DEVELOPING AN ENHANCED VERSION OF TIC-TAC-TOE USING PYGAME AND OPENGL

OUR TEAM

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

INTRODUCTION

- The project introduces an enhanced version of the classic game Tic-Tac-Toe, designed to provide players with an exciting and limitless gaming experience.

Purpose:

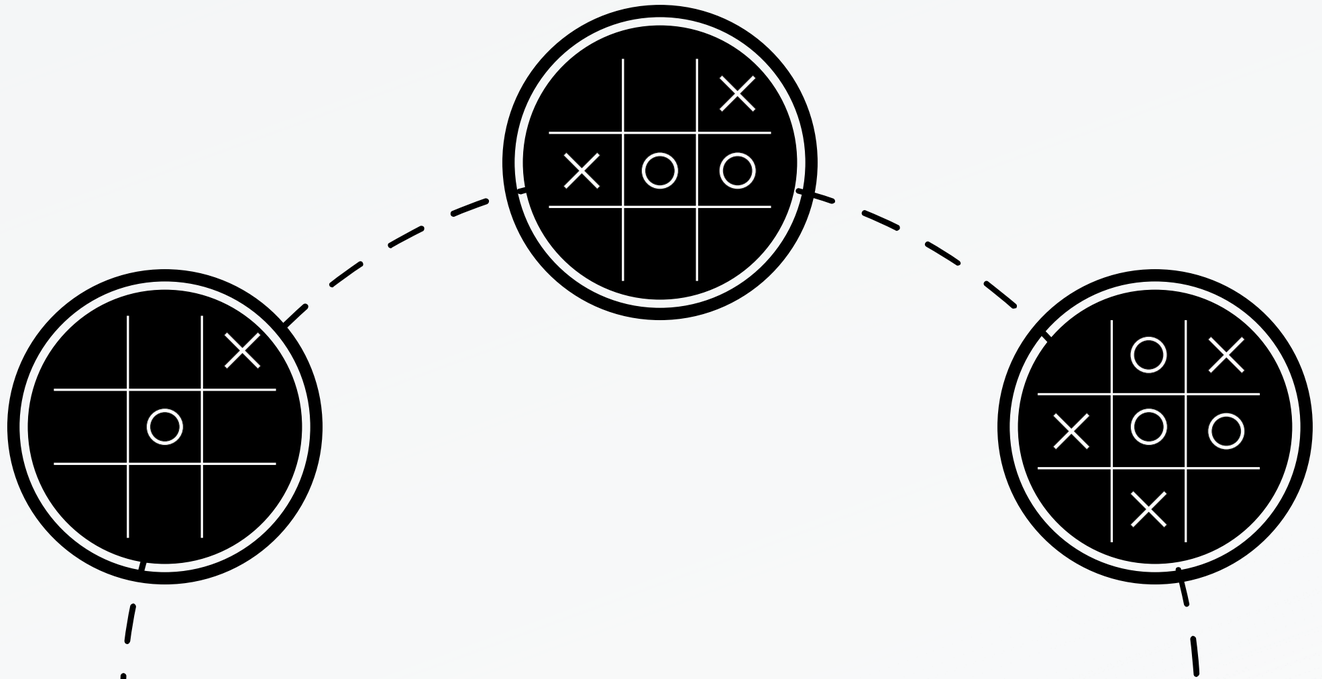
- The objective is to develop an engaging game that challenges players' strategic thinking and offers extended gameplay.
- Developed using Pygame for window management and OpenGL for graphics rendering.

Unique Features:

- Offers extended gameplay until one player wins.
- Encourages strategic thinking and adaptation to opponents' moves.

