JSS MAHAVIDYAPEETHA

JSS Science and Technology University

A Report on

"CHECKERS WITH AI"

EVENT 1

(Subject: Artificial Intelligence)

(Subject code: 22CS540)

Submitted by

Roll. No.	USN	NAME
55	02JST22UCS082	Purnaa L
04	01JST22UCS007	Aditi Katta
16	01JST22UCS067	Keerthana K V
30	01JST22UCS126	Raksha N Urs

Under the guidance of

Prof. Raksha R

Assistant Professor

Dept. of CS&E

SJCE, JSS STU, Mysuru

Department of Computer Science & Engineering

2024-2025

JSS MAHAVIDYAPEETHA

JSS Science and Technology University

Certificate

Certified that the project entitled "Checkers with AI" for Event 1, conducted for the course "Artificial Intelligence **(22CS540)**", was undertaken by Ms. Purnaa L- 02JST22UCS082, Ms. Aditi Katta - 01JST22UCS007 , Ms. Keerthana K V - 01JST22UCS067, Ms. Raksha N Urs -01JST22UCS126 in partial fulfillment of the requirements of [Semester V/Year III], as prescribed by the JSS Science and Technology University, Mysore, during the year 2024-2025. It is certified that all corrections/suggestions indicated for the project have been incorporated. The Event 1 project report has been approved as it satisfies the academic requirements for the respective course.

Smt. Raksha R
Assistant Professor
Department of CS & E,
JSS STU, Mysuru-06

Dr Srinath S
Associate Professor and Head
Department of CS & E
JSS STU, Mysuru-06

DECLARATION

We do hereby declare that the project titled **"Checkers With AI"** is carried out by Ms. Purnaa L - 02JST22UCS082, Ms. Aditi Katta - 01JST22UCS007 , Ms. Keerthana K V – 01JST22UCS067, Ms. Raksha N Urs -01JST22UCS126 of the requirements of [V/III], as prescribed by JSS Science and Technology University, Mysore, during the year 2024-2025.

Date: 10/1/2025

Place: Mysuru

Purnaa L (02JST22UCS082)

Aditi Katta (01JST22UCS007)

Keerthana K V (01JST22UCS067)

Raksha N Urs (01JST22UCS126)

ABSTRACT

Checkers is a strategic two-player board game that has been widely played for centuries. This project involves the development of a Checkers game using the Python programming language, utilising the Pygame library for graphical rendering and providing a visually interactive environment. The game supports two human players or a human player against an AI opponent, which uses the Minimax algorithm with alpha-beta pruning for decision-making. This allows the AI to simulate a realistic and challenging opponent by evaluating potential moves and optimising its strategy. The game includes essential features such as piece movement, capturing, kinging, and determining the winner, along with intuitive user input handling for selecting pieces and making moves. The system is designed to offer an engaging and accessible platform for players of all skill levels while also serving as a tool for studying the practical application of game theory algorithms in real-time gameplay, performance, and strategic decision-making.

ACKNOWLEDGEMENTS

An endeavor is successful only when it is carried out under proper guidance and blessings. We would like to thank the few people who helped us in carrying out this work by lending invaluable assistance. We thank Dr. C Nataraju, Principal, JSSSTU, Mysuru and Dr. Srinath S, Associate Professor and Head, Department of Computer Science and Engineering, JSSSTU, Mysuru who encouraged us at this venture. It is our foremost duty to thank our project guide Smt. Raksha R for her encouragement, effective guidance and valuable suggestions right from the beginning of this project to till its completion. We also extend our regards to all the teaching and nonteaching members of the Department of Computer Science and Engineering for their direct or indirect support towards the completion of this project. We would also like to thank our family and friends for their constant support.

TABLE OF CONTENTS

CHAPTER 1
INTRODUCTION

The Checkers AI Game is an interactive and engaging two-player strategy game where players can enjoy a competitive match against an AI opponent or another human player. This project leverages advanced algorithms to create a challenging and intelligent gaming experience. The game is designed to combine traditional Checkers gameplay with modern artificial intelligence techniques, ensuring a stimulating environment for players of all skill levels. The game is developed using Python and Pygame, providing a smooth graphical interface that enhances user interaction. Key features include:

- Minimax Algorithm: The AI uses the Minimax algorithm to evaluate the game tree and select optimal moves. The algorithm recursively analyses moves and countermoves, ensuring the AI makes the best decision based on the current game state. Simulating the game's potential outcomes helps the AI predict the opponent's moves and strategise accordingly.

