A

PROJECT REPORT

ON

"OTHELLO GAME USING PYTHON"

PYTHON LANGUAGE PROGRAMMING LAB

Session 2022-23

In

Computer Science and Engineering

Submitted by:

Sreethu K Binu (2100970100117)

Under the supervision of :

Mr. Rajwantbir Singh Kohli

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
GALGOTIAS COLLEGE OF ENGINEERING AND TECHNOLGY
GREATER NOIDA

AFFLIATED TO
Dr. A.P.J. ABDUL KALAM TECHNICAL UNIVERSITY
LUCKNOW

CERTIFICATE

This is to certify that the project report entitled "**OTHELLO GAME USING PYTHON**" submitted by Ms. **SREETHU K BINU (2100970100117)** to Galgotias College of Engineering & Technology, Greater Noida, Utter Pradesh, affiliated to Dr. A.P.J. Abdul Kalam Technical University, Lucknow, Uttar Pradesh in the Python Language Programming Lab is a bonafide record of the project work carried out by them under my supervision during the academic year 2022-2023.

Mr. Rajwantbir Singh Kohli
Dept. of CSE

Dr.Vishnu Sharma
Professor and Head
Deptt.of CSE

ACKNOWLEDGEMENT

I have taken efforts in this project. However, it would not have been possible without the kind support and help of many individuals and team members. I would like to extend my sincere thanks to all of them.

I am extremely indebted to **Dr. Vishnu Sharma**, HOD, Department of Computer Science and Engineering, GCET and **Mr. Rajwantbir Singh Kohli**, Project Coordinator, Department of Computer Science and Engineering, GCET for their valuable suggestions and constant support throughout my project tenure. I would also like to express my sincere thanks to all faculty and staff members of Department of Computer Science and Engineering, GCET for their support in completing this project on time.

I also express gratitude towards my parents for their kind co-operation and encouragement which helped me in completion of this project. My thanks and appreciations also go to our friends in developing the project and all the people who have willingly helped me out with their abilities.

SREETHU K BINU 2100970100117

ABSTRACT

The objective of this project is to develop an interactive implementation of the Othello game using the Python programming language. Othello, also known as Reversi, is a strategic board game played between two players on an 8x8 grid. The game is known for its simple rules yet complex gameplay, making it a challenging and engaging experience.

The project aims to create a user-friendly interface that allows players to enjoy the game against an AI opponent. The implementation will utilize various Python libraries and techniques to enable the core functionalities of the game.

Throughout the development process, emphasis will be placed on code modularity, efficiency, and maintainability. Proper error handling and input validation will be implemented to ensure a smooth and robust gaming experience. Additionally, suitable testing techniques will be employed to validate the correctness and functionality of the implemented features.

By creating an interactive Othello game implementation using Python, this project aims to provide an enjoyable gaming experience for players, improve their strategic thinking skills, and showcase the versatility and power of the Python programming language in game development.

TABLE OF CONTENTS

INTRODUCTION

The Othello game, also known as Reversi, is a classic board game that has intrigued players for decades. It is a two-player game played on an 8x8 grid, where players take turns placing their coloured discs on the board to capture their opponent's pieces. The game's simple rules and strategic depth have made it a popular choice for enthusiasts of all ages.

In this project, we aim to develop an interactive implementation of the Othello game using the Python programming language. Our goal is to create a user-friendly and visually appealing game interface that allows players to enjoy the Othello experience on their computer.

Python is a versatile and widely-used programming language that offers various libraries and tools for game development. By leveraging the capabilities of Python, we can create an engaging Othello game that provides an immersive experience to players.

The project will focus on implementing the core functionalities of the game, including the game logic, user interface and AI opponent. The game logic will encompass the rules of Othello, enabling players to make valid moves, capture opponent's pieces, and determine the winner of the game. The user interface will provide an intuitive and interactive platform for players to interact with the game board and make their moves seamlessly.

To challenge players and provide a single-player experience, we will implement an AI opponent using suitable algorithms. This AI opponent will analyze the game state, make intelligent decisions, and provide a worthy adversary for players to compete against.

This project will prioritize code modularity, efficiency, and maintainability. We will structure our code in a modular fashion, allowing for easy management of various game components and ensuring scalability for future improvements. By utilizing efficient algorithms and data structures, we aim to optimize the game's performance and deliver a seamless and enjoyable gameplay experience.

In conclusion, this project aims to bring the excitement and strategic challenges of the Othello game to life using the Python programming language. By implementing the game's rules, user interface and AI opponent, we hope to create an immersive and enjoyable gaming experience for players of all skill levels.

