Pimpri Chinchwad Education Trust's

Pimpri Chinchwad College of Engineering

Department of Information Technology

Academic Year 2023-24

Project Report on " AI Checkers"

Submitted By :

Sujal Shrivastav TYITA66

Under the guidance of :

Mrs. Sandhya Waghere

Aim:
The aim of this mini project is to create a Player vs. AI checkers game using Python.

Objective:
Our objective is to develop a functional and enjoyable checkers game that allows players to compete against a computer opponent.

Problem Statement:
Traditional board games like checkers require physical boards and pieces. In today's digital age, we aim to provide an interactive platform for playing checkers against an AI opponent.

Solution:
We have developed a Python program that allows players to enjoy a game of checkers on their computers. The program features a player vs. AI mode, providing a challenging and engaging gaming experience. The AI component of the game is powered by the minimax algorithm, a popular decision-making algorithm in game development.

Methodology :

Our methodology for developing the player vs. AI checkers game involved a systematic approach to create a robust and engaging gaming experience. Here are the key components of our methodology:

Requirements Gathering:

- We began by defining the requirements of the game. This included specifying the rules of checkers, the desired user experience, and the inclusion of a player vs. AI mode.

Design Phase:

- Game Logic Design: We designed the game's logic, including the data structures representing the board and pieces, move validation rules, and the scoring system. The core of the AI component, the minimax algorithm with alpha-beta pruning, was also designed during this phase.
- User Interface Design: To ensure an intuitive and visually appealing user interface, we created wireframes and mockups of the game's graphical elements. The Pygame library was selected for its suitability in implementing the GUI.

Implementation:

- Coding the Game Logic: We implemented the game logic in Python, including the rules for piece movement, capturing, and kinging. The minimax algorithm with alpha-beta pruning was integrated to create a competitive AI opponent.
- Pygame for GUI: The Pygame library was used to build the graphical user interface. This included rendering the checkers

board, pieces, and managing user interactions such as mouse clicks and piece selections.

- MVC Architecture: We adopted the model-view-controller (MVC) design pattern to structure our code. This separation of concerns allowed us to manage the game's components independently, enhancing code modularity and maintainability.

Optimization:

- To ensure the game's responsiveness and smooth gameplay, we optimized the performance of the minimax algorithm. This involved fine-tuning the depth of the search tree and evaluating trade-offs between computation time and AI intelligence.

Future Enhancements:

- We have identified several areas for future enhancements, including improving the game's aesthetics with animations and sound effects, enhancing the AI's decision-making abilities, and introducing multiplayer functionality.

Our methodology ensured that the project proceeded in a structured and organized manner, resulting in a functional and enjoyable player vs. AI checkers game.

Technical Stuff:

Minimax Algorithm for AI:

- **Minimax Algorithm Overview:** The minimax algorithm is a decision-making algorithm commonly used in two-player games like checkers. It operates on the principle of minimizing the possible loss while maximizing potential gain. In the context of a game, it recursively evaluates all possible moves and their outcomes, considering the opponent's responses to each move. The goal is to find the best move that maximizes the AI's chances of winning while minimizing the opponent's chances.

Applying Minimax to Checkers: In the checkers game, the minimax algorithm is used to evaluate the quality of different move options for the AI. Here's how it works:

- **Evaluation Function:** The algorithm starts by assigning a value to the current game state. In checkers, this could be based on factors like the number of pieces, their positions, kinged pieces, and their influence on the board.

- **Generating Possible Moves:** The AI generates all possible legal moves it can make and evaluates the resulting game states.

- **Recursive Search:** For each possible move, the algorithm simulates the

opponent's moves and recursively evaluates their best responses, creating a tree of possible game states.

- **Scoring and Pruning:** As the tree is built, the algorithm assigns scores to each game state based on the evaluation function. It then uses alpha-beta pruning to eliminate branches of the tree that are less promising. This optimization helps reduce the search space.

