# AI BASED ACTION SELECTION CHECKERS GAME

**A Project Report**

Submitted in partial fulfillment of the

Requirements for the award of the Degree of

**BACHELOR OF SCIENCE (INFORMATION TECHNOLOGY)**

**By**

Weqar Fatima Sayed Abedi

UID : 18BIT046

**Under the esteemed guidance of**

**Ms. Bertilla Fernandes**

**Assistant Professor**

**DEPARTMENT OF INFORMATION TECHNOLOGY**

**JAI HIND COLLEGE**

**(Autonomous)**

**MUMBAI, 400020**

**MAHARASHTRA 2020-21**

**JAI HIND COLLEGE**

***(Autonomous)***

**MUMBAI-MAHARASHTRA-400020**

**DEPARTMENT OF INFORMATION TECHNOLOGY**

# CERTIFICATE

This is to certify that the project entitled, "**AI Based Action Selection Checkers Game**", is bonafied work of **WEQAR FATIMA SAYED ABEDI** bearing UID / Roll No. **: (18BIT046)** submitted in partial fulfillment of the requirements for the award of degree of BACHELOR OF SCIENCE in INFORMATION TECHNOLOGY from Jai Hind College Autonomous (University of Mumbai).

**Internal Guide** **Coordinator**

**External Examiner**

**Date:** **College Seal**

# ACKNOWLEDGEMENT

I would like to express my thanks to the people who have helped me most throughout my project. I am grateful to my Prof. Ms. Bertilla Fernandes for nonstop support for the project. I can’t say thank you enough for her tremendous support and help.

I owe my deep gratitude to our HOD of Information Technology Department Mr. Wilson Rao who took keen interest on our project work and guided us all along, till the completion of our project work by providing all the necessary information for developing a good system.

At last but not the least I want to thank all of my friends who helped/treasured me out in completing the project, where they all exchanged their own interesting ideas, thoughts and made this possible to complete my project with all accurate information. I wish to thank my parents for their personal support or attention who inspired/encouraged me to go my own way.

# DECLARATION

I hereby declare that the project entitled**, " AI Based Action Selection Checkers Game"** done at **Jai Hind College,** has not been in any case duplicated to submit to any other university for the award of any degree. To the best of my knowledge other than me, no one has submitted to any other university.

The project is done in partial fulfillment of the requirements for the award of degree of **BACHELOR OF SCIENCE (INFORMATION TECHNOLOGY)** to be submitted as final semester project as part of our curriculum.

**Name and Signature of the Student**

# ABSTRACT

Artificial Intelligence (AI) in the game industry matures and continues to expand due to its capabilities to explore and exploit in bringing gaming to life. Checkers, which is developed by using a Minimax algorithm approach. This is implemented by starting with the user logic and eventually moving forward to implementing the algorithm and training the system until the computer learns how to perform the next move and with what level of smartness. The testing activity ends after the final bug fix and the result shows that the game executes properly as expected. The result indicates that the game meets the objective of the system, which is the game, may learn and explore by itself. Checkers is a very popular game all over the world. The first attempts to build the first English draughts computer program were in the early 1950s. In 2007, it was published that the program “Chinook” was able to “solve” the 8X8 board from all possible positions.

# TABLE OF CONTENTS

## Contents


**1 INTRODUCTION**

**1.1 Background ..... 1**

**1.2 Objectives ..... 2**

**1.3 Purpose, Scope, and Applicability .....**

**1.2.1** Purpose **..... 3**

**1.2.2** Scope **..... 4**

**1.2.3** Applicability **..... 5**

**1.3 Achievements ..... 6**

**1.4 Organization of Report ..... 7**

**2 SURVEY OF TECHNOLOGIES ..... 8**

**3 REQUIREMENTS AND ANALYSIS**

**3.1 Problem Definition ..... 10**

**3.2 Requirement Specification ..... 12**

**3.3 Planning and Scheduling ..... 14**

**3.4 Software and Hardware Requirements ..... 16**

**3.5 Preliminary Product Description ..... 17**

**3.6 Conceptual Models ..... 19**

**4 SYSTEM DESIGN**

**4.1 Basic Modules ..... 32**

**4.3 User Interface Design ..... 34**

**4.4 Security Issues ..... 36**

**4.5 Test Case Design ..... 37**

**5 IMPLEMENTATION AND TESTING**

**5.1 Implementation Approach ..... 40**

**5.2 Coding Details and Code Efficiency ..... 44**

**5.2.1** Code Efficiency **..... 69**

**5.3 Testing Approach .....**

**5.3.1** Unit Testing **..... 70**

**5.3.2** Integration Testing **..... 70**

**5.3.3** System Testing **..... 70**

**5.4 Modification and Improvements ..... 71**

**6 RESULTS AND DISCUSSION**

**6.1 Test Reports ..... 72**

**6.2 User Documentation ..... 77**

**7 CONCLUSION**

**7.1 Conclusion ..... 89**

**7.2 Limitations of the System ..... 90**

**7.3 Future Scope of the Project ..... 91**

# List of Figures


1. Pert Chart ..... 14
2. Gantt Chart ..... 15
3. Event Table ..... 19
4. Use Case Diagram ..... 21
5. Class Diagram ..... 22
6. Object Diagram ..... 23
7. Activity Diagram ..... 24
8. Sequence Diagram ..... 25
9. State Diagram ..... 26
10. Package Diagram ..... 27
11. Deployment Diagram ..... 28
12. Component Diagram ..... 29
13. Data Flow Level 0 Diagram ..... 30
14. Data Flow Level 1 Diagram ..... 31
15. Test case Design ..... 37
16. Main Menu screen ..... 79
17. Board without pieces screen ..... 80
18. Board with pieces ..... 81
19. Human piece Movement ..... 82
20. Human piece Movement ..... 83
21. AI Agent piece movement ..... 84
22. King piece assigned ..... 85
23. King piece attack with triple jump ..... 86
24. King piece attacked ..... 87
25. Normal pieces double jump attack ..... 88
26. King has 4 options to attack ..... 89
27. Human player has triple jump attack ..... 90

# 1 INTRODUCTION

## 1.1 Background

This project seeks to create a new interactive way of playing checkers by combining minimax algorithm and an Artificially Intelligent checkers player. This hybrid game is a combination of a board game and a computer game. The game of checkers is considered a complicated game with 10^20 possible legal positions in the English draughts version (8*8 board) alone. Our approach is to create a computer agent based on the minimax algorithm, together with possible improvements, which is the state of the art in one-on-one games. We start from implementing a basic minimax player with a basic evaluation heuristic, and step by step we improve the pruning by implementing alpha-beta pruning and in addition, we improve our evaluation function to reach better approximation of the value of the position for our computer player. A checkers board will be displayed on the screen. The human player will have their 12 blue pieces on the checkers board screen and the computer's 12 white pieces will be displayed on the screen. When it is human player's turn to play, the player will move one of their own pieces which will be given 2 prior valid positions and then they will choose the desired position. When it is computer's turn to play, computer AI player will calculate an optimal movement and will execute it on the screen. When any one of the pieces of either the human or the computer reaches the opponents side of the game then it will be tagged as the KING. Combining all this functionality will create a completely new interactive way to play Checkers.

## 1.2 Objectives

Checkers is game that is played by everyone irrespective of their age. Since this game is manually made and played at homes. I would be creating this game so that families, friends, or individuals can play it online, without the need of any board or even gathering to play it together. This game will connect people and people will play this game with a computer which will make it even more interesting. An AI concept will be incorporated which will train the computer to make the right decision and make playing checkers fun!

- ✓ This "AI based Action Selection Checkers" game will create the feeling of playing checkers on an actual game board
- ✓ The unique user interface and playing style of this game will make it more enjoyable and easier to play.
- ✓ Implementing the checkers AI code using a minimax algorithm that eliminates the need for an extra.

## 1.3 Purpose, Scope, and Applicability

### 1.3.1 Purpose

The purpose of this game is to create a platform where user's can play the checkers game anywhere and at anytime with an opponent instead of traditional board game at home. The objective of the game is to capture as many discs of the opponent as possible or if the opponent has no valid moves left to make. The game takes place on a checkers board which will be displayed on the screen. There will be 1 user's side on the board and the opponent that is the computer which will be trained by using the Minimax algorithm and tested to verify his capabilities and how optimally he plays the game on the other side of the board.

### 1.3.2 Scope

This game is created using a computer agent based on the minimax algorithm, together with possible improvements, which is the state of the art in one-on-one games. We start from implementing a basic minimax player with a basic evaluation heuristic, and step by step we improve the pruning by implementing alpha-beta pruning and in addition, we improve our evaluation function to reach better approximation of the value of the position for our computer player. A checkers board will be displayed on the screen. The human player will have their 12 blue pieces on the checkers board screen and the computer's 12 white pieces will be displayed on the screen. When it is human player's turn to play, the player will move one of their own pieces which will be given 2 prior valid locations and then they will chose the desired location. When it is computer's turn to play, computer AI player will calculate an optimal movement and will execute it on the screen. When any one of the pieces of either the human or the computer reaches the opponents side of the game then it will be tagged as the KING. Combining all this functionality will create a completely new interactive way to play Checkers.

### 1.3.3 Applicability

This project is beneficial to teach all the kids as well as adults, it can boost the memory recall since a trained computer agent is used thereby making the user to think critically and accurately on how to play the next move, with this it develops the concentration skills, teaches patience and also promotes confident decision making skills. Thus the more the game is played the more the computer agent is being trained and ultimately gains high accuracy and complexity to play and have fun with the user.

## 1.4 Achievements

The project has successfully met its objectives. The system is working. Users can play the game and enjoy using the various functionalities such as attacking the pieces, jumping, double jump, triple jump, king, backward king movements.

## 1.5 Organization of reports

The project name is AI based Action Selection Checkers Game; this project can be played by users irrespective of their age. The following chapters of this project consist of the technologies used, Requirement analysis, Designing, Implementation and testing phases of the project. The first phase consists of the technologies that are required to build the project. Hence a survey was carried out and this project includes both client side and server side and a middleware. I am using Visual Studio code as the editor and python as the programming language. The next phase is requirement analysis which defines the requirements specification of the project which includes both software and hardware requirements. It includes planning, scheduling, product description. The next phase is the Design phase where each module, align with its functionality is explained in detail. After this phase is the important phase called the implementation phase. The implementation phase is time consuming and complex to think of. It includes a design and how to program the functionalities or the modules mentioned above. The testing phase includes testing of the functionalities and implementation in every way. After all these phases are completed, the project is finally implemented, which is checked and concluded to have met its objectives successfully.

# 2 SURVEY OF TECHNOLOGIES

## Hardware

- **Laptop/PC**

  The user can access this website through either a laptop or a PC.

- **Mobile Devices**

  This website can be viewed in any handheld mobile phone through the link of the website given.

- **Processor**

  Intel Processor i5

- **Memory**

  RAM - 4GB or More – ROM - 40GB or More

## Software

- **Frontend:**

  Python: This is a simple, object-oriented, high level programming language with dynamic semantics. Python supports modules and packages, which makes the code modular and easy to write and understand.

- **Middleware**

  Pygame: By using the pygame module, I will create the logic and graphics of the checkers games without worrying about the backend complexities required for working with video and audio.

- **Backend**

  Minimax library: focusing on the implementation of Artificial Intelligence, minimax algorithm is the best use for board games and

# 3 REQUIREMENTS AND ANALYSIS

## 3.1 Problem Definition

### 3.1.1 Existing System

Currently on the internet there are a few games provided but mostly they all come with the traditional approach where there are 2 users and a competitiveness between them without any complexity of a non user concept were the user does not have to have a opponent in person but a trained AI agent will be alongside playing with the user with much more interest and fun. Many previous technologies have implemented the heuristic method into the game industry.