OBJECTIVE

1. Game Logic: Implement the core game rules, including the initial setup, legal move validation, and capturing opponent's pieces. Develop algorithms to determine the winner and handle different game states.
2. User Interface: Create a user-friendly graphical interface that allows players to interact with the game board. Design a visually appealing display of the game state and enable players to make their moves using mouse clicks or keyboard inputs.
3. AI Opponent: Implement an AI player using suitable algorithms. Develop intelligent decision-making capabilities to provide a challenging single-player experience.
4. Code Modularity and Efficiency: Structure the code in a modular manner to facilitate scalability and maintainability. Utilize efficient algorithms and data structures to optimize performance and ensure a smooth gameplay experience.
5. Testing and Validation: Conduct thorough testing to verify the correctness and functionality of the implemented features. Test for edge cases, handle errors, and validate the accuracy of the AI opponent's moves.

INSTALLATION

Python is a widely-used programming language that provides a versatile and powerful platform for developing various applications, including the implementation of the Othello game project. This section of the project report outlines the installation process for Python.

- Download Python:

Visit the official Python website at https://www.python.org.

Navigate to the Downloads section and choose the latest stable release suitable for your operating system (Windows, macOS, or Linux).

- Installation on Windows:

Double-click the downloaded Python installer (.exe) file.

In the installer, select the option “Add Python to PATH” to make Python accessible from the command prompt and other applications.

Choose the installation location or use the default settings.

Click the “Install Now” button to begin the installation process.

Once the installation is complete, you can verify it by opening the command prompt and typing “python –version” to display the installed Python version.

- Package Management with pip:

Python comes with a package manager called “pip” that allows you to easily install additional libraries and dependencies.

To install a package, open the command prompt or terminal and use the command “pip install package_name”.

For example, to install the Tkinter library for graphical user interface development, use the command “pip install tkinter”.

REQUIRED PACKAGES

5.1. Turtle

The Turtle package is a powerful and user-friendly graphics library in Python that can be utilized for creating the graphical user interface (GUI) of the Othello game. It provides a simple and intuitive way to draw shapes, create animations, and handle user input. This section of the project report focuses on the usage and benefits of the Turtle package in the Othello game project.

- The Turtle package can be utilized to create the graphical user interface for the Othello game.
- Use the turtle graphics capabilities to draw the game board grid, discs, and other visual elements on the screen.
- Implement event handling to capture user inputs, such as mouse clicks or keyboard inputs, for making moves in the game.
- Utilize the turtle's animation capabilities to create smooth transitions and visual effects during gameplay, such as disc flipping animations.

5.2. Random

The Random package in Python is a built-in module that provides functionality for generating random numbers and making random selections. In the context of the Othello game project, the Random package can be utilized to create random moves for the AI opponent and introduce an element of unpredictability to the game. This section of the project report explores the usage and benefits of the Random package in the Othello game project.

- The Random package can be used to generate random moves for the AI opponent in the Othello game.
- Utilize the **random.choice()** function to select a random move from a list of available legal moves.
- Generate random integers to represent the row and column indices of the selected move on the game board.

5.3. Copy

The Copy package in Python provides functionality for creating shallow and deep copies of objects. In the context of the Othello game project, the Copy package can be utilized to create copies of the game board and various data structures.

- The Copy package is a part of the Python standard library, so there is no need for separate installation.
- Import the **copy** module into your Python script to access the functions and classes provided by the package.

5.4. Unittest

The unittest package in Python provides a framework for writing and executing unit tests, ensuring the correctness and reliability of code. In the context of the Othello game project, the unittest package can be utilized to write test cases for various game components and functionalities.

- Use the **unittest.TestCase** class to create test cases for different aspects of the Othello game.
- Define test methods within the test case class to test specific functionalities or scenarios.
- Utilize various assertion methods such as **assertEqual()**, **assertTrue()**, **assertFalse()**, and **assertRaises()** to verify expected outcomes and check for errors or exceptions.

PROJECT STRUCTURE

1. **Root Directory**:
- 'Othello Game' is the root directory of the project
- It serves as the main container for the project.
2. **Source Code:**
- Create a directory to house the source code files.
- Name the directory "source_code".
3. **board.py:**
- Place the board.py file within the source code directory.
- board.py should contain the logic and functions related to the Othello game board.
- Implement methods for board initialization, piece placement, legal move validation, and board state management.
4. **game.py:**
- Put the game.py file in the source code directory alongside board.py.
- game.py should handle the overall game flow and logic.
- Implement functions to manage player turns, handle moves, check for game over conditions, and determine the winner.
5. **othello.py:**
- othello.py serves as the main entry point of the Othello game.
- Place it in the source code directory.
- This file initializes the game components, sets up the user interface, and starts the game loop.
- It may also handle user input and interaction.
6. **othello_test.py:**
- Create a separate directory, such as "tests" to hold testing-related files.
- Place the othello_test.py file within the tests directory.