GAME OVERVIEW

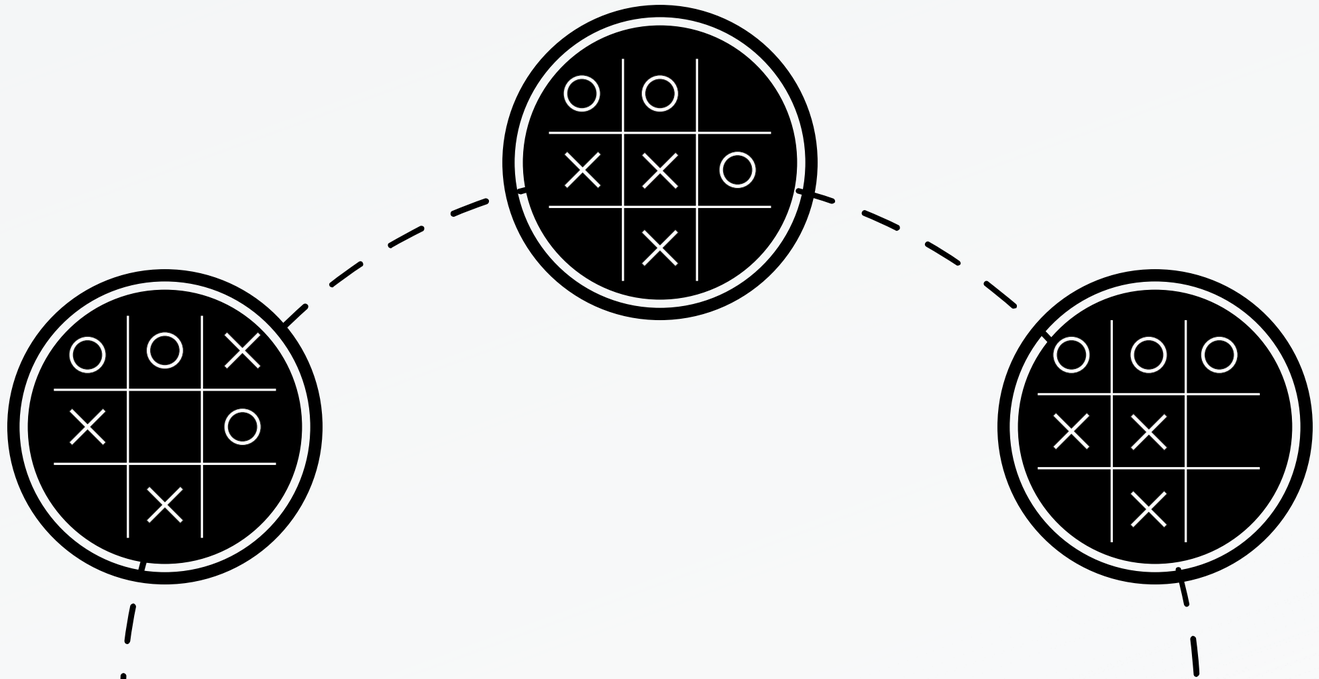
- Players place marks (X or O) on a 3x3 grid, aiming to form a winning pattern horizontally, vertically, or diagonally.
- Only three X's or three O's can be placed on the board; the fourth mark erases the first one placed by the player.



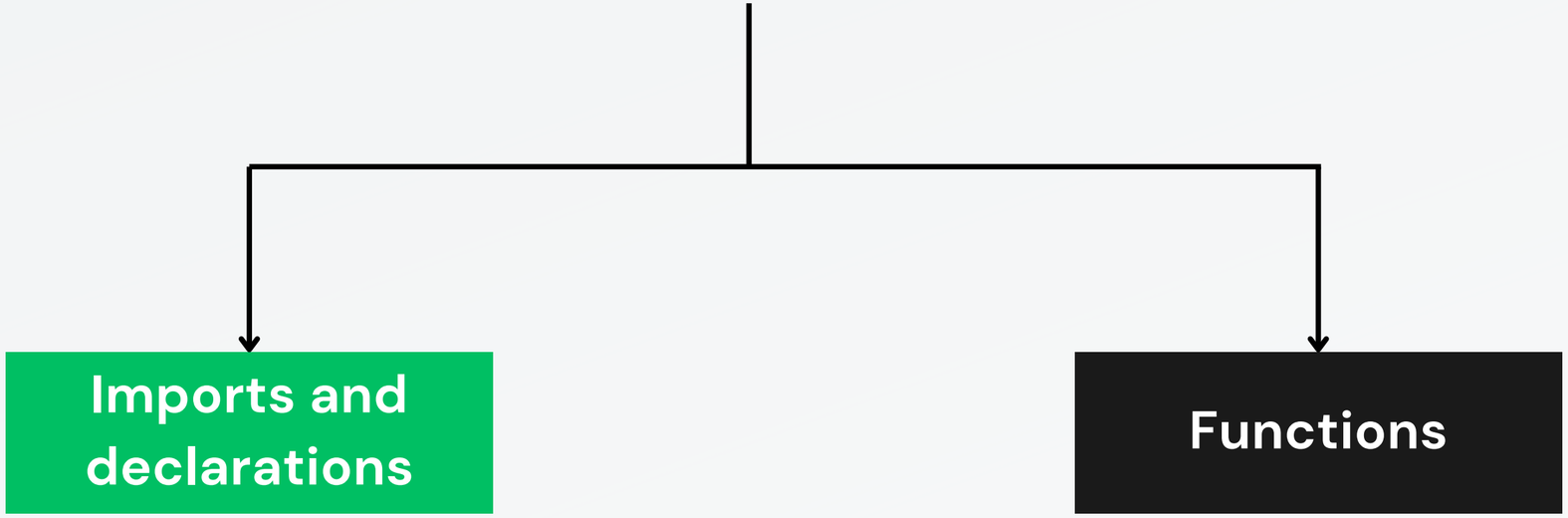
GAME OVERVIEW

Significance:

- Enhances players' strategic thinking skills.
- Provides an enjoyable and challenging gaming experience.



