King Abdulaziz University

Department of computer Science

Faculty of Computing and Information Technology

CPCS-391 Computer Graphics Project Report, 2024_02

Prepared by:

Omar Samer Madani	ID: 2136047
Abdulelah Ali Al Turki	ID: 2136110
Ammar Abdulilah Bin Madi	ID: 2135146
Faisal Ahmed Balkhair	ID: 2136412
Mahmued Ahmed Al Ardawi	ID: 2135209

Contents

Introduction

The given code introduces a unique version of the well-known game Tic-Tac-Toe due to which both players can play without limits. The game will last until one player finally wins. The purpose of this version is to develop an interesting and exciting game for players. The game was developed using the Pygame library for window management and OpenGL for rendering graphics. The game is designed to give the players a feeling of excitement since they have to use their strategic skills and adapt to their rivals. Moreover, playing Tic-Tac-Toe is uncomplicated, but this version allows players to play games of a longer duration. Thus, it will help develop and improve their strategic thinking.

The easy and flawless display of images, the creation of dynamic visual effects, as well as the excellent processing of numerous methods of user input, are some of the features of Pygame with OpenGL, and these are three factors that improve the gaming process. Players move around the board vertically, horizontally, and diagonally utilizing the arrow keys to place their respective marks, either X or O, in a pattern that will allow them to win or continue the game eternally. This, along with the traditional Tic Tac Toe game, adds to its replay value, forcing players to develop various strategies and theories to capture their rivals. A player-friendly interface that is hassle-free and simple enough that users can focus on the game rather than thinking about how to play.

Code breakdown

Imports and declarations

```python
import pygame
import math
from pygame.locals import *
from OpenGL.GL import *
from OpenGL.GLUT import *
from OpenGL.GLU import *
import sys
import random

# Initialize the game board
board = [None] * 9
x_positions = []
o_positions = []
current_selection = 0  # Start at the top-left corner of the board
x_color = [random.random(), random.random(), random.random()]
o_color = [random.random(), random.random(), random.random()]
```

Figure 1: import and declaration section of the code.

The code starts with the imports which are necessary for our project, we used OpenGL to draw the graphics and pygame to set up the window and read keyboard button presses.

After the imports we set up global variables for the board which is initialized as a list of 9 of None, X, and O positions which will be used to track their positions, and current_selection is the current position of the selected square.

Then, we make 2 new variables x_color and o_color which are given a vector of 3 random variables between 0 and 1 to use as random colors.

Init()

```
15      def init():
16          glClearColor(0.8, 0.8, 0.8, 1.0)  # Set background color
17          glEnable(GL_LINE_SMOOTH)
18          glLineWidth(5)
```

Figure 2: init() function of the code.

sets up the color of the background as 0.8 for red, green, and blue, and 1.0 for alpha.

And adjust the properties of the lines.

Reset_board()

```
20      def reset_board():
21          global board, x_positions, o_positions, current_selection
22          board = [None] * 9
23          x_positions = []
24          o_positions = []
25          current_selection = 0
```

Figure 3: reset_board() function of the code.

This function will be called when the game ends such that another game can be played immediately.

First, the variables are set to global so they can be changed from the local function, then, each variable is set to the initial declaration such that another game can begin from scratch.

Draw_x()

```
def draw_x(pos_x, pos_y):
    glColor3fv(x_color)  # Red color for X
    glBegin(GL_LINES)
    glVertex2f(pos_x - 0.2, pos_y - 0.2)
    glVertex2f(pos_x + 0.2, pos_y + 0.2)
    glVertex2f(pos_x - 0.2, pos_y + 0.2)
    glVertex2f(pos_x + 0.2, pos_y - 0.2)
    glEnd()
```

Figure 5: draw_x function of the code

This function is responsible for drawing X in the given position. It takes x and y as input to select the position to draw the shape in. It first sets the color to the random variable we made earlier x_color, then uses GL_LINES to draw the X using the x and y input to determine positions of the lines.

Draw_o()

```
40  def draw_o(pos_x, pos_y):
41      glColor3fv(o_color)  # Red color for O
42      glBegin(GL_LINE_LOOP)
43      for angle in range(360):
44          rad = angle * 3.14159 / 180
45          glVertex2f(pos_x + 0.2 * math.cos(rad), pos_y + 0.2 * math.sin(rad))
46      glEnd()
```

Figure 6: draw_o function of the code.

This function is responsible for drawing O in the given position. It takes x and y as input to select the position to draw the shape in. It first sets color to the random variable we made earlier o_color, then uses GL_LINE_LOOP to draw 360 different points and connect them using GL_LINE_LOOP to draw the loop around the position of the x and y inputs.