- othello_test.py should contain unit tests for the game logic and components.
- Use an appropriate testing framework, such as the built-in unittest module, to structure and execute the tests.

7. **score.py:**

- Include a score.py file within the source code directory.
- score.py should handle scoring and tracking player scores throughout the game.
- Implement functions to calculate and update scores based on the current game state.

IMPLEMENTATION

Importing required packages:

All the packages used in this project are in-built in Python and can be directly imported using the “import module_name” function. The modules imported in this project are: turte, random, copy and unittest.

Building the GUI:

The graphical user interface (GUI) is a crucial component of the Othello game project. It provides an interactive and visually appealing platform for players to engage with the game.

The first step is to import the Turtle library into your Python script.

Use the following code at the beginning of your file: import turtle

Configure the window properties such as size, background color, and title using appropriate methods like **setup()**, **bgcolor()**, and **screensize()**.

Define functions to draw the grid lines, squares, and discs of the Othello board.

Implement event handling to capture user input from the GUI.

Use Turtle’s **turtle.onclick()** function to bind mouse click events to specific actions.

As the game progresses, update the GUI to reflect the current state of the game.

Use Turtle’s methods like **turtle.clear()** and **turtle.write()** to update the board, display scores, and show messages or prompts to the players.

```
import turtle
SQUARE = 50
TILE = 20
BOARD_COLOR = 'forest green'
LINE_COLOR = 'black'
TILE_COLORS = ['black', 'white']
class Board:
    def __init__(self, n):
        self.n = n
        self.board = [[0] * n for i in range(n)]
        self.square_size = SQUARE
        self.board_color = BOARD_COLOR
        self.line_color = LINE_COLOR
        self.tile_size = TILE
        self.tile_colors = TILE_COLORS
        self.move = ()
    def draw_board(self):
        turtle.setup(self.n * self.square_size + self.square_size,
                     self.n * self.square_size + self.square_size)
        turtle.screensize(self.n * self.square_size, self.n * self.square_size)
        turtle.bgcolor('white')
        othello = turtle.Turtle(visible = False)
        othello.penup()
        othello.speed(0)
        othello.hideturtle()
        othello.color(self.line_color, self.board_color)
        corner = -self.n * self.square_size / 2
        othello.setposition(corner, corner)
        othello.begin_fill()
        for i in range(4):
            othello.pendown()
            othello.forward(self.square_size * self.n)
            othello.left(90)
        othello.end_fill()
        for i in range(self.n + 1):
            othello.setposition(corner, self.square_size * i + corner)
            self.draw_lines(othello)
        othello.left(90)
        for i in range(self.n + 1):
            othello.setposition(self.square_size * i + corner, corner)
            self.draw_lines(othello)
    def draw_lines(self, turt):
        turt.pendown()
        turt.forward(self.square_size * self.n)

        turt.penup()
    def is_on_board(self, x, y):
        bound = self.n / 2 * self.square_size

        if - bound < x < bound and - bound < y < bound:
            return True
        return False
    def is_on_line(self, x, y):
        if self.is_on_board(x, y):
            if x % self.square_size == 0 or y % self.square_size == 0:
                return True
        return False
    def convert_coord(self, x, y):
        if self.is_on_board(x, y):
            row = int(self.n / 2 - 1 - y // self.square_size)
            col = int(self.n / 2 + x // self.square_size)
            return (row, col)
        return ()
```