### 3.1.2 Proposed System

In the proposed system I will be developing a game that uses an Alpha Beta pruning method and an algorithm called as “Minimax”. Minimax is a recursive algorithm used to select an optimal move for a player provided the other player plays optimally. The main idea of this project is to develop a Checkers game that integrated with artificial intelligence and the ability to act as the player’s opponent using a heuristic approach.

Each player takes their turn by moving a piece. Pieces are always diagonally and can be moved in the following ways:

1. Diagonally in the forward direction (towards the opponent) to the next empty square.
2. If there is one of the opponent’s pieces next to a piece and an empty space on the other side, you jump your opponent and remove their piece. You can do multiple jumps if they are lines up in the forward direction

There are King Pieces, the last row is called the King row. If you get a piece across the board to the opponents king row, that piece becomes a king. King pieces can move in both directions, forward and backward, perform double and triple jumps.

## 3.2 Requirement Specification

### 3.2.1 Functional Requirements

- System shall be able to play against the single player
- System shall be able to keep check of the valid moves
- System shall be able to keep check of the invalid moves
- System shall be able to tell at the end that which player as won the match
- System shall be able to keep track that currently which player has turn either the user or the computer itself
- System shall be able not to allow the player to take the wrong moves

### 3.2.2 Non-Functional Requirements

- **Software Quality Attributes**

  - Availability
    - This game will be on server side and is made available at all times so that the users don't have to face any problem while playing Checkers game.

  - Correctness
    - Accurate services will be provided according to the user requirement.

  - Maintainability
    - The system should keep track of all the user activities.

  - Usability
    - The system should satisfy a maximum number of users interest.

  - Cost
    - There is no installation cost since free and open source technologies are used. The only cost that might incur is the hardware cost such as server.

## 3.3 Planning and Scheduling

### PERT CHART


**Figure 1: Pert Chart**

**GANTT CHART**

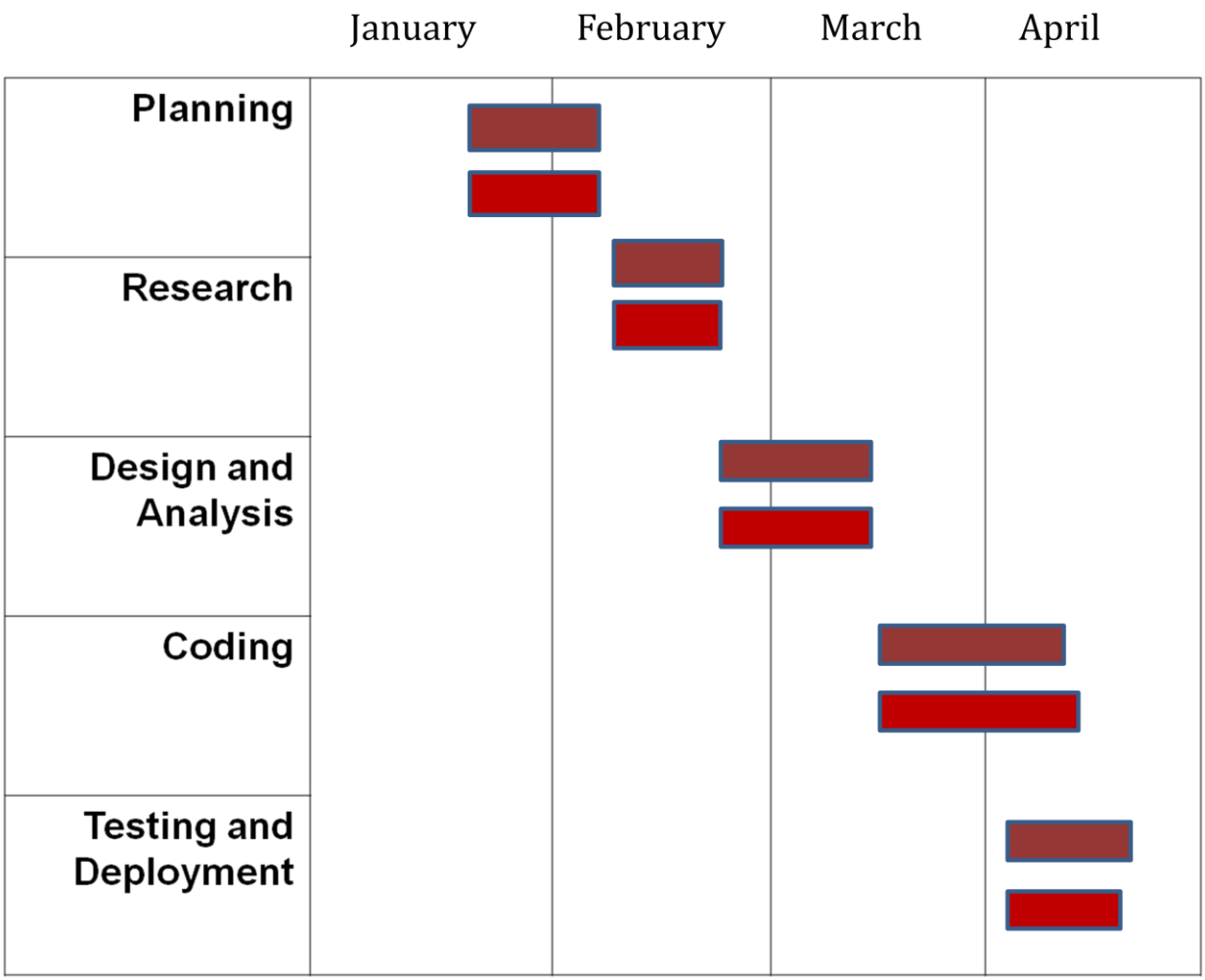



**Figure 2: Gantt chart**

## 3.4 Software and Hardware Requirements

- **Software Requirements:**

  - Python
  - Minimax algorithm
  - Pygame

- **Hardware Requirements:**

  - RAM -4GB or More.
  - ROM -4GB or More.
  - Intel Processor i5

## 3.5 Preliminary Product Description.

i. The checkers board designing:

Designing an attractive GUI board of checkers that will be displayed on the screen which will include:

a. The size of the board.
b. Assigning the coloured pieces to the computer and the user.
c. On the click of each piece it should move.
d. Giving 2 valid positions in prior for performing the best move.

ii. Piece placement:

The pieces placement will do accordingly, the user pieces on one side of the board and the computer pieces on the other side of the board.

iii. Main game:

This module will be a main programming core, which will have the functionality to change turns, move the pieces, the valid moves, etc.

iv. User/Human Player:

The users turn to play the game. Since there will be an opponent called the computer with whom the user will play against. The system will give ideas to perform the next move for helping the user. Human/user Player extends the Player main board class. It offers various verification functions and methods that are used to force human player to play by rules.

v. Jumps, Multi jumps and Triple jumps:

The pieces can move diagonally wither one piece ahead, 2 pieces ahead and also 3 pieces ahead. This uses a complex algorithm in which it first checks whether the piece in front to jump is of the opponent's piece or itself. With the detection of colours and position the system will give the valid moves hint and then the user can move accordingly.

vi. AI/Computer Player

Like previous class extends Player main board function this will extend the algorithm function. There is a constant level of competitiveness given to the system, as the algorithm trains it will be able to train itself and become harder, thus its using real AI algorithm. The algorithm's name is Minimax

vii. Computer coding:

For the computers turn to play, an AI algorithm known as the ”Minimax” algorithm will be incorporated which will be trained in order for the computer to smartly perform the next move.

## 3.6 Conceptual Modules

### 3.6.1 Event Table

| Sr.No | Event | Trigger | Source | Use Case | Response | Destination |
|---|---|---|---|---|---|---|
| 1. | Piece Movement | Piece is played and moved. | Board | Play move | Piece is moved to the next position. | Board |
| 2. | King Position | Piece placed at the end of the target. | Board | Detects the piece and assigns it King. | King is shown at the piece placed at the end target | Board |
| 3. | Valid Moves | 2valid moves are given. | Board | 2 valid moves which are taken as proper moves. | User takes help of any one of the valid move | Board |
| 4. | Jumps | Single jump is performed to a valid square | Board | Piece movements according to valid | Piece is moved to the valid square with a single jump. | Board |
| 5. | Play | Play the turn | Board | Play the game | Playing the move | Board |

| | | | | | | |
|---|---|---|---|---|---|---|
| 6. | Winner | One of the 2 players has won the game. | Board | The player's piece who has reached the maximum on the opponent's team or has the most pieces on board has Won the game. | The Player or Agent has won | Board |
| 7. | Quit | Close the game. | Board | Quit the game | Quit the game | Board |

