame()
571. **def** main():
572. app.setStyleSheet(stylesheet)
573. OXO.show()
574. OXO.messages\_from\_server.insertPlainText("=>Enter server name to connect and press 'ENTER' key or press connect button to activate your server.\n")
575. sys.exit(app.exec\_())
576. main()