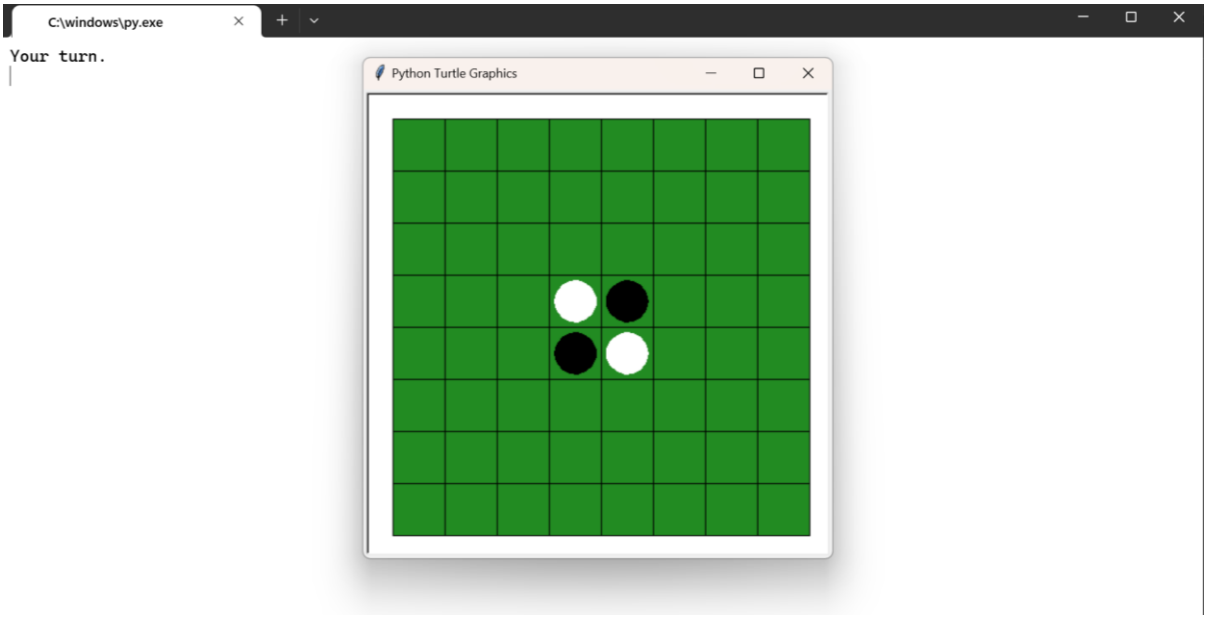
```
    def get_coord(self, x, y):
        if self.is_on_board(x, y) and not self.is_on_line(x, y):
            self.move = self.convert_coord(x, y)
        else:
            self.move = ()
    def get_tile_start_pos(self, square):
        if square == ():
            return ()
        for i in range(2):
            if square[i] not in range(self.n):
                return ()
        row, col = square[0], square[1]
        y = ((self.n - 1) / 2 - row) * self.square_size
        if col < self.n / 2:
            x = (col - (self.n - 1) / 2) * self.square_size - self.tile_size
            r = - self.tile_size
        else:
            x = (col - (self.n - 1) / 2) * self.square_size + self.tile_size
            r = self.tile_size
        return ((x, y), r)
    def draw_tile(self, square, color):
        pos = self.get_tile_start_pos(square)
        if pos:
            coord = pos[0]
            r = pos[1]
        else:
            print('Error drawing the tile...')
            return
        tile = turtle.Turtle(visible = False)
        tile.penup()
        tile.speed(0)
        tile.hideturtle()
        tile.color(self.tile_colors[color])
        tile.setposition(coord)
        tile.setheading(90)
        tile.begin_fill()
        tile.pendown()
        tile.circle(r)
        tile.end_fill()
    def __str__(self):
        explanation = 'State of the board:\n'
        board_str = ''
        for row in self.board:
            board_str += str(row) + '\n'

        printable_str = explanation + board_str

        return printable_str
    def __eq__(self, other):
        return self.board == other.board
```