- AI vs. Human Gameplay: The game supports gameplay between two human players or a human against the AI. The AI opponent is designed to simulate intelligent, human-like decision-making, making the game enjoyable and challenging for players of varying expertise.

- Deep Copy and Simulation: To ensure the AI can explore potential moves without affecting the current game state, the game uses deep copy and simulation. Creating a copy of the game board for each simulated move allows the AI to evaluate multiple possibilities without interfering with the live game, leading to more accurate and thoughtful decision-making.

- Game Visualisation with Pygame: The graphical interface built with Pygame provides smooth, real-time gameplay. It features visual highlights that indicate valid moves, making it easier for players to understand the game's current state. The user-friendly interface improves the gaming experience by providing precise and engaging visuals.

- Customisable Difficulty: The AI's difficulty can be tailored by adjusting the depth of the Minimax algorithm. Players can challenge themselves by increasing the algorithm's depth for more complex and strategic gameplay or reducing it for a more relaxed, beginner-friendly experience.

- GTTS Integration: The project integrates the Google Text-to-Speech (GTTS) API, adding a layer of interactivity. The game can announce events or outcomes verbally, such as capturing pieces or determining the winner, creating a more immersive experience for the player.

1.1 Problem Domain

Checkers, or Draughts, is a classic board game that involves strategic thinking, planning, and decision-making. The problem domain here is the simulation of the Checkers game, which is traditionally played on an 8x8 grid with two players controlling coloured pieces. Players aim to capture or block the opponent's pieces by jumping over them. The problem domain includes developing an algorithmic solution to simulate a Checkers game with player input and AI decision-making while rendering the game's visual aspects. The main challenge is designing a system where an AI can play competently and where the user interface is engaging and interactive.

1.2 Aim

This project aims to develop a fully functional and interactive Checkers game that allows players to either compete against each other or face off against an AI-powered opponent. The primary objective is to create a system where the AI uses advanced game theory algorithms, such as the Minimax algorithm with alpha-beta pruning, to simulate intelligent decision-making and offer a challenging and engaging experience. Using Python and the Pygame library, the game will feature a visually appealing interface and intuitive user controls.

Additionally, the project aims to provide insights into the practical application of AI in real-time gameplay, particularly in decision-making and strategic planning. By integrating customisable difficulty levels, the game seeks to cater to players of various skill levels, allowing them to adjust the challenge according to their preferences. Ultimately, this project strives to combine entertainment with a learning platform for understanding AI algorithms in a competitive environment.

1.3 Statement of the Problem

The problem is to design a Checkers game simulation that enables two players to compete against each other on a digital platform. One of the players can either be human-controlled or AI-controlled. The AI must be capable of making intelligent decisions by evaluating possible moves and considering the best strategy to win, using techniques such as the Minimax algorithm. The game should handle piece movement captures and promote pieces to kings when they reach the last row. The system must also display the board, handle user interactions, and provide real-time feedback to the players. Additionally, the game should be able to detect the winner when one player captures all of the opponent's pieces or blocks their movements.

1.4 Objective of the project work

The primary objective of the project is to create a Checkers game simulation, where two players or a player and AI can effectively play. Specific goals include:

- Create a visually appealing and interactive interface where players can view the board, select pieces, and make moves.
- Implement a mode where two human players can play against each other by taking turns, selecting pieces, and making moves on the board.
- Implement an AI opponent using the Minimax algorithm, enabling a human player to compete against the AI.
- Create algorithms to handle piece movement, capturing, and promoting pieces to kings.
- Implement an evaluation system to determine when a player has won the game based on the rules of Checkers.
- Implement the Minimax algorithm with alpha-beta pruning to evaluate and decide the AI's moves, making the AI challenging to play against.
- Design the system with modularity in mind, allowing for easy expansion and modification of game rules, AI strategies, and user interfaces.