**Figure 3: Event Table**

**USE CASE DIAGRAM**

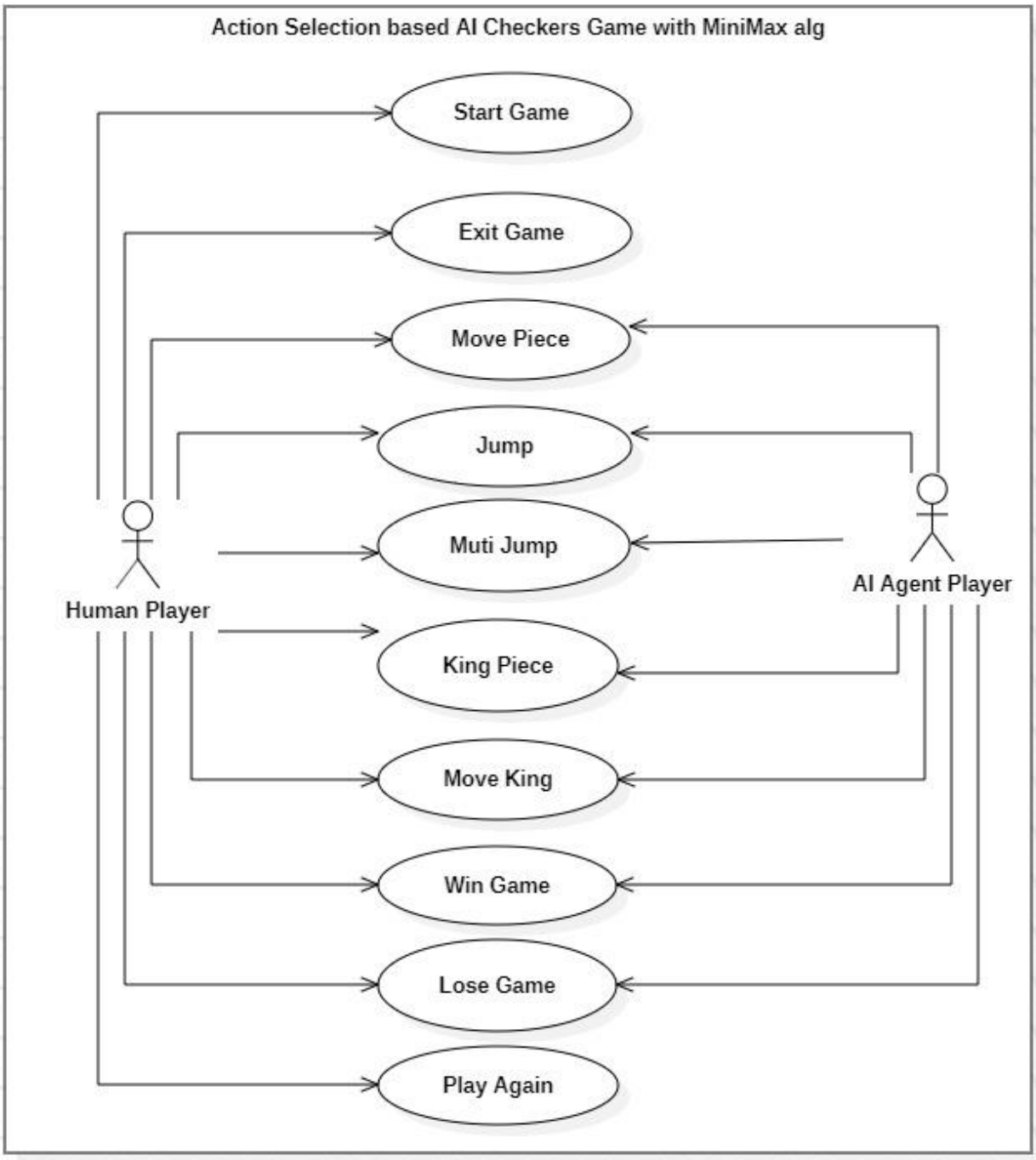



**Figure.4 USE CASE DIAGRAM**

## CLASS DIAGRAM

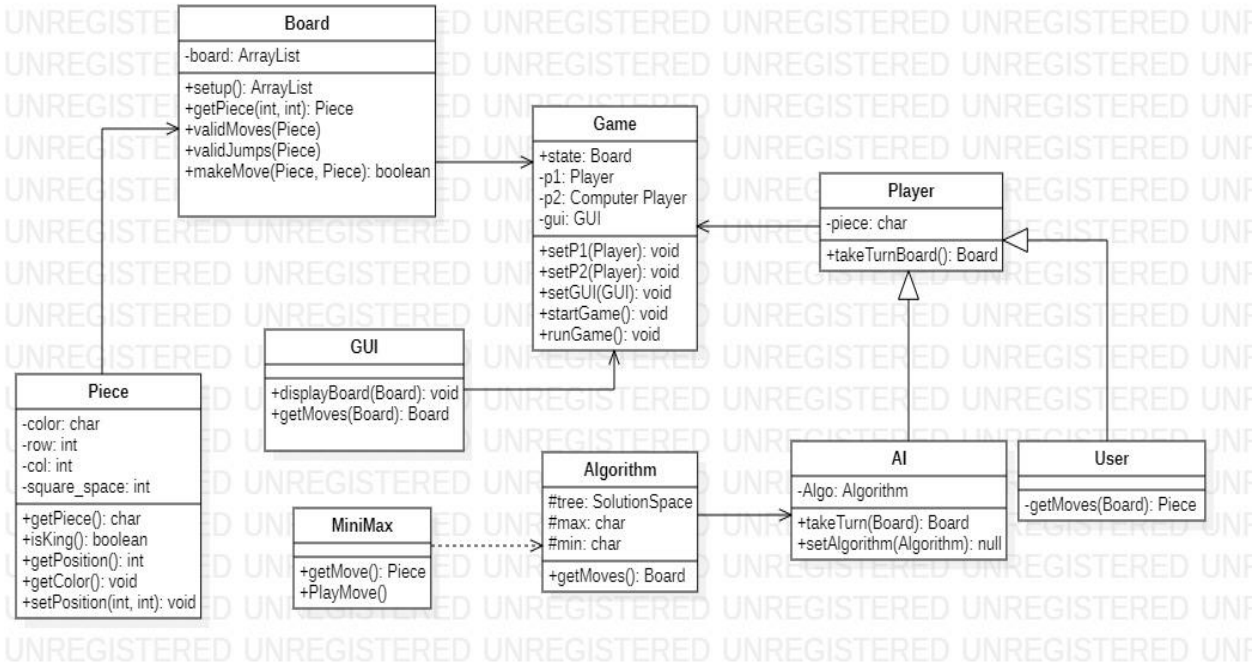



**Figure.5 CLASS DIAGRAM**

**OBJECT DIAGRAM**

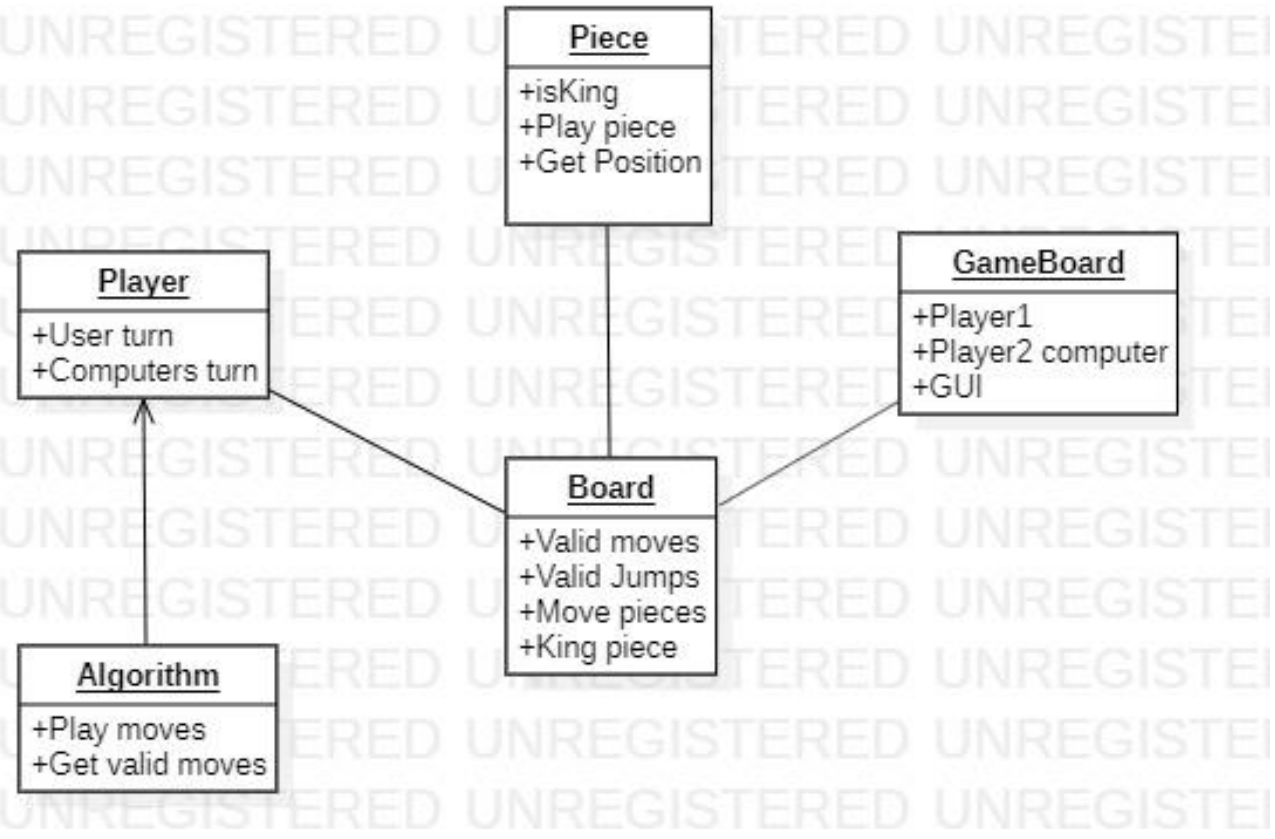



**Figure.6 OBJECT DIAGRAM**

## ACTIVITY DIAGRAM

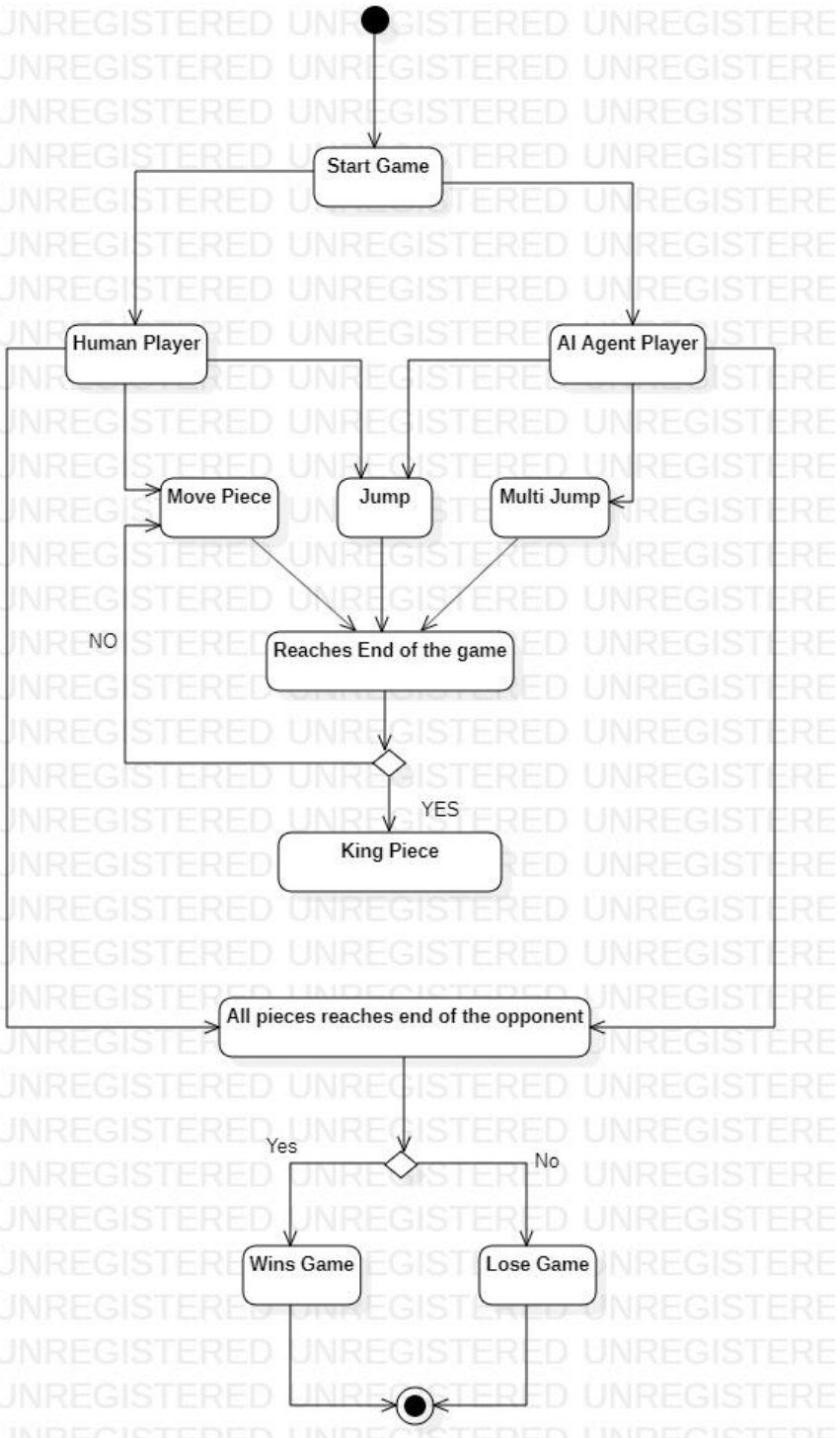



**Figure.7 ACTIVITY DIAGRAM**

## SEQUENCE DIAGRAM

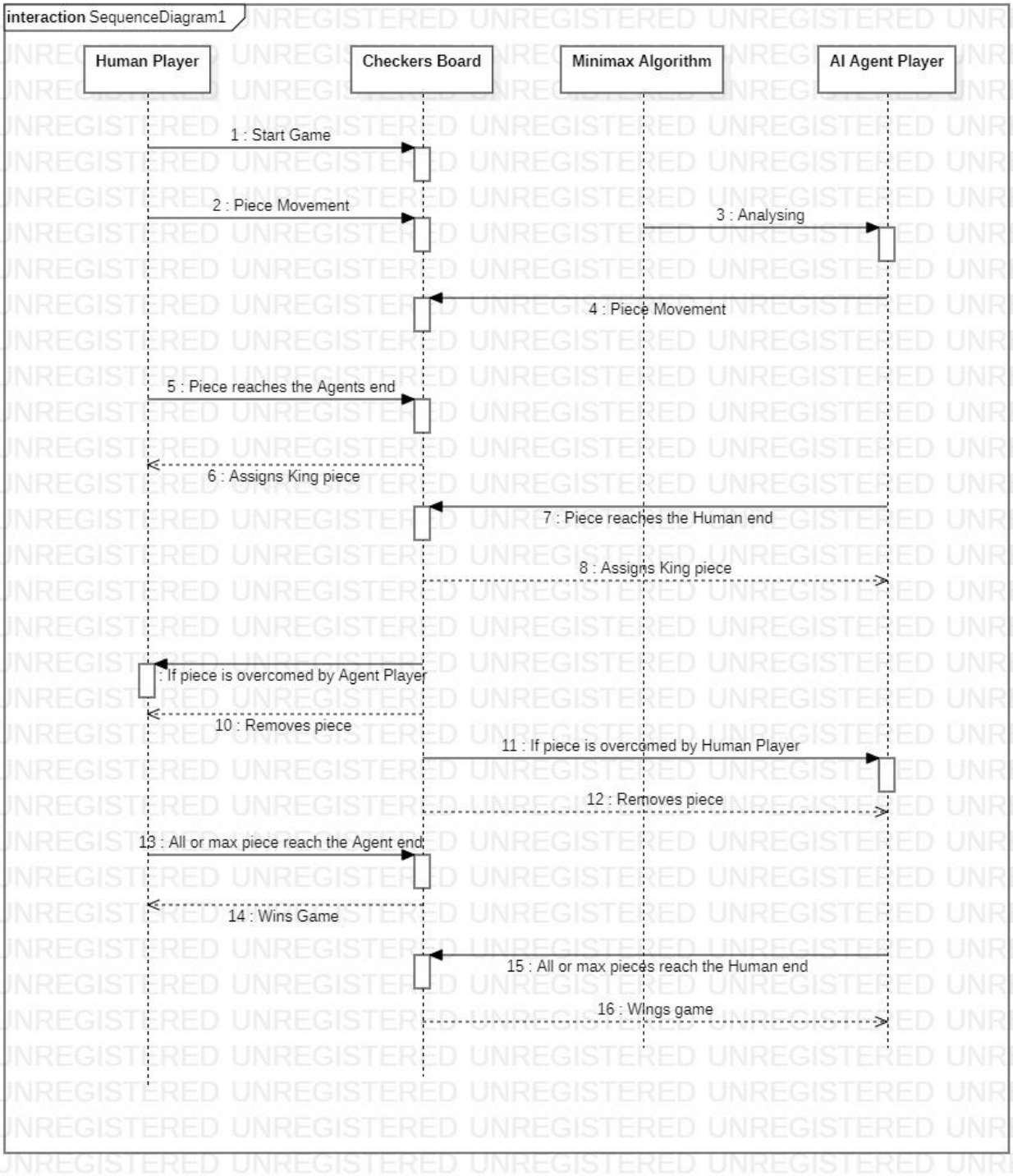



**Figure.8 SEQUENCE DIAGRAM**

## STATE DIAGRAM

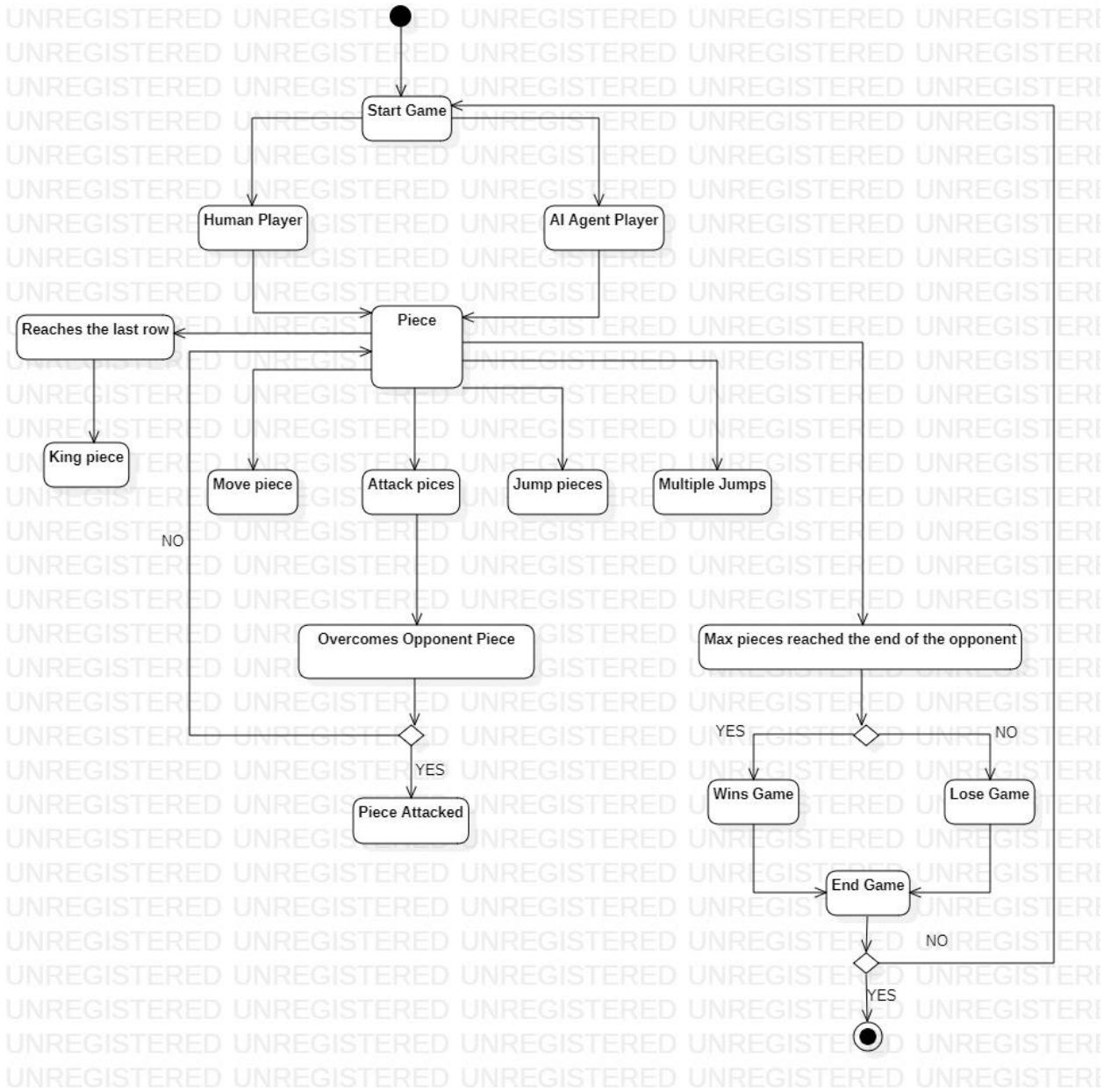



**Figure.9 STATE DIAGRAM**

## PACKAGE DIAGRAM


**Figure.10 PACKAGE DIAGRAM**

## DEPLOYMENT DIAGRAM

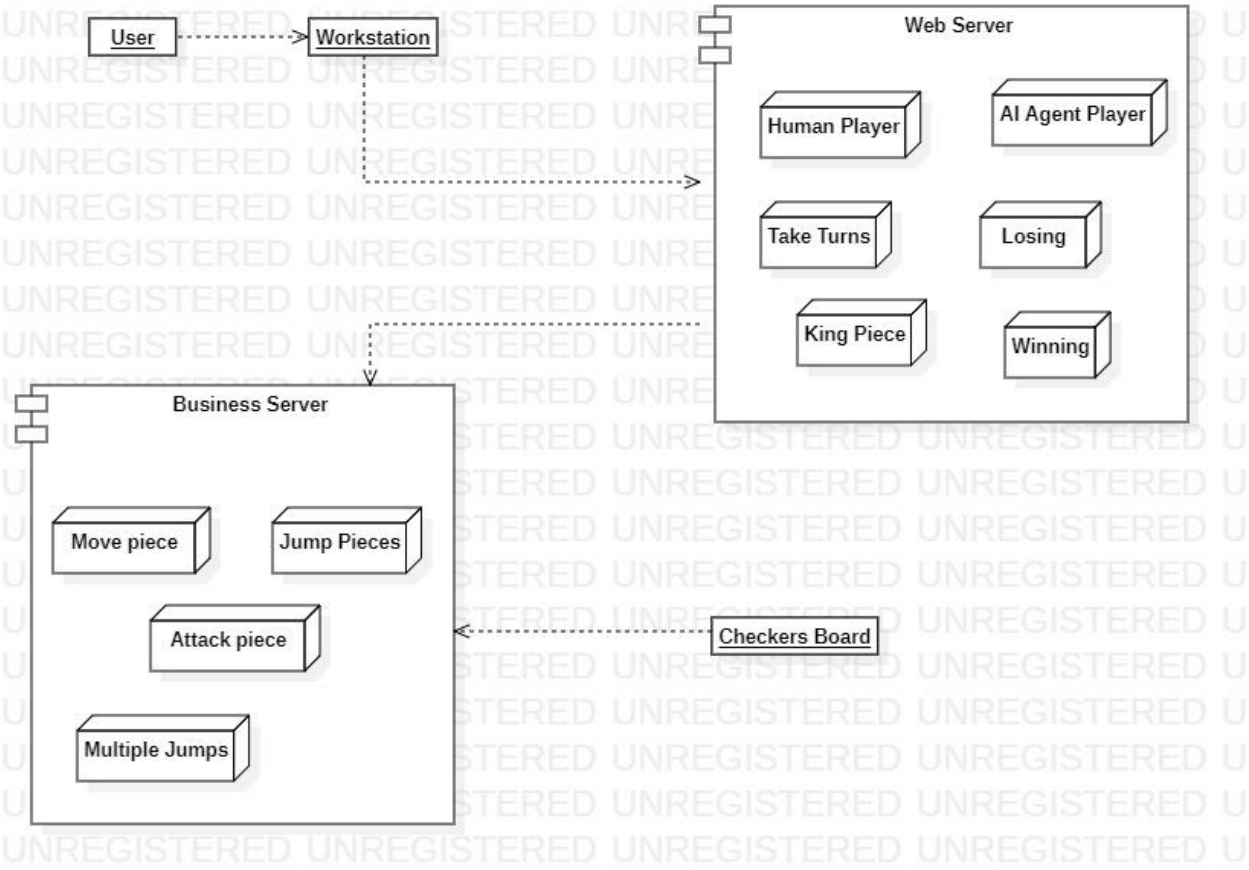



**Figure.11 DEPLOYMENT DIAGRAM**

## COMPONENT DIAGRAM

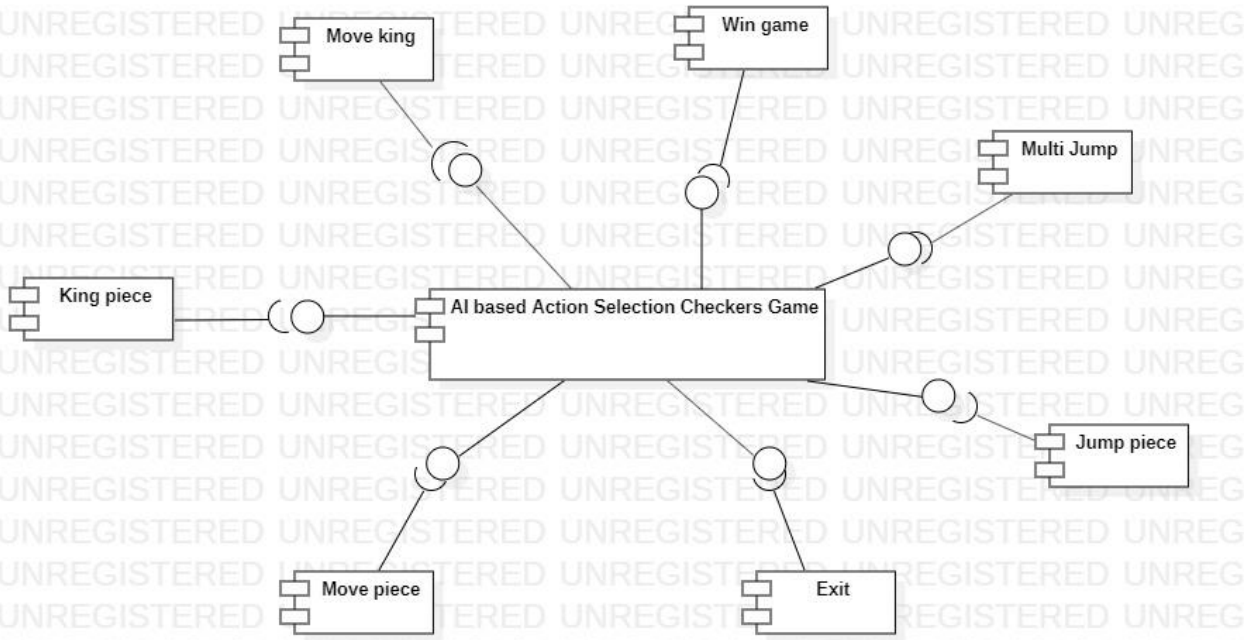



**Figure.12 COMPONENT DIAGRAM**

## DATAFLOW DIAGRAM LEVEL 0

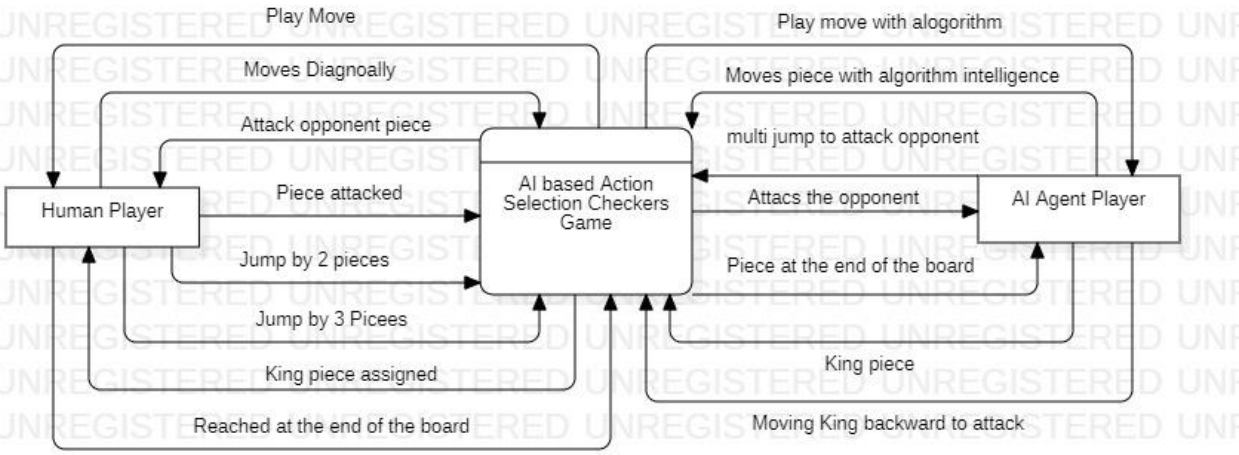



**Figure.13 DATA FLOW DIAGRAM LEVEL 0**

## DATA FLOW DIAGRAM LEVEL 1

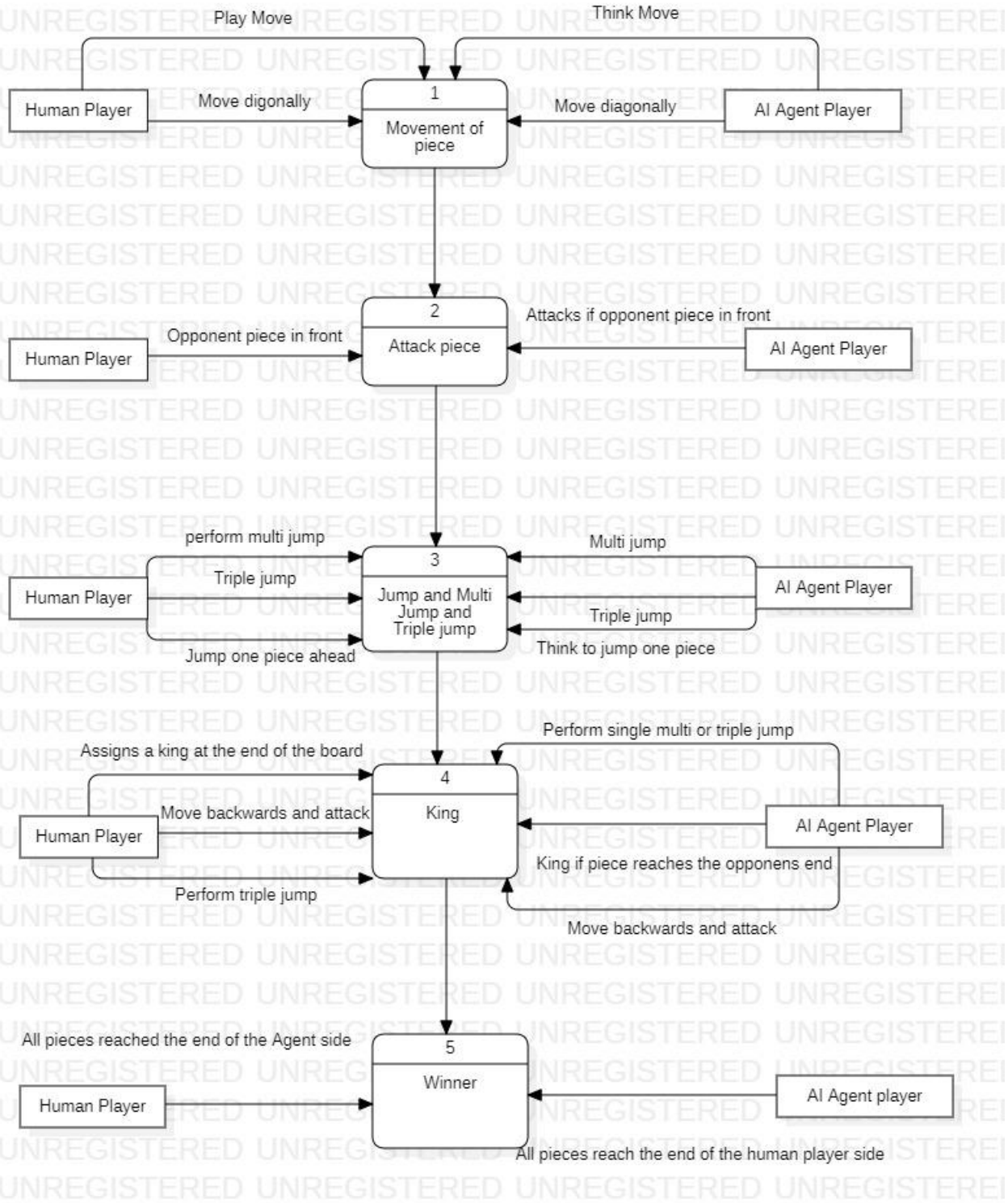