CODE BREAKDOWN



IMPORTS AND DECLARATIONS

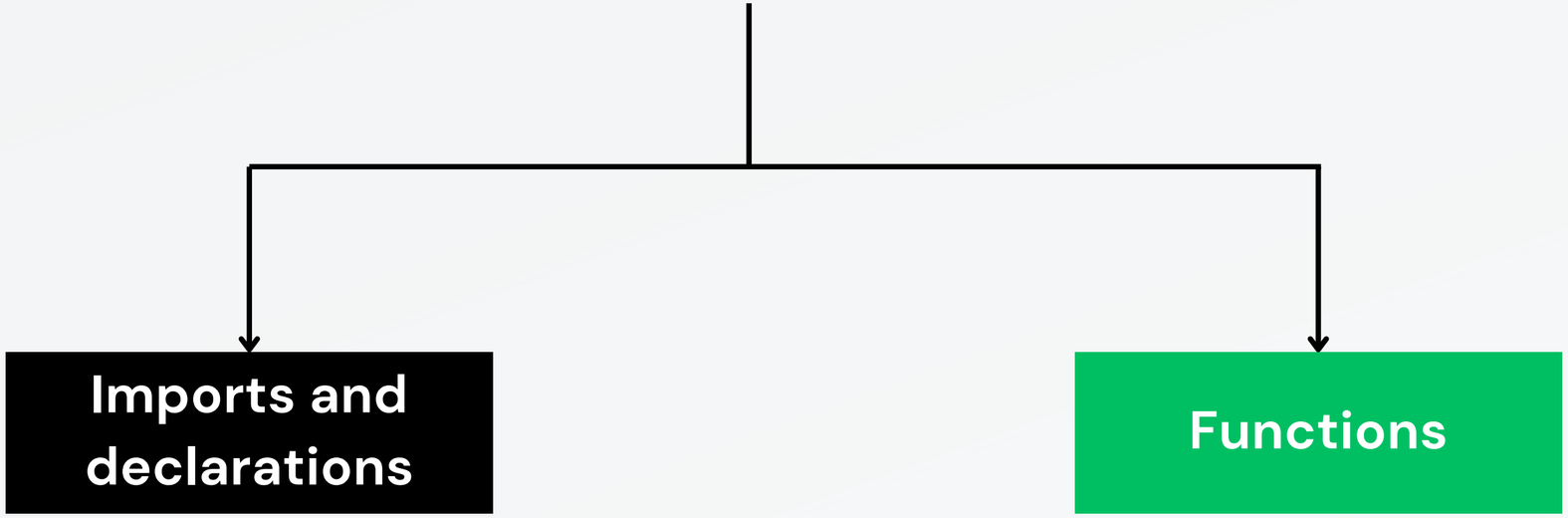
```python
import pygame
import math
from pygame.locals import *
from OpenGL.GL import *
from OpenGL.GLUT import *
from OpenGL.GLU import *
import sys
import random

# 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()]
```

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.

CODE BREAKDOWN



FUNCTIONS

Init()

```
def init():
    glClearColor(0.8, 0.8, 0.8, 1.0)  # Set backg
    glEnable(GL_LINE_SMOOTH)
    glLineWidth(5)
```

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()

```
def reset_board():
    global board, x_positions, o_positions, current_selection
    board = [None] * 9
    x_positions = []
    o_positions = []
    current_selection = 0
```

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.

FUNCTIONS

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()
```

This function is responsible for drawing X in the given position. It takes x and y as input to select position to draw the shape in. it first set color to red, then uses GL_LINES to draw the X using the x and y input to determine positions of the lines.

Draw_o()

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

This function is responsible for drawing O in the given position. It takes x and y as input to select position to draw the shape in. it first sets color to blue 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.

FUNCTIONS

Draw_board()

```
def draw_board():
    glColor3f(0, 0, 0)  # Black color for board lines
    glBegin(GL_LINES)
    glVertex2f(-0.33, 1)
    glVertex2f(-0.33, -1)
    glVertex2f(0.33, 1)
    glVertex2f(0.33, -1)
    glVertex2f(-1, 0.33)
    glVertex2f(1, 0.33)
    glVertex2f(-1, -0.33)
    glVertex2f(1, -0.33)
    glEnd()
```

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

Draw_highlight()

```
def draw_highlight():
    row, col = divmod(current_selection, 3)
    x = -0.66 + col * 0.66
    y = 0.66 - row * 0.66
    glColor3f(1, 1, 0)  # Yellow color for highlight
    glBegin(GL_QUADS)
    glVertex2f(x - 0.33, y + 0.33)
    glVertex2f(x + 0.33, y + 0.33)
    glVertex2f(x + 0.33, y - 0.33)
    glVertex2f(x - 0.33, y - 0.33)
    glEnd()
```

This function is responsible for highlighting the selected square in yellow. It 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.

FUNCTIONS

Update_board()

```python
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
```

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()

```python
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
```

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.

FUNCTIONS

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()
```

CHALLENGES

Development challenges:

- Managing multiple frameworks (Pygame and OpenGL).
- Implementing efficient win condition logic.
- Validating user input for smooth gameplay.
- Optimizing drawing operations for performance.
- Designing a user-friendly interface without distracting from gameplay.

FUTURE WORK

Suggestions for Future Enhancements:

- Introduce advanced game modes with larger boards and varying win conditions.
- Implement AI opponents with varying difficulty levels.
- Enable networked multiplayer capabilities.
- Develop a mobile version for wider accessibility.
- Allow customization of UI and themes according to player preferences.

THANK'S FOR WATCHING