1.5 Applications

This project is designed to serve multiple purposes, combining entertainment, education, and research opportunities. It provides a platform for both strategic gameplay and the exploration of artificial intelligence techniques, specifically in the context of game theory. A few applications are mentioned here:

- The game can be an entertainment tool for people who enjoy strategic board games, offering a digital platform for playing Checkers.
- This project is an example of implementing AI techniques like the Minimax algorithm in game development, offering opportunities for further research in decision-making and game theory.
- The game can teach players about strategic thinking, planning, and decision-making, which are important concepts in various fields such as business, finance, and problem-solving.
- The project can be used to study game theory, including AI strategies, decision-making under uncertainty, and competitive interaction between players.
- The game offers insights into the design of user interfaces and interaction patterns that enable smooth communication between players and the game.

1.6 Existing Solution Methods

Existing solution methods for checker games generally fall into two categories: basic AI implementations and more advanced strategies that utilise algorithms like Minimax. These methods vary in their ability to provide a challenging experience, user interface, and overall gameplay dynamics. Some of the standard methods used in existing Checkers games include:

- **Random AI**: One of the simplest AI methods involves the AI making random moves without considering strategy. This approach is easy to implement but fails to offer any real challenge to the player, as the AI does not plan or evaluate the state of the game.
- **Rule-Based AI**: Rule-based systems implement predefined rules for the AI to follow, such as prioritising capturing pieces or avoiding certain moves. While this offers a more structured AI behaviour, it still lacks depth in decision-making and can be easily beaten by experienced players.

- **Minimax Algorithm (Without Pruning)**: The Minimax algorithm is a decision-making process that evaluates all possible moves by considering the potential consequences of each move. The algorithm recursively simulates moves from both the AI and the opponent's perspective. However, without Pruning, the algorithm can be computationally expensive and inefficient, especially for large game trees, leading to slow decision-making.
- **Heuristic Evaluation**: To improve the performance of the AI, some checker games use heuristic evaluation functions to rank game states. These functions assign a score to different board configurations based on factors like piece count, control of the board, and potential for future moves. This provides a way to evaluate the best moves without exploring the entire game tree.
- **Monte Carlo Tree Search (MCTS)**: Some modern implementations use MCTS, an algorithm that simulates many possible game outcomes by making random moves and evaluating the results. While MCTS can be effective for some games, its application to Checkers is less common than the Minimax algorithm.
- **User Interface and Graphics**: Many Checkers games use graphical libraries such as Pygame or Unity to render the board, pieces, and animations. These libraries allow developers to create visually appealing and interactive user interfaces that improve the gaming experience. However, the complexity of the user interface and graphics can vary widely from one solution to another.

Each method has strengths and weaknesses, with more straightforward approaches offering easier implementation but limited challenge. At the same time, more advanced algorithms provide more vigorous AI opponents but may require more computational power.

1.7 Limitations

Like many others, this project may have specific limitations that can impact the overall user experience and gameplay. These limitations are common in game development and can provide insights into areas for further improvement and enhancement. Some of the potential limitations include:

- **Limited AI Competence**: Although AI uses the Minimax algorithm, its strategic

decision-making may still be limited compared to highly advanced AI systems. The AI might struggle against more experienced human players, especially at lower difficulty levels.

- **Performance Issues**: As the Minimax algorithm with alpha-beta pruning requires significant computation, the game may experience occasional lag or slow decision-making during more complex gameplay, particularly at higher difficulty levels.
- **Lack of Realism**: While the game provides a graphical interface through Pygame, the visuals and sound effects may be rudimentary compared to commercial games. The lack of advanced animations or sound effects may reduce the overall immersion.
- **Non-Intuitive Interface**: The game interface may not be as polished as some commercial products. New players could struggle to grasp the game's mechanics, especially with no tutorials or instructional guides.
- **Single Game Mode**: The game may offer a limited range of gameplay modes, with some versions being restricted to single-player versus AI or two-player modes. There might not be options for online multiplayer or advanced modes of play.

1.8 Proposed Solutions

The proposed solution is to develop an interactive Checkers game using the Pygame library in Python, integrating a competent AI opponent that uses the Minimax algorithm with alpha-beta pruning to select moves. The system should allow for both human vs human and human vs AI gameplay modes. The user interface will include graphical elements such as the checkered board, pieces of different colours, and visual indicators for valid moves and captures. The AI will make decisions based on a depth-limited Minimax search, considering different board states and selecting moves that optimise its chances of winning. The system will also include features like piece promotion to kings and an end-game detection system that identifies when a player wins. Additionally, the game will be modular, enabling further enhancements such as supporting different difficulty levels for the AI, multiplayer over a network, or adding sound effects and animations for a more engaging experience. By creating a flexible and engaging Checkers game, this solution aims to provide a fun, interactive, and strategic experience for players of all levels.