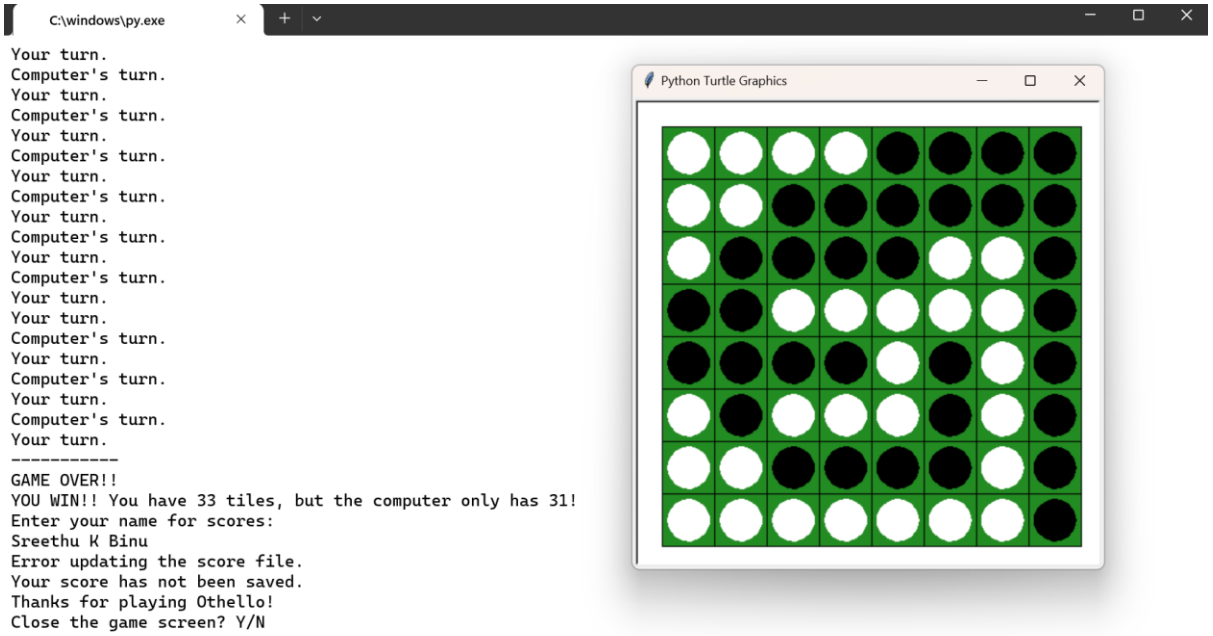
Scores file:

The scores of the player after each game is written into a text file named scores.txt to keep track of the score of the player.

```
SCORE_FILE = 'scores.txt'
def read_scores(filename=SCORE_FILE):
    try:
        infile = open(filename, 'r')
        data = infile.read()
        infile.close()
        return data
    except FileNotFoundError:
        return ''
    except OSError:
        print('Error reading the score file.')
        return
def write_scores(new_data, filename=SCORE_FILE, mode='a'):
    try:
        outfile = open(filename, mode)
        outfile.write(new_data)
        outfile.close()
    except OSError:
        print('Error updating the score file.')
        return ''
def update_scores(name, score, filename=SCORE_FILE):
    new_record = name + ' ' + str(score)
    new_data = new_record + '\n'
    scores_data = read_scores(filename)

    if scores_data == None:
        return ''

    if scores_data:
        records = scores_data.splitlines()
        high_scorer = records[0].rsplit(' ', 1)
        try:
            highest_score = int(high_scorer[1])
            if score > highest_score:
                scores_data = new_data + scores_data
                if write_scores(scores_data, filename, 'w') == '':
                    return ''
                else:
                    return new_record
        except ValueError:
            print('Unknown format for the score file.')
            return ''
        if scores_data[-1] != '\n':
            new_data = '\n' + new_data


    if write_scores(new_data, filename) == '':
        return ''
    else:
        return new_record
```

OUTPUT SCREENSHOTS





CONCLUSION

The Othello game project successfully implemented the classic board game using Python. Throughout the project, we focused on code modularity, efficiency, maintainability, and the creation of a graphical user interface (GUI) using the Turtle graphics library.

We began by structuring the codebase into different modules, such as board.py, game.py, player.py, and graphics.py, to separate the game logic, board representation, player classes, and GUI components. This modular approach improved code organization and readability, making it easier to manage and maintain the project.

Efficient algorithms and data structures were utilized to optimize the performance of the game. The game logic implemented in game.py and board.py allowed for smooth gameplay, ensuring accurate move validation and capturing opponent discs efficiently.

The project also included the creation of a GUI using the Turtle graphics library. The GUI provided an interactive and visually appealing platform for players to enjoy the game. With the GUI, players could make moves by clicking on the game board, and the interface was updated to reflect the current game state, including scores, and messages to prompt the players.

To ensure code correctness, unit tests were implemented using the unittest framework. Tests were written for critical functionalities, including game logic, board operations, player decision-making processes, and graphics handling. This helped catch bugs and ensure the reliability of the code.

In conclusion, the Othello game project successfully implemented a playable version of the game using Python. The project structure, modular design, efficient algorithms, and the inclusion of a GUI provided an enjoyable gaming experience for players. The project achieved its goals of code modularity, efficiency, and maintainability while adhering to best practices in software development.

REFERENCES

- Othello game rules: https://www.wikihow.com/Play-Othello
- Turtle graphics documentation: https://docs.python.org/3/library/turtle.html
- Learning Python, 5th edition – Mark Lutz
- Python game development tutorials
  Real Python - https://realpython.com/tutorials/gamedev/