**Figure.14 DATA FLOW DIAGRAM LEVEL**

# 4 SYSTEM DESIGN

## 4.1 basic Module:

### Project structure

The project's structure is the first important step towards a successful project. Creating a structure that is easily extendable and self-explanatory could be difficult and tricky. Having said that, my project is not really big or that complicated to make it hard. I created two main 2 main classes. One contains all the functions related to the board designing which later extends other functionalities that are overridden on the board such as the piece designing, piece movement, king movement, jumps, piece removal, etc. and the main class which has an object of the classes used and runs the respective code in a systematic manner.

### Game control

All the game's controlling is done by the cursor and keyboard for saving and names. You can use mouse or any actuator that is able to perform the mouse tasks with cursor that is moving. The game itself is very easy to control.

### Game Board

The board is divided into 8 rows and 8 columns. The entire background was colored in black and later red color was placed at alternative positions and also looking at the even and odd placements. Moving on to the pieces, the pieces are designed giving the appropriate radius, padding, height, width, in which rows and columns it should be placed and lastly the rgb colors. One of the sides (Top) is the AI Agent Player and downwards is the Human Player. The King pieces will be identified when the opposite player team has reached at the end of the game, thus assigning a King to that piece.

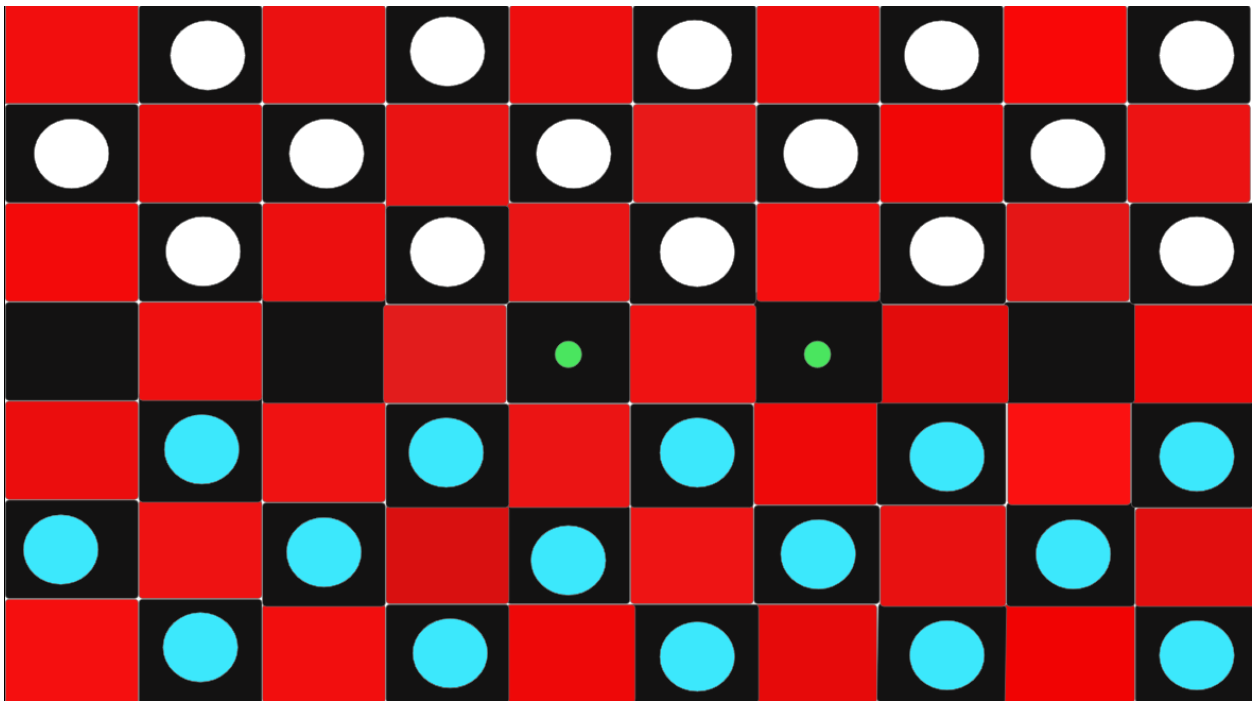
## Piece and King Movements

- Valid Moves

  When the Human Player clicks on the piece they want to move or play the turn with, the system gives 2-3 clues to where they can place the piece to. This functionality increases the interest of the user and also critical thinking in decision making is done.