CHAPTER 2
LITERATURE SURVEY

2.1 Introuction

The literature survey conducted for this project encompasses an exploration of existing research, tools, and techniques relevant to the development of a **Checkers game using Pygame and Artificial Intelligence (AI)**. Checkers, a classic strategy game, has been a subject of interest in both game development and AI research due to its relatively simple rules and deep strategic elements. By utilizing **Pygame**, a Python library designed for creating 2D games, developers can efficiently build the game interface and handle user interactions. On the other hand, the integration of **AI** techniques such as **Minimax** and **Reinforcement Learning** enables the creation of intelligent opponents capable of challenging human players.

2.2 Literature Survey

2.2.1 Carter, E., & Dyer, M. (2014). *Game Development with Python and Pygame*. Wiley.
A comprehensive guide to game development with Python and PyGame, providing insights into creating games where AI can be implemented.

2.2.2 Winder, J. (2018). *Python Game Programming by Example*. Packt Publishing.
A practical book showing how to build games with PyGame and integrate AI for game mechanics.

2.2.3 Knuth, D. E., & Moore, R. W. (1975). *An Analysis of Alpha-Beta Pruning*. Artificial Intelligence, 6(4), 293-326.
This paper formalizes the concept of Alpha-Beta pruning, an optimization to Min-Max, significantly improving its performance.

2.2.4 Shannon, C. E. (1950). *Programming a Computer for Playing Chess*. Philosophical Magazine, 41(7), 256-275.
One of the earliest discussions of the Min-Max algorithm in the context of game-playing AI.

2.2.5 McCarthy, J. (1956). *Proposal for the Dartmouth Summer Research Project on Artificial Intelligence*. AI Magazine, 27(4), 12-14

McCarthy's introduction of **LISP**, the programming language he developed, became the cornerstone for AI programming. His proposal for the Dartmouth conference is considered the birth of the AI field itself.

2.2.6 Turing, A. (1950). *Computing Machinery and Intelligence*. Mind, 59(236), 433-460.
Turing, A. (1936). *On Computable Numbers, with an Application to the Entscheidungsproblem*. Proceedings of the London Mathematical Society, 2(1), 230-265.
This foundational paper by Turing outlined the concept of a universal machine, now known as the **Turing Machine**, which laid the basis for modern computing and AI.

2.3 PyGame

2.3.1 Introduction to PyGame

PyGame is a cross-platform library for Python that simplifies game development, particularly for 2D games. It provides easy access to multimedia functionalities like graphics, sound, and event handling, making it a great tool for both beginners and professionals. It leverages the **Simple DirectMedia Layer (SDL)** for low-level multimedia operations, allowing developers to focus on game logic and design.

2.3.2 Key Features of PyGame

- **Cross-Platform Compatibility:**
  Supports Windows, macOS, and Linux for multi-platform game development.
- **2D Graphics Handling:**
  Simplifies drawing shapes, sprite animations, and image manipulation for 2D games.
- **Sound and Music Support:**
  Allows loading and playing sound effects and background music in formats like WAV, MP3, and OGG.

- **Event Handling:**

  Manages user input via keyboards, mice, and other devices for interactive gameplay.

- **Educational Use:**

  Its ease of use and Python integration make it ideal for teaching game programming basics.

2.3.3 How PyGame has changed the Gaming Industry

- **Lowered Barriers for Game Development**

  PyGame has made game development accessible to a wider audience, particularly hobbyists and beginner developers, by offering a simpler alternative to larger game engines.

- **Fostering Indie Game Development**

  By offering a lightweight and free platform, PyGame has contributed to the growth of indie game development, allowing smaller studios and individual developers to create games.

- **Encouraged Educational Use**

  It's widely used in educational settings, helping students learn programming concepts through interactive game development, and providing an easy entry point for aspiring game designers.

- **Rapid Prototyping**

  PyGame allows for quick prototyping of game ideas, enabling developers to test and iterate on gameplay concepts faster than with more complex engines.

- **Open-Source and Community Driven**

  As an open-source library, PyGame has fostered a community of developers who contribute to its improvement, making it an adaptable tool for a variety of game development projects.