Draw_board()

```
50  def draw_board():
51      glColor3f(0, 0, 0)  # Black color for board lines
52      glBegin(GL_LINES)
53      glVertex2f(-0.33, 1)
54      glVertex2f(-0.33, -1)
55      glVertex2f(0.33, 1)
56      glVertex2f(0.33, -1)
57      glVertex2f(-1, 0.33)
58      glVertex2f(1, 0.33)
59      glVertex2f(-1, -0.33)
60      glVertex2f(1, -0.33)
61      glEnd()
```

Figure 7: draw_board() function of the code.

This function is responsible for drawing the board at the start of the game. It first sets the color to black and then uses GL_LINES to draw crossing lines to make the board shape.

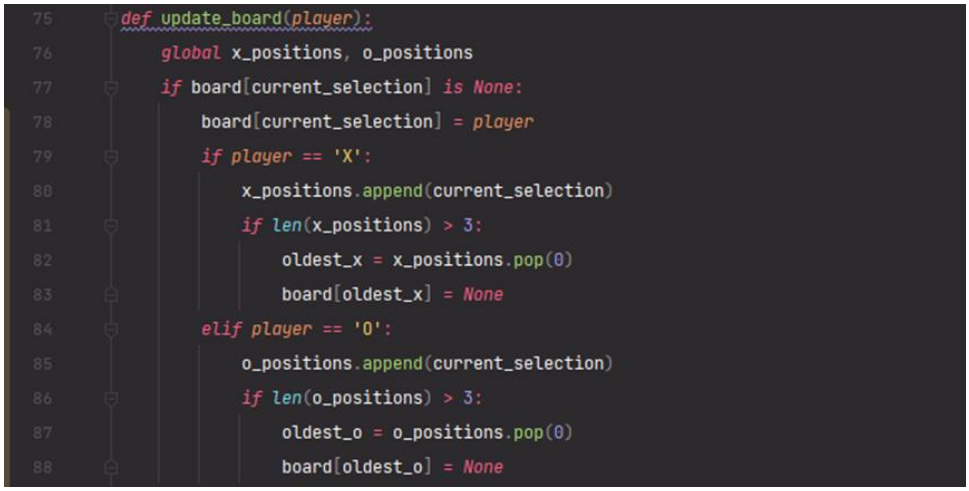
Draw_highlight()

```
63  def draw_highlight():
64      row, col = divmod(current_selection, 3)
65      x = -0.66 + col * 0.66
66      y = 0.66 - row * 0.66
67      glColor3f(1, 1, 0)  # Yellow color for highlight
68      glBegin(GL_QUADS)
69      glVertex2f(x - 0.33, y + 0.33)
70      glVertex2f(x + 0.33, y + 0.33)
71      glVertex2f(x + 0.33, y - 0.33)
72      glVertex2f(x - 0.33, y - 0.33)
73      glEnd()
```

Figure 8: draw_highlight function of the code.

This function is responsible for highlighting the selected square in yellow. First, the function gets the row and column values from the current selection_variable and applies an equation to get the borders of the square, then it sets the color to yellow by specifying 1 for red and 1 for green, after which it colors it by drawing a colored square using GL_QUADS.

Update_board()

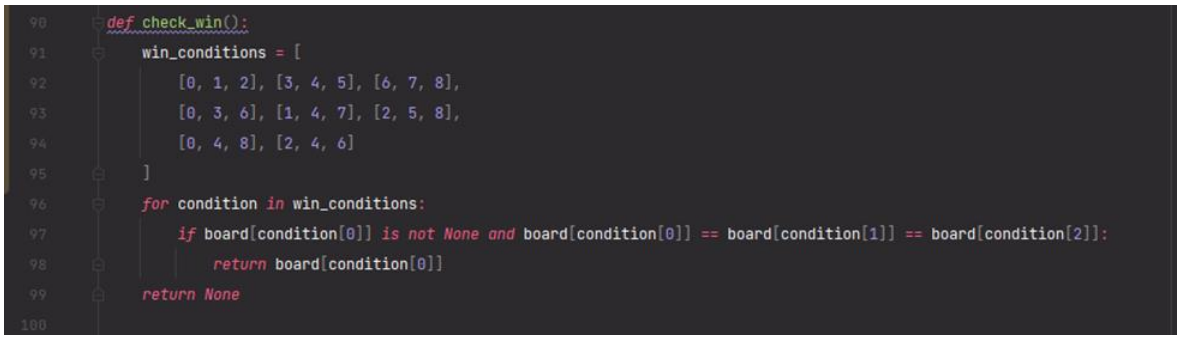


```
def update_board(player):
    global x_positions, o_positions
    if board[current_selection] is None:
        board[current_selection] = player
        if player == 'X':
            x_positions.append(current_selection)
            if len(x_positions) > 3:
                oldest_x = x_positions.pop(0)
                board[oldest_x] = None
        elif player == 'O':
            o_positions.append(current_selection)
            if len(o_positions) > 3:
                oldest_o = o_positions.pop(0)
                board[oldest_o] = None
```

Figure 9: update_board function of the code.