- For jumps place the piece on its final position. Multiple jumps are performed sequentially by single jumps. If another jump is possible, then the cursor will not change back to normal and requires another positioning, until no other jumps are possible. It is not possible to place the piece back on its original position. Application does not allow plays that are against the rules.
- First click on the piece gives clues or options. Second click places the lifted piece on desired square.
- Whenever the Human Player has its turn it clicks on its piece, that piece will be disappeared from that position and when the Human Player clicks on the new position if that particular position is valid position then the piece will shown there as its valid movement otherwise the piece will come back to its original position or if the Human Player cannot make a decision on where to place the piece the system will give hints. These movements are discussed below:
  - None of the piece is selected. Now click on the piece shown by arrow.
  - When I will click on the piece that particular piece is removed from its original position.
  - Now the piece will be place at the position according to where the user wants.
  - When the piece has been moved at a new valid position so that the piece will be placed and shown at the new position.

### 4.3 User Interface Design.



## 4.4 Security Issues.

1. **Detect:**

   Although the system can't predict future attacks, it's very good at predicting the next move by an adversary once an attack is detected. That allows it to quickly close the door on an intrusion. If a resource opens up a connection to a known malicious IP address, for example, ML can recognize that and automatically shut it down before any data is exfiltrated.

2. **Protect:**

   In this game between adversary and defender, once an attacker makes a move, all the outcomes from that move can be determined through the system and can be flagged or blocked.

3. **Respond:**

   The system responds to any type of attack caused during the game or after or before the game has instantiates.

4. **Identify:**

   The system can identify the threats and intrusions.

5. **Recover**

   If the system has been attacked the system can easily recover the data.

## 1.5 Test Case Design.

| Test Condition | Input Selected | Expected Result | Actual Result |
|---|---|---|---|
| **Human Player's turn** | Piece Movement on a non-valid square | Piece cannot be placed there. | Piece cannot be placed there. |
| | Piece Movement on a valid square. | Piece moved to the new position. | Piece moved to the new position. |
| | Attack the AI piece on a non-valid square. | Piece cannot be placed there. | Piece cannot be placed there. |
| | Attack the AI piece on a valid square. | AI Agents piece is attacked and changed its color to Human player's piece. | AI Agents piece is attacked and changed its color to Human player's piece. |
| | Piece reached at the end of the opponents (AI) area of the board. | Piece is marked as a king. | Piece is marked as a king. |
| | King piece performs backward movement and attacking with | The AI pieces are attacked by king. | The AI pieces are attacked by king. |

| | | | |
|---|---|---|---|
| | single, multi or a triple jump. | | |
| | Pieces perform single, multi jump to attack the AI piece to a non valid square. | Piece cannot be moved, jumped to the desired attacked positions. | Piece cannot be moved, jumped to the desired attacked positions. |
| **AI Agent Player's turn** | Piece Movement on a non-valid square | Piece cannot be placed there. | Piece cannot be placed there. |
| | Piece Movement on a valid square. | Piece moved to the new position. | Piece moved to the new position. |
| | Attack the Human piece from a non-valid square. | Piece cannot be placed there. | Piece cannot be placed there. |
| | Attack the Human piece on a valid square. | Human Player piece is attacked and changed its color to AI player's piece. | Human Player piece is attacked and changed its color to AI player's piece. |

| | | | |
|---|---|---|---|
| | Piece reached at the end of the opponents (Human Player) area of the board. | Piece is marked as a king. | Piece is marked as a king. |
| | King piece performs backward movement and attacking with single, multi or a triple jump. | The Human pieces are attacked by king. | The Human pieces are attacked by king. |
| | Pieces perform single, multi jump to attack the Human piece to a non valid square. | Piece cannot be moved, jumped to the desired attacked positions. | Piece cannot be moved, jumped to the desired attacked positions. |
| **Winner** | All pieces of either player reaches the opponents side or there exists maximum number of 1 players pieces on the board. | Human Player or AI player won the game | Human Player or AI player won the game |

**Figure: 15 Test Case Designs**

# 5 IMPLEMENTATION AND TESTING

## 5.1 Implementation Approach

- Introduction

  The implementation of Checkers game starts with the UI designing of the board that contains a grid view with different colors in each grid to identify the attractiveness of the board game. Later the pieces were designed with an objective of playing or movement of the pieces, since there are 2 players, one is the Human Player and the other is the AI Agent Player there will be 2 parts of the pieces. On one side that is on the top side of the game there will be the AI Agent pieces followed by the Human Player pieces at the bottom of the board game. 2 different colors are then assigned to both the players, white is assigned to the AI player and blue color is assigned to the Human player. After the Board has been successfully designed the main and complex programming begins which is as follows: first the assignment of the king image on top of the piece is done followed by the movement of the pieces when clicked then 3 valid moves are assigned which will be shown with 2 blue dots and when the player selects any one of the blue dots it can only then be positioned to its new place or square. Later on if the player tries to move the piece to any other position other than the 2 valid positions give, they are forbidden to do that. Later, when any of the pieces reaches the end of the opponent's side will automatically be assigned king and then the king will be given to perform backward movements and attacks unlike the normal pieces which cannot move backward. A very fun part is of jumping the pieces, single, multi and triple jumps have been provided to the players but only if the valid blue dots shows it to them. The main and complex part of coding begins with the Minimax algorithm where I will be coding an agent to play against the Human player which will use the best attack methods and moves to play and have fun with the Human player. When any one of the players have maximum number of pieces on board and the other pieces have been attacked then that player is assigned as the winner of checkers game.

- Input and Output Design Implementation

  To start with my Project Implementation Plan, firstly, I had to decide which language I will be coding my project in, what dependencies are required, and what will be the frontend UI design. Once these Functionalities were decided I started with my Modules Implementation and I moved on to making the front-end design of my website application.

- Code Module

  This Activity shows that what Technologies I am using or my Front-End and Back-End. So, I am using Pythons pygame functions and methods for Front- End designing and Minimax algorithm for Back-End. At the User end a Web server will reside, the user will use a workstation and an internet connection to connect to my game or even install it then he/she can play the desktop game anywhere and anytime.

  My project consists of multiple modules. Firstly, the Human player will start the game and followed by the AI Player. Later on the 2 valid moves will provide a hint for the user to attack the opponent n a smart way as to win the game. The player can also do single, multi or triple jumps according to the valid positions given. Later when any of the pieces reach to the end of the opponent's side it will be assigned as a king and later the king can perform backward attacks and movements to win the game. The king can perform single, multi or triple jumps too.

- Project Summary

  When the user first launches the game, they are greeted by an appealing and user-friendly game board interface. Later, the pieces are built with the intention of playing or moving the pieces; there will be two parts of the pieces, one for the Human Player and the other for the AI Agent Player, so there are two players. The AI Agent pieces will be on one side of the board, at the top, and the Human Player pieces will be on the other side, at the bottom. Both players are then given two different colors: white for the AI player and blue for the Human player. Following the efficient design of the Board, the key and complex programming starts, which is as follows: The assignment of the king picture on top of the piece is completed first, followed by the movement of the pieces when clicked. Finally, three correct moves are assigned, which are marked by two blue dots, and the player can pick each of the blue dots only then will it be relocated to its new location or square. If the player attempts to move the piece to a position other than the two correct positions given later, they will be disqualified. When each of the pieces hits the end of the opponent's hand, the king is automatically assigned, and the king is granted the ability to move backwards and strike, unlike the regular pieces, which cannot move backwards. Jumping the pieces is a lot of fun, and the players have been given single, multi, and triple jumps, but only if the valid blue dots reveal it to them. The most complicated and time-consuming part of coding starts with the Minimax algorithm, in which I will create an agent to play against the Human player, using the best attack methods and moves to compete and have fun. When one of the players has the most pieces on the board and none of the other pieces have been hit, that player is declared the winner of the checkers game.