CHAPTER 3:
REQUIREMENTS

3.1 Platform Compatibility:

- The game is cross-platform and can run on:
- Windows (Windows 7 and above).
- macOS (10.9 and above).
- Linux-based systems (Ubuntu, Fedora, etc.).
- The game should run with no platform-specific adjustments, as Python and Pygame are cross-platform tools.

3.2 Hardware Requirements:

- A computer with at least 2 GB of RAM.
- Processor: Dual-core or higher (preferably Intel i3 or AMD equivalent).
- 500 MB of free storage for the game and its dependencies.
- Graphics card with basic 2D rendering support (integrated graphics are sufficient for this project).
- Input devices: Keyboard and Mouse for user interaction.

3.3 Software Requirements:

- Python (version 3.6 or higher) as the primary programming language.
- Pygame library for graphical rendering and game logic.
- Text-to-Speech (gTTS) library for adding audio output (optional).
- A code editor (e.g., Visual Studio Code, PyCharm, or Sublime Text) for writing and
- editing Python code.
- Python package manager (pip) for installing dependencies.

3.4 Tools and Frameworks:

- **Pygame**: Used for graphical rendering, event handling, and real-time game interaction.

- **gTTS (Google Text-to-Speech)**: An optional library for converting text to speech, adding sound-based feedback to the game.
- **Minimax Algorithm with Alpha-Beta Pruning**: Implemented to provide AI decision-making for the Checkers game.
- **Python**: The main programming language used for implementing the logic and gameplay mechanics.

3.5 Internet Connectivity:

- **Offline Functionality**: The game does not require constant internet connectivity and can be played offline. However, some features such as downloading or updating dependencies (libraries) will require an internet connection during the setup phase.
- **Online Mode (Optional)**: If an online mode were to be implemented (for multiplayer against others), an internet connection would be necessary for player matchmaking and data exchange.
- **gTTS Library**: This requires an internet connection for generating speech, as it uses Google's Text-to-Speech service for audio output

CHAPTER 4:

CODE

4.1 Code Implementation:

I Main Code

```
import pygame
from checker.constants import WIDTH, HEIGHT, SQUARE_SIZE, PURPLE, YELLOW, BLACK, FONT_SIZE
from checker.game import Game
from minimax.algorithm import minimax
import gtts
import playsound as py
from os import path
import os

def speak(msg):
    vd = gtts.gTTS(msg, lang='en-au')
    if path.exists('temp_audio.mp3'):
        os.remove('temp_audio.mp3')
    vd.save('temp_audio.mp3')
    py.playsound('temp_audio.mp3')

FPS = 60
WIN = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption('Checkers With AI')

def get_row_col_from_mouse(pos):
    x, y = pos
    row = y // SQUARE_SIZE
    col = x // SQUARE_SIZE
    return row, col

def display_winner(winner, time_taken):
```

```
        font = pygame.font.SysFont('Times New Roman', FONT_SIZE)
        winner_text = font.render(f"Winner: {winner}", True, BLACK)
        time_text = font.render(f"Time Taken: {time_taken}s", True, BLACK)
        WIN.blit(winner_text, (WIDTH // 2 - winner_text.get_width() // 2, HEIGHT // 2 40))
        WIN.blit(time_text, (WIDTH // 2 - time_text.get_width() // 2, HEIGHT // 2 + 10))
        pygame.display.update()

    def winner(self):
        human_pieces = self.get_all_pieces(PURPLE)
        if not human_pieces or not self.can_move(PURPLE):
            if not self.can_move(YELLOW):
                return "Draw"
            return "AI"

        ai_pieces = self.get_all_pieces(YELLOW)
        if not ai_pieces or not self.can_move(YELLOW):
            if not self.can_move(PURPLE):
                return "Draw"
            return "Human"

        if len(human_pieces) == 1 and len(ai_pieces) == 1 and not
    self.can_move(PURPLE) and not self.can_move(YELLOW):
            return "Draw"
        if len(human_pieces) == 1 and not self.can_move(PURPLE):
            return "Draw"
        return None

    def main():
        pygame.font.init()
        speak("Welcome to Checkers with AI. Do you want to listen to the instructions?")
        response = input("Do you want to listen to the instructions? (yes/no):
    ").strip().lower()
        if response == 'yes':
```

```
        instructions = (
            "The game is played between a Human and an AI. "
            "The goal is to capture all opponent's pieces or block them from making any
move. "
            "Click on your pieces and then select the destination to make a move. "
            "AI plays as Yellow and you play as Purple. Please proceed to the pygame
window to play your game! Good luck!"
        )
        speak(instructions)
    else:
        instructions = (
            "Very Well. Please proceed to the pygame window to play your game. Good
Luck!"
        )
        speak(instructions)

    run = True
    clock = pygame.time.Clock()
    game = Game(WIN)
    start_time = pygame.time.get_ticks()

    while run:
        clock.tick(FPS)
        if game.turn == YELLOW:
            value, new_board = minimax(game.get_board(), 4, YELLOW, game)
            game.ai_move(new_board)

        winner = game.winner()
        if winner is not None:
            time_taken = (pygame.time.get_ticks() - start_time) // 1000
            display_winner(winner, time_taken)
            pygame.time.delay(10000)
            run = False
```

```
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                run = False
            if event.type == pygame.MOUSEBUTTONDOWN:
                pos = pygame.mouse.get_pos()
                row, col = get_row_col_from_mouse(pos)
                game.select(row, col)

        game.update()

    pygame.quit()

if _name_ == "_main_":
    main()
```