This function is responsible for updating the state of the game after a turn has been played. It first checks to make sure the square is empty. If it is empty, it adds the player's symbol to the square and updates the stack, such that if the length of the stack exceeds 3 after appending it pops the earliest corresponding symbol from the stack and the board (symbol referring to X or O).

Check_win()



```
def check_win():
    win_conditions = [
        [0, 1, 2], [3, 4, 5], [6, 7, 8],
        [0, 3, 6], [1, 4, 7], [2, 5, 8],
        [0, 4, 8], [2, 4, 6]
    ]
    for condition in win_conditions:
        if board[condition[0]] is not None and board[condition[0]] == board[condition[1]] == board[condition[2]]:
            return board[condition[0]]
    return None
```

Figure 10: check_win() function of the code

This function simply checks whether a player has won or not by checking the positions of X and O and comparing them to the win_conditions list which specifies all the possible ways to win in tic-tac-toe.

Main()

```python
def main():
    global current_selection
    pygame.init()
    display = (300, 300)
    pygame.display.set_mode(display, DOUBLEBUF|OPENGL)
    gluOrtho2D(-1, 1, -1, 1)
    init()
    running = True
    player = 'X'
    while running:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                sys.exit()
            elif event.type == pygame.KEYDOWN:
                if event.key == pygame.K_RIGHT and current_selection % 3 < 2:
                    current_selection += 1
                elif event.key == pygame.K_LEFT and current_selection % 3 > 0:
                    current_selection -= 1
                elif event.key == pygame.K_UP and current_selection > 2:
                    current_selection -= 3
                elif event.key == pygame.K_DOWN and current_selection < 6:
                    current_selection += 3
                elif event.key == pygame.K_SPACE:
                    if check_win() or None not in board:
                        reset_board()
                    else:

                        if check_win() or not board[current_selection] is None:
                            continue  # Skip flipping player if game is won or board is full
                        update_board(player)
                        player = 'O' if player == 'X' else 'X'
                elif event.type == pygame.KEYDOWN and event.key == pygame.K_ESCAPE:
                    running = False

        glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT)
        draw_board()
        draw_highlight()  # Highlight the current selected cell
        for i in range(9):
            if board[i] == 'X':
                draw_x(-0.66 + (i % 3) * 0.66, 0.66 - (i // 3) * 0.66)
            elif board[i] == 'O':
                draw_o(-0.66 + (i % 3) * 0.66, 0.66 - (i // 3) * 0.66)

        winner = check_win()
        pygame.display.flip()
    pygame.quit()
if __name__ == "__main__":
    main()
```

Figure 11: the main function of the code.

The last function is main:

It first initializes pygame and sets the window to 300x300 pixels then calls the above init() method to set the properties of lines and background color, then it sets the game to running and the first player to X.

Then it proceeds into a while loop that represents the game loop, then it takes input from the user if it is a quit command such as the close window button on the GUI or the escape button, it terminates the program, if however it is a keyboard press, it checks if it is an arrow key, if it is it moves the selected square according to the appropriate arrow key.

After which, it checks if it is a space button, if it is, it updates the board using the update_board() function and checks if the player has won the game using the check_win() function.

Then it proceeds to redraw the board to account for the last input using the draw_board() function highlights the selected square in yellow using the draw_highlight() function and redraws all the X's and O's again on the screen to account for the last input.

Lastly, it checks if a winner is declared and restarts the game if that is the case.

Challenges

Managing Multiple Frameworks

Integrating “pygame” for handling windows and events with “OpenGL” for graphics can be tricky. Ensuring smooth coordination between both libraries may require extra care to avoid bugs.

Win Condition Logic

Implementing a reliable win condition detector is important, as multiple winning scenarios need to be checked. The win condition checks should be efficient to avoid performance lags.

Input Validation

Properly validate keyboard inputs to avoid errors. Arrow key navigation should not exceed the board boundaries, and the selected square should be valid for placement.

Drawing Optimization

Redrawing the board and symbols every frame can be computationally bad if not optimized.

User Experience and Interface

Designing a suitable user interface requires a smart layout and rendering. Make sure the interface doesn't distract from gameplay and is clear to the players.

Future Work

Advanced Game Modes

Introduce new game modes with larger boards, varying win conditions, or different player limits to offer more challenges and engagement.

AI Opponent

Add computer-controlled opponents with varying difficulty levels, possibly implementing algorithms like Minimax for a challenging experience.

Networked Multiplayer

Include online multiplayer capabilities, allowing players to connect and compete remotely.

Mobile Version

Develop a mobile version for Android or iOS.

Customizable UI and Themes

Allow players to customize the game board, colors, and sounds according to their preferences.

References

- Sweigart, Al. Invent Your Own Computer Games with Python, 4th Ed. No Starch Press, 2016.
- "Pygame Documentation." Pygame, Pygame Development Team, https://www.pygame.org/docs/.
- "OpenGL Documentation." Khronos Group, https://www.khronos.org/opengl/wiki/Getting_Started.