## 5.2 Coding Details and Code Efficiency

**Main.py:**

```
import pygame
from checkers.constants import WIDTH, HEIGHT, SQUARE_SIZE, RED, WHITE, BLUE
from checkers.board import Board
from checkers.game import Game
from minimax.algorithm import minimax
import pygame
from pygame.locals import *
import os

# Game Initialization
pygame.init()

# Center the Game Application
os.environ['SDL_VIDEO_CENTERED'] = '1'

# Game Resolution
screen_width=700
screen_height=700
screen=pygame.display.set_mode((screen_width, screen_height))

# Text Renderer
def text_format(message, textFont, textSize, textColor):
    newFont=pygame.font.Font(textFont, textSize)
    newText=newFont.render(message, 0, textColor)

    return newText
```

```
# Colors
white=(255, 255, 255)
black=(0, 0, 0)
gray=(50, 50, 50)
red=(255, 0, 0)
green=(85,107,47)
blue=(0,0,128)
yellow=(255, 255, 0)
pink =(255,0,127)
dblue= (0,255,255)

# Game Fonts
font = "freesansbold.ttf"



# Game Framerate
clock = pygame.time.Clock()
FPS=30

# Main Menu
def main_menu():

    menu=True
    selected="start"
    selected1="options"
    selected1="option"
    selected1="option1"
```

```
while menu:
    for event in pygame.event.get():
        if event.type==pygame.QUIT:
            pygame.quit()
            quit()
        if event.type==pygame.KEYDOWN:

            if event.key==pygame.K_UP:
                selected="start"

            elif event.key==pygame.K_DOWN:
                selected="quit"

            elif event.key==pygame.K_LEFT:
                selected1="options"
            elif event.key==pygame.K_LEFT:
                selected1="option"
            elif event.key==pygame.K_LEFT:
                selected1="option1"

            if event.key==pygame.K_RETURN:
                if selected=="start":
                    main()
                if selected=="quit":
                    pygame.quit()
                    quit()
                if selected1=="options":
                    pass
                if selected1=="option":
                    pass
                if selected1=="option1":
```

```
            pass

# Main Menu UI
screen.fill(blue)
title=text_format("AI Based Action Selection Checkers Game", font, 30, yellow)
if selected=="start":
    text_start=text_format("START", font, 30, red)
else:
    text_start = text_format("START", font, 30, white)

if selected=="quit":
    text_quit=text_format("QUIT", font, 30, red)
else:
    text_quit = text_format("QUIT", font, 30, white)

if selected1=="options":
    text_opt=text_format("::INSTRUCTIONS::", font, 40, pink)
else:
    text_opt = text_format("::INSTRUCTIONS::",font,40,pink)

if selected1=="option":
    text_opts=text_format("BLUE  IS THE HUMAN PLAYER", font, 20, dblue)
else:
    text_opts = text_format("BLUE PIECE IS THE HUMAN PLAYER",font,20,dblue)

if selected1=="option1":
    text_opt1=text_format("WHITE PIECE IS THE COMPUTER PLAYER", font, 20,dblue)
else:
    text_opt1 = text_format("WHITE IS THE COMPUTER PLAYER",font,20,dblue)

title_rect=title.get_rect()
```

```
        start_rect=text_start.get_rect()
        quit_rect=text_quit.get_rect()
        opt_rect=text_opt.get_rect()
        opts_rect=text_opts.get_rect()
        opt1_rect=text_opts.get_rect()

        # Main Menu Text
        screen.blit(title, (screen_width/2 - (title_rect[2]/2), 80))
        screen.blit(text_start, (screen_width/2 - (start_rect[2]/2), 300))
        screen.blit(text_quit, (screen_width/2 - (quit_rect[2]/2), 360))
        screen.blit(text_opt, (screen_width/2 - (opt_rect[2]/2), 430))
        screen.blit(text_opts, (screen_width/2 - (opts_rect[2]/2), 500))
        screen.blit(text_opt1, (screen_width/2 - (opt1_rect[2]/2), 550))
        pygame.display.update()
        clock.tick(FPS)
        pygame.display.set_caption("AI Based Action Selection Checkers Game")

        #Initialize the Game
#creating a module around the checkrs game and then add an API on top of the game
# so that we can use it with an AI later on

#1 setup a pygame display where we will be drawing everything onto.
#2 setup a basic event loop: this will check if we press the mouse, if we press a certain key.
MAIN LOOP
#3 setup basic drawings, draw the board for the chessboard, draw the pieces

FPS = 60     #G=Frame per second

WIN = pygame.display.set_mode((WIDTH, HEIGHT))
#WIN: a constant value. WIDTH,HEIGHT: 2 Variables we will define(in constants.py)
```

```
#set a caption
pygame.display.set_caption('Checkers Game using Minimax Agent')

#this will take the position of our mouse, and based on the position of our mouse what row and col we are in
def get_row_col_from_mouse(pos):
    x, y = pos   #this is gonna be a tuple that will have the x pos of our mouse and the y position of our mouse
    #and based on the square size we can calcu very easily which row and col we are in
    row = y // SQUARE_SIZE
    col = x // SQUARE_SIZE
    return row, col

#SEE MOUSE BUTTON DOWN
#when we press the mouse down,
# we will get what row and col were in, we will select that piece and move that piece wherever we want to move

#define a main funcion
def main():
    #create an event loop: this will run 'x' times per second which will check everything
    run = True

    #define a clock:define our game to run at a constant frame rate.
    #the clock will make sure that the main event loop wont run too fast or too slow.
    clock = pygame.time.Clock()

    #create a Baord obj
    #board = Board()
```

```
    #create a Game object
    game = Game(WIN)

    #for moving the pieces
    #piece = board.get_piece(0,1)
    #board.move_pieces(piece,4, 3)

    while run:
        clock.tick(FPS)

        #FOR AI MINIMAX
        if game.turn == WHITE:
            value, new_board = minimax(game.get_board(), 3, WHITE, game)
            game.ai_move(new_board)

        #for winner
        if game.winner() != None:
            print(game.winner())
            run = False

        #setup the basic event loop for pygame
        for event in pygame.event.get():
            if event.type == pygame.QUIT:    #this means we hit the red button on the top of the
screen
                run = False

            if event.type ==  pygame.MOUSEBUTTONDOWN:     #This means we pressed any
mouse on our mouse down
                pos = pygame.mouse.get_pos()
                row, col = get_row_col_from_mouse(pos)
```

```
            #call the select moethod from game.py
            #if game.turn == BLUE:
            game.select(row, col)

        game.update()

        #board.draw_squares(WIN)    #from board.py file, this function is called.
        #board.draw(WIN)            #for pieces, from board.py
        #pygame.display.update()

    pygame.quit()
main_menu()
main()
```

**Game.py**

```
#Make another file inside checkers, call this game.py: this file is responsible for actually
handling the game.
# Whose turn is it?, did we select the piece? Can the piece move here or there?
#Game class: this class will allow us to interface with the board, the pieces in which i used few
simple methods.
#this is not depending on anything

import pygame
from .constants import RED, WHITE, BLUE, GREEN, SQUARE_SIZE
from checkers.board import Board

class Game:
    def __init__(self, win):         #win:window we want to draw this game on
        self._init()
        self.win = win
```

```
    #update method to update the pygame display
    def update(self):
        self.board.draw(self.win)
        self.draw_valid_moves(self.valid_moves)
        pygame.display.update()

    #this is essentially initializing the game. This method is private but only sees the rest method.
    def _init(self):
        self.selected = None        #which piece is selected by someone whose playing the game
        self.board = Board()
        #this line is used because, from main() we will not make a board object but a Game
        #the Game will control the board for us, so rather having to get the piece or move the piece
all this will use the game.

        self.turn = BLUE
        #self.turn = RED
        self.valid_moves = {}     #this will tell what the current valid moves are for whatever player
is playing

    def winner(self):
        return self.board.winner()

    #reset method to reset the game
    def reset_game(self):
       self._init()

    #slect method to select the row and column.
    def select(self, row, col):   #this will tell the row and col we have selected
        #based on the selected row and column
        if self.selected:
```

```
            result = self._move(row, col)
            if not result:        #If selected an invalid move then this will call the main select method again to reste
                self.selected = None
                self.select(row, col)

        piece = self.board.get_piece(row, col)
        #If we're not selecting an emptypiece, we are actually selecting RED or WHITE
        if piece != 0 and piece.color == self.turn:
            self.selected = piece
            self.valid_moves = self.board.get_valid_moves(piece)
            return True      #If the selection is valid, then return True

        return False        #else if not valid move selected return False.

    #This is a private method, when the user selects the piece it calls the select piece.
    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:
            #piece = 0 that is an empty place, if its !=0 then there is a piece in that place.
            self.board.move_pieces(self.selected, row, col)
            skipped = self.valid_moves[(row, col)]
            if skipped:
                self.board.remove(skipped)
            self.change_turn()    #call the change_turn method

        else:
            return False
        return True
```

```
#if its RED it will go to WHITE or WHITE then itll go to RED
    def change_turn(self):
        self.valid_moves = {}
        if self.turn == BLUE:
            self.turn = WHITE
        else:
            self.turn = BLUE



    #for JUMP. draw the valid moves
    def draw_valid_moves(self, moves):       #all "moves" are dictionary
        for move in moves:                   #this loops through all the keys of the dictioanry
            row, col = move
            pygame.draw.circle(self.win, GREEN, (col * SQUARE_SIZE + SQUARE_SIZE//2, row
* SQUARE_SIZE + SQUARE_SIZE//2), 15)

                                #FOR AI MINIMAX

    def get_board(self):
        return self.board

    #when the AI makes a moves then this will return to us the new board after its move
    def ai_move(self, board):
        self.board = board
        self.change_turn()
```

**Piece.py**

```
import pygame
from .constants import RED, WHITE, SQUARE_SIZE, GREY, BLUE, GREEN, CROWN

#when we make a new piece we need to pass what row its in, what column its in, and what color it is
class Piece:

    PADDING = 15   #for radius FROM DRAW_PIECE()
    OUTLINE = 2   #for the outline  FROM DRAW_PIECE()

    def __init__(self, row, col, color):
        self.row = row
        self.col = col
        self.color = color
        self.king = False  #This will tell us, are we a King piece? that means we can jump backwards.
        self.x = 0
        self.y = 0
        self.calc_position()

        #DOWN = POSITIVE
        #UP = NEGATIVE
        #if self.color == BLUE:        #Direction will be negative, we are going up(down pieces)
            #self.direction = -1    #what way are we going, positive or negative.
        #else:
            #self.direction = 1
```

```
    #This will calculate the x and y positions base on the row and column we are in.
    #We need to know what our x and y positions will be according to the square size.
    def calc_position(self):
        self.x = SQUARE_SIZE * self.col + SQUARE_SIZE //2
        #col=dealing with the x which is the horizontal axis, and
        # squaresize//2: we want the piece to be in the middle of the squarex and y pos
        self.y = SQUARE_SIZE * self.row + SQUARE_SIZE //2

    #This function will change the King variable
    def King_piece(self):
        self.king = True    #we will make this piece a king

    #we will draw, we will draw the actual piece itself
    def draw_piece(self, win):

        radius = SQUARE_SIZE // 2 - self.PADDING

        #draw an outline (LARGER CIRCLE)
        pygame.draw.circle(win, GREY, (self.x, self.y), radius + self.OUTLINE)

        #start by drawing a circle and outline so that we can see it  (SMALLER CIRCLE)
        pygame.draw.circle(win, self.color, (self.x, self.y), radius)
        #x, y pos=center of the circle
        #pick a radius: based on padding bw the edge of the square and the circle. define class vari a
the top.

        #for king piece
        if self.king:
            #blit():put some image on to the screen, or put some surface on the screen.pygame
method
```

```
            win.blit(CROWN, (self.x - CROWN.get_width()//2, self.y - CROWN.get_height()//2))
            #take the x and y radius and the crown image height and wdth so that it fits in the middle


    #for moving the pieces, defined even in board.py
    def move_pieces(self, row, col):
        #when we move a piece, then it will have a new row, the piece will be updated
        self.row = row
        self.col = col
        self.calc_position()   #tells us the x and y position of our piece should be, so we have to recalculate that

    def __repr__(self):
        return str(self.color)
```