II. constants.py (Code to define all the constants used throughout the game)

```
import pygame

WIDTH, HEIGHT = 800, 800
ROWS, COLS = 8, 8
SQUARE_SIZE = WIDTH//COLS
PURPLE=(222, 111, 161)
FONT_SIZE=30
RED = (255, 0, 0)
PINK=(225, 204, 229)
WHITE = (255, 255, 255)
YELLOW=(255,204,0)
BLACK = (0, 0, 0)
BLUE = (0, 0, 255)
GREY = (225, 204, 229)

CROWN = pygame.transform.scale(pygame.image.load('assets/crown.png'), (44, 25))
```

III. game.py (This class encapsulates the game logic, handling piece selection, movement, turn changes, and AI interactions)

```
import pygame
from .constants import PINK, YELLOW, BLUE, SQUARE_SIZE, PURPLE
from checker.board import Board

class Game:
    def _init_(self, win):
        self._init()
        self.win = win

    def update(self):
        self.board.draw(self.win)
        self.draw_valid_moves(self.valid_moves)
        pygame.display.update()

    def _init(self):
        self.selected = None
        self.board = Board()
        self.turn = PURPLE
        self.valid_moves = {}

    def winner(self):
        return self.board.winner()
    def reset(self):
        self._init()
    def select(self, row, col):
        if self.selected:
            result = self._move(row, col)
            if not result:
                self.selected = None
                self.select(row, col)

        piece = self.board.get_piece(row, col)
```

```
        if piece != 0 and piece.color == self.turn:
            self.selected = piece
            self.valid_moves = self.board.get_valid_moves(piece)
            return True
        return False

    def _move(self, row, col):
        piece = self.board.get_piece(row, col)
        if self.selected and piece == 0 and (row, col) in self.valid_moves:
            self.board.move(self.selected, row, col)
            skipped = self.valid_moves[(row, col)]
            if skipped:
                self.board.remove(skipped)
            self.change_turn()
        else:
            return False
        return True

    def draw_valid_moves(self, moves):
        for move in moves:
            row, col = move
            pygame.draw.circle(self.win, BLUE, (col * SQUARE_SIZE +
SQUARE_SIZE // 2,
    row * SQUARE_SIZE + SQUARE_SIZE // 2), 15)

    def change_turn(self):
        self.valid_moves = {}
        if self.turn == PURPLE:
            self.turn = YELLOW
        else:
            self.turn = PURPLE

    def get_board(self):
        return self.board
```

```
    def ai_move(self, board):
        self.board = board
        self.change_turn()
```

IV. piece.py (This class defines the movement of pieces, structure and design of the piece

```
from .constants import PURPLE, YELLOW, SQUARE_SIZE, GREY, CROWN
import pygame

class Piece:
    PADDING = 15
    OUTLINE = 5

    def _init_(self, row, col, color):
        self.row = row
        self.col = col
        self.color = color
        self.king = False
        self.x = 0
        self.y = 0
        self.calc_pos()

    def calc_pos(self):
        self.x = SQUARE_SIZE * self.col + SQUARE_SIZE // 2
        self.y = SQUARE_SIZE * self.row + SQUARE_SIZE // 2

    def make_king(self):
        self.king = True

    def draw(self, win):
        radius = SQUARE_SIZE // 2 - self.PADDING
        pygame.draw.circle(win, GREY, (self.x, self.y), radius + self.OUTLINE)
        pygame.draw.circle(win, self.color, (self.x, self.y), radius)
```