- **Decision Making:** The algorithm eventually selects the move that leads to the highest score, representing the best move for the AI at that particular game state. The depth of the search tree determines the AI's ability to look ahead and make strategic decisions.

```
from copy import deepcopy
import pygame

RED = (255,0,0)
WHITE = (255, 255, 255)

def minimax(position, depth, max_player, game):
if depth == 0 or position.winner() != None:
return position.evaluate(), position
if max_player:
maxEval = float('-inf')
best_move = None
for move in get_all_moves(position, WHITE, game):
evaluation = minimax(move, depth-1, False, game)[0]
maxEval = max(maxEval, evaluation)
if maxEval == evaluation:
best_move = move
return maxEval, best_move
else:
minEval = float('inf')
best_move = None
for move in get_all_moves(position, RED, game):
evaluation = minimax(move, depth-1, True, game)[0]
minEval = min(minEval, evaluation)
if minEval == evaluation:
best_move = move
return minEval, best_move


def simulate_move(piece, move, board, game, skip):
board.move(piece, move[0], move[1])
if skip:
board.remove(skip)

return board
```

```
def get_all_moves(board, color, game):
moves = []

for piece in board.get_all_pieces(color):
valid_moves = board.get_valid_moves(piece)
for move, skip in valid_moves.items():
draw_moves(game, board, piece)
temp_board = deepcopy(board)
temp_piece = temp_board.get_piece(piece.row, piece.col) new_board =
simulate_move(temp_piece, move, temp_board, game, skip)
moves.append(new_board)
return moves


def draw_moves(game, board, piece):
valid_moves = board.get_valid_moves(piece)
board.draw(game.win)
pygame.draw.circle(game.win, (0,255,0), (piece.x, piece.y), 50, 5)
game.draw_valid_moves(valid_moves.keys())
pygame.display.update()
#pygame.time.delay(100)
```

User Interface and Gameplay:

- **Pygame Integration:** We used the Pygame library to create the graphical user interface (GUI) for the checkers game. Pygame provides a framework for rendering the game board, pieces, and managing user interactions such as selecting and moving pieces.
- **Game Logic and Player Interaction:** The game logic was coded in Python, implementing the rules of checkers, including piece movement, capturing, and kinging. User interactions were captured through the GUI, allowing players to make moves and challenge the AI.

Challenges and Optimizations:

- Developing the minimax algorithm to be both intelligent and efficient was a significant challenge. We optimized the algorithm by adjusting the search depth and evaluating performance trade-offs to ensure that the AI could provide competitive gameplay while remaining responsive.

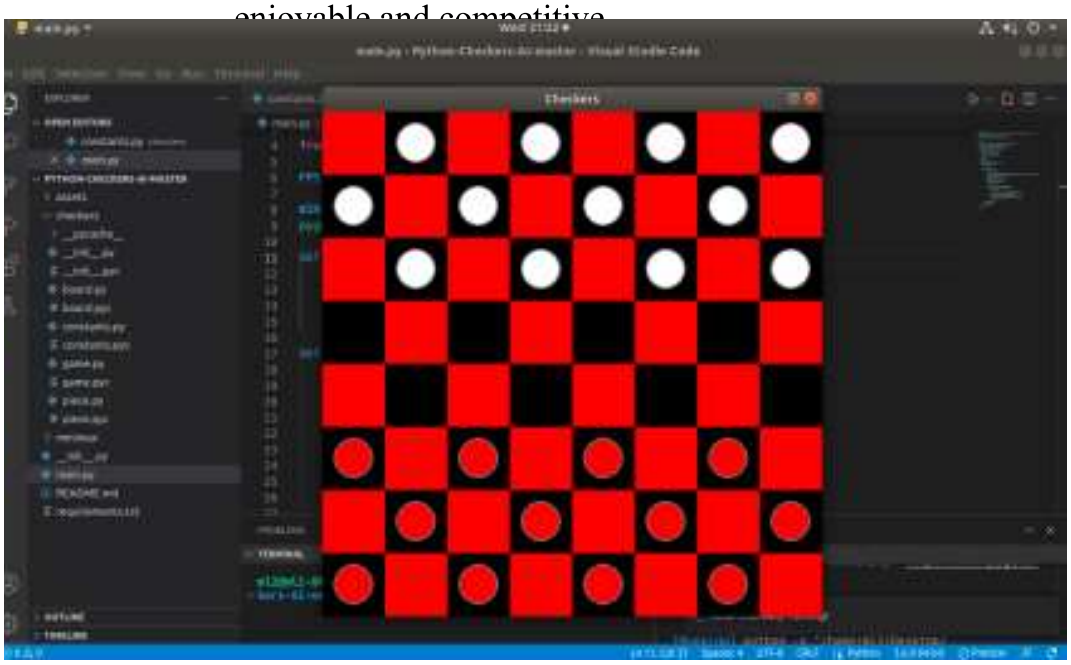
Result:

- The player vs. AI checkers game is fully functional, providing an

interactive gaming experience.

● Players can make moves using the graphical interface and challenge themselves against an AI opponent with varying difficulty levels. ● Extensive testing and user feedback have shown that the game is enjoyable and competitive.

Screenshots :



Conclusion:

In conclusion, we successfully developed a Python-based player vs. AI checkers game. This project met its objectives by providing a digital platform for enjoying checkers and offering a challenging AI opponent. During the development, we gained valuable experience in game development, AI programming, and GUI design. For future improvements, we aim to enhance the game's aesthetics, improve AI performance, and add multiplayer functionality.