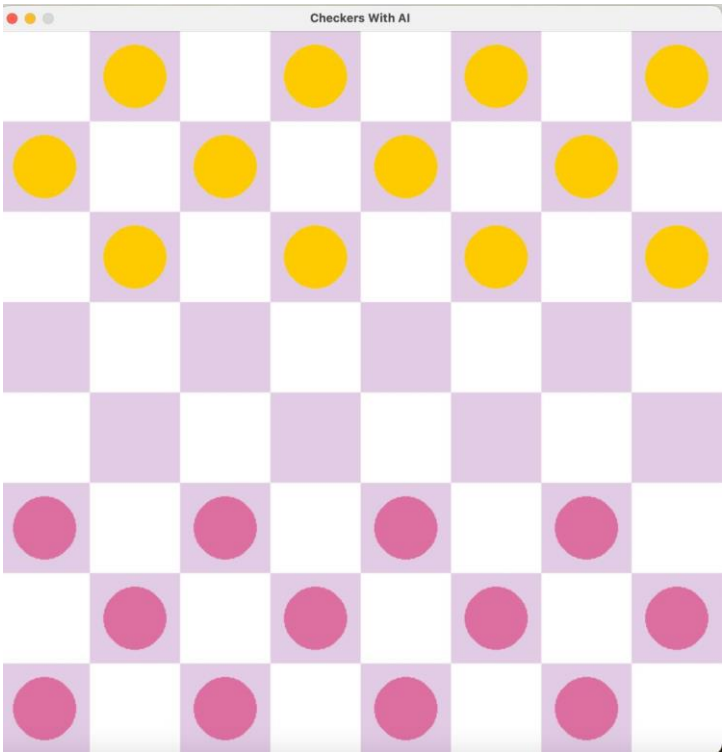
```
        if self.king:
            win.blit(CROWN, (self.x - CROWN.get_width() // 2, self.y -
CROWN.get_height() // 2))

    def move(self, row, col):
        self.row = row
        self.col = col
        self.calc_pos()
    def _repr_(self):
        return str(self.color)
```

V. algorithm.py (This class contains the algorithm implemented to facilitate the computer (AI) to make smart, informed and valid moves)

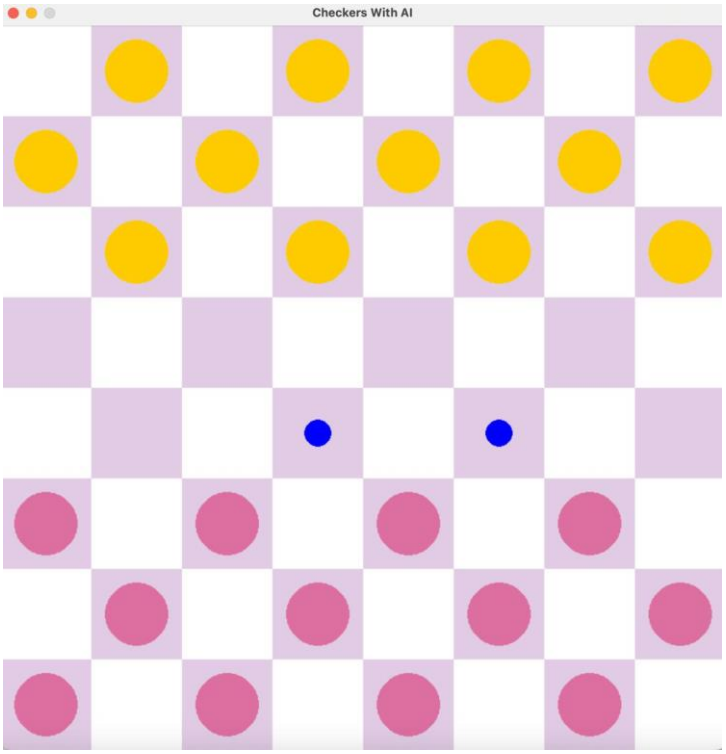
```
from copy import deepcopy
import pygame

PURPLE = (222, 111, 161)
YELLOW = (255, 204, 0)

def minimax(position, depth, max_player, game):
    if depth == 0 or position.winner() is not None:
        return position.evaluate(), position

    if max_player:
        maxEval = float('-inf')
        best_move = None
        for move in get_all_moves(position, YELLOW, 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, PURPLE, 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():
            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(10000)
```

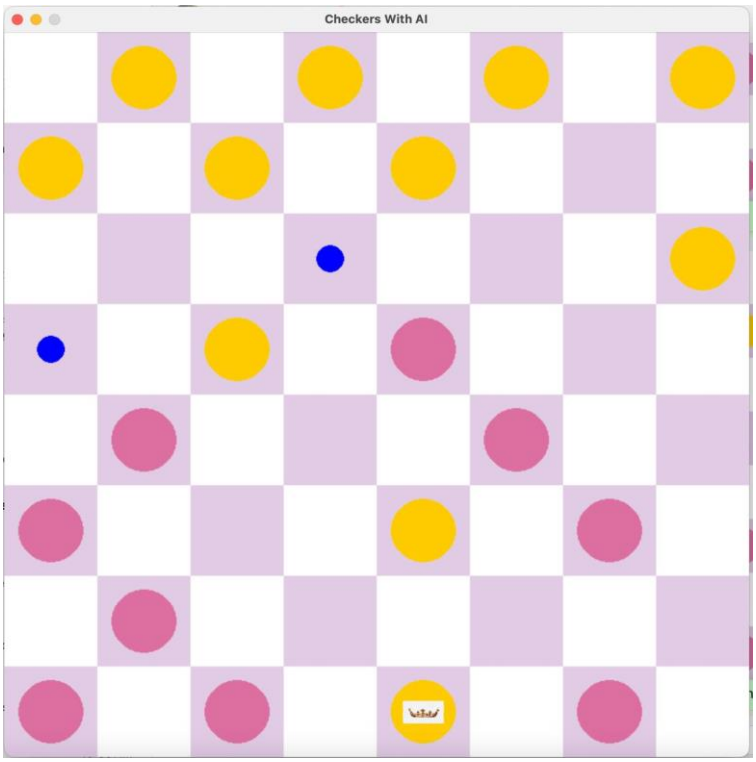
4.2 Game Snapshots:



I Board



II. A piece selected, and the possible moves shown



III. Mid game. A piece is also ‘kinged’ when it reaches the other end of the board, and is denoted by a ‘crown’ which appears on the piece.

IV. Game ends when there is no possible way for the humans (denoted by pink pieces) to win. The time taken, along with the winner (denoted here by the RGB shade chosen by the user in the beginning of the game) is displayed

CONCLUSION

In conclusion, developing the Checkers AI Game using Python and Pygame offers a robust platform for casual players and those interested in studying the application of AI in strategic gameplay. The game provides players with a challenging and dynamic experience by leveraging the Minimax algorithm with alpha-beta pruning. The project successfully combines interactive graphics, intelligent decision-making, and user-friendly features, ensuring enjoyment and education. While addressing some common limitations found in existing Checkers games, such as limited AI complexity and lack of immersion, this project provides a significant step forward in improving the digital Checkers experience. Additionally, it showcases how AI algorithms and game theory can be practically applied in real-time gameplay scenarios. Ultimately, this project is an engaging entertainment tool and a valuable resource for further research in AI, human-computer interaction, and strategic decision-making.

REFERENCES

Websites

1. https://www.pygame.org/docs/
2. https://devdocs.io/pygame/
3. https://docs.python.org/3/
4. https://gtts.readthedocs.io/en/latest/
5. https://docs.python.org/3/library/copy.html
6. https://gameprogrammingpatterns.com/

Literary

1. **Russell, S. J., & Norvig, P. (2016).** *Artificial Intelligence: A Modern Approach* (3rd ed.). Pearson.
2. **Nisan, N., Roughgarden, T., Tardos, E., & Vazirani, V. V. (2007).** *Algorithmic Game Theory*. Cambridge University Press.
3. **Preece, J., Rogers, Y., & Sharp, H. (2015).** *Interaction Design: Beyond Human-Computer Interaction* (4th ed.). Wiley.