**Board.py**

```
#Put a class named "Board": This class will represent a checkers board.
#This class will handle all the different pieces moving, leading specific pieves, deleting specific pieces, rawing itself on the screen.

import pygame
from .constants import BLACK, ROWS, RED, SQUARE_SIZE, COLS, WHITE, BLUE, GREEN
from .piece import Piece

class Board:
    def __init__(self):
        #1. Internal representation of the board
        self.board = []   #creating a 2D list
        #Whose turn is it
        #self.selected_piece = None  #have we selected a piece yet, or not
```

```
        self.red_left = self.white_left = 12 #keeps track of how many red, how many white pieces
we have.
        self.red_kings = self.white_kings = 0
        self.create_board()

    #A surface/window to draw the red and black cubes on in a checkerboard pattern
    def draw_squares(self, win):
        win.fill(BLACK)
        for row in range(ROWS):
            for col in range(row % 2, COLS, 2):
                pygame.draw.rect(win, RED,(row*SQUARE_SIZE, col*SQUARE_SIZE,
SQUARE_SIZE, SQUARE_SIZE
                )) #Drawing the red rectangle starting from the top left

    #for moving the pieces
    #sel, piece=which piece we want to move it to, row, col=on which row and col we want to
move it to
    def move_pieces(self, piece, row, col):
        #move the piece within the list, also chenge the piece itself
        self.board[piece.row][piece.col], self.board[row][col] = self.board[row][col],
self.board[piece.row][piece.col]
        #swapping is happening by reversing

        piece.move_pieces(row, col)

        #Checking: if the piece hits the last row or first col, then making that piece as a king.
        if row == ROWS -1 or row == 0:
            piece.King_piece()   #King_piece(): from piece file, this will make the piece a king
```

```
        if piece.color == WHITE:
            self.white_kings += 1

        else:
            self.red_kings += 1

    #giving the baord object a row and col, and it wll give a piece back
    def get_piece(self, row, col):
        return self.board[row][col]

    #PIECES, creating the actual internal representation of the board and we will add a bunch of pieces to the list
    #A new class "piece.py"
    def create_board(self):
        for row in range(ROWS):
            self.board.append([])    #empty list: interior list for each row,
                                #a list that represents what each row is going to have inside of it
            for col in range(COLS):

                #if the current column tha we are one, if its divisble by if that equals to row+1, so we are on row 1
                if col % 2 == ((row + 1) % 2):   #part 2 3mins

                    #draw when we are in a certain row.
                    # if row < 3, 0 1 2 are the first 3 rows, we want to draw the white pieces in
                    if row < 3:
                        self.board[row].append(Piece(row, col, WHITE))

                    #if row>4, if irow is 5 6 7, then  we want to draw the red pieces
                    elif row > 4:
                        self.board[row].append(Piece(row, col, BLUE))
```

```
            #when no piece, add a zero
            else:
                # this will be a blank piece
                self.board[row].append(0)
        #when we dont add a piece, add a zero
        else:
            self.board[row].append(0)

#This will draw the board, this will draw all of the pieces and the squares.
def draw(self, win):
    self.draw_squares(win)   #we will draw the sqaures on the window

    #Loop throuhout the pieces and draw those
    for row in range(ROWS):
        for col in range(COLS):

            #loop through the board
            piece = self.board[row][col]
            if piece != 0:     #if piece is 0, we will not draw anything
                piece.draw_piece(win)    # draw the oiece on the window


def remove(self, pieces):
    for piece in pieces:
        self.board[piece.row][piece.col] = 0
        if piece != 0:
            if piece.color == BLUE:
                self.red_left -=1
            else:
                self.white_left -= 1
```

```
    def winner(self):      #This will return the color if they won
        if self.red_left <= 0:
            return WHITE
        elif self.white_left <= 0:
            return BLUE

        return None

    def get_valid_moves(self, piece):
        moves = {}      #store the move as the key, so what place we could potentially move to as a
row

        #these are the diagonals (left or right)
        left = piece.col - 1   #we are moving left one
        right = piece.col + 1
        row = piece.row

        #checking whether we can move up or we can move down based on the color and based on
if the piece is a KING
        if piece.color == BLUE or piece.king:
            moves.update(self._traverse_left(row -1, max(row-3, -1), -1, piece.color, left))
            moves.update(self._traverse_right(row -1, max(row-3, -1), -1, piece.color, right))

        if piece.color == WHITE or piece.king:
            moves.update(self._traverse_left(row +1, min(row+3, ROWS), 1, piece.color, left))
            moves.update(self._traverse_right(row +1, min(row+3, ROWS), 1, piece.color, right))

        return moves
```

```
    #This function will look at the left diagonals for us.
    def _traverse_left(self, start, stop, step, color, left, skipped=[]):
        moves = {}
        last = []
        for r in range(start, stop, step):
            if left < 0:
                break   #when we are at the edge of the board on the left side then no further cols we have


            current = self.board[r][left]
            if current == 0:
                if skipped and not last:
                    break
                elif skipped:
                    moves[(r, left)] = last + skipped

                else:
                    moves[(r, left)] = last

                if last:
                    if step == -1:
                        row = max(r-3, 0)
                    else:
                        row = min(r+3, ROWS)

                    moves.update(self._traverse_left(r+step, row, step, color, left-1, skipped=last))
                    moves.update(self._traverse_right(r+step, row, step, color, left+1, skipped=last))
                break

            elif current.color == color:
                break
```

```
            else:
                last = [current]

            left -= 1

        return moves

    #WHY USE THESE PARAMETERS:
    #start, stop, step is going to be for the for loops we're going to put inside of here
    #step: very imp, becuase it tells us, we go up? or down? when im traversing thro the rows for the diagonals
    #skipped: this will tell us, have we skipped any pieces yet? if yes, we can only move to squares when we skip another piece

    #This method will move to the right
    #right: where are we starting in terms of col, when were traversing to the left
    def _traverse_right(self, start, stop, step, color, right, skipped=[]):
        moves = {}
        last = []
        for r in range(start, stop, step):
            if right >= COLS:
                break   #when we are at the edge of the board on the left side then no further cols we have

            current = self.board[r][right]
            if current == 0:
                if skipped and not last:
                    break
                elif skipped:
                    moves[(r, right)] = last + skipped
```

```
            else:
                moves[(r, right)] = last

            if last:
                if step == -1:
                    row = max(r-3, 0)
                else:
                    row = min(r+3, ROWS)

                moves.update(self._traverse_left(r+step, row, step, color, right-1, skipped=last))
                moves.update(self._traverse_right(r+step, row, step, color, right+1, skipped=last))
            break

        elif current.color == color:
            break
        else:
            last = [current]

        right += 1

    return moves

                            #FOR AI MINIMAX
  #The evaluate(): this method will tell us given the state of this board what is its score?
  def evaluate(self):   #This will take into account hwo many kings we have? and how many pieces we have?
    return self.white_left - self.red_left + (self.white_kings * 0.5 - self.red_kings * 0.5)
    #Thsi will give us another value.

  #This method will return to us all the pieces of a certain color. so we can use the board
  def get_all_pieces(self, color):
```

```
    pieces = []
    for row in self.board:
        for piece in row:
            if piece != 0 and piece.color == color:
                pieces.append(piece)
    return pieces
```

**Constants.py**

```
#Put all the constant values inside of this file
import pygame

#DEFINE THE HEIGHT AND WIDTH
WIDTH, HEIGHT = 700, 700   #800 pixels

#DEFINE ROWS AND COLS IN CHECKERS BOARD
ROWS, COLS = 8, 8
#HOW BIG IS ONE SQAURE OF THE CHECKER BOARD
SQUARE_SIZE = WIDTH//COLS

#DEFINE VARIABLE FOR COLOURS
#RGB COLOR
RED = (255, 0, 0)
WHITE = (255, 255, 255)
BLACK = (0, 0, 0)
BLUE = (0,255,255)
GREY = (128,128,128)
GREEN = (0, 153, 0)

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

**Algorithm.py**

```
from copy import deepcopy
import pygame

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



#position:- This stands for the current posisiton that we are in.
   #depth:- Tells us how how far or extendeting the tree.
      #This will be a recursive call. we will decrement this
   #max_player: A boolean value, which tells are we minimizing the value or min the value.
   #game:- The actual game object which we will get from main.py file and pass that to the alg

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, BLUE, 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_pieces(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
```

### 5.2.1 Code Efficiency

Pygame has many libraries and modules which provides a user friendly and we can design a very attractive UI design. By implementing a Minimax function, the AI agent will perform and act optimistically with the best possible move.

## 5.3 Testing Approach

I have tested my project with Unit Testing, Integration Testing, System Testing, Black Box Testing, White Box Testing, Regression Testing, etc. Regression Testing means when we do some changes after testing and then test those changes only which are made after testing is done that's called as Regression Testing. I have tested my each module separate that comes under Unit Testing and I have also Integrated my one module with another then test it that comes under integration Testing.

### 1.1.1 Unit Testing

Unit Testing is done by testing each module as a separate function. As an example the board has multiple modules to it. So each module had to be tested e.g. Testing if the king placement works, if valid moves works, is doing an attack working and when doing jumps, multi jumps or single jumps are working.

### 1.1.2 Integration Testing

Integration Testing means testing more than one Module at a time. Taking the same example, the king placement module had to be checked if it worked together e.g. if there are 2 kings and both are now moving backwards to attack on a particular piece, reflected changes will be made to the board game.

### 1.1.3 System Testing

System testing is conducted on a complete integrated system to evaluate the system's compliance with its specified requirements. System testing takes, as its input, all of the integrated components that have passed integration testing. The application has successfully passed the system testing as per the objectives defined and has been tested on multiple devices covering different browser platforms.

## 5.4 Modification and Improvements

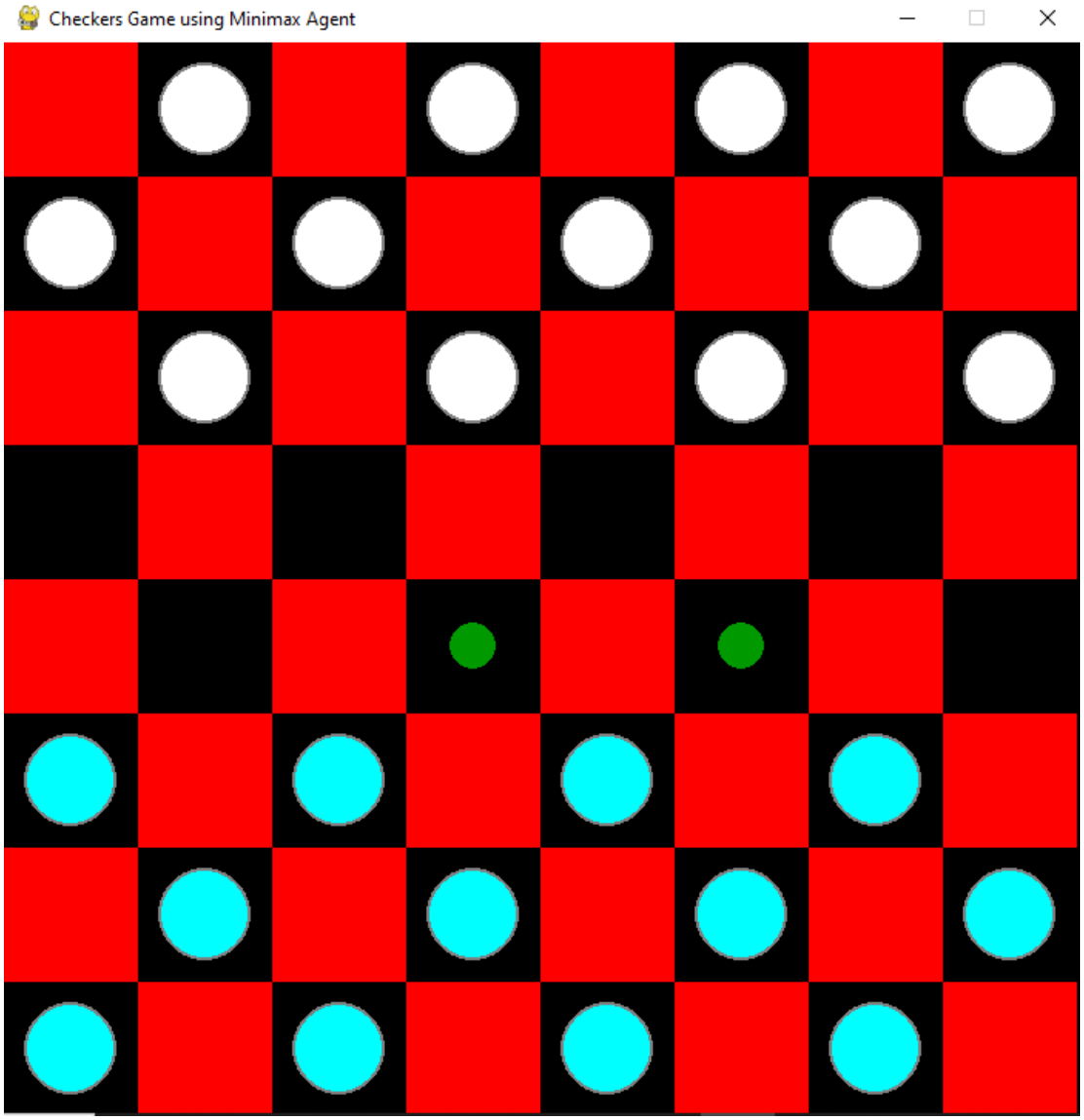
After testing and handling all errors, exceptions and bugs, errors were solved by troubleshooting, exceptions were handled by try catch block at the correct places avoiding redundancy of code from index files. Bugs can never be removed completely but most of them are solved and remaining is under development.

# 6 RESULTS AND DISCUSSION

## 6.1 Test Reports

1. Input: - Piece Movement on a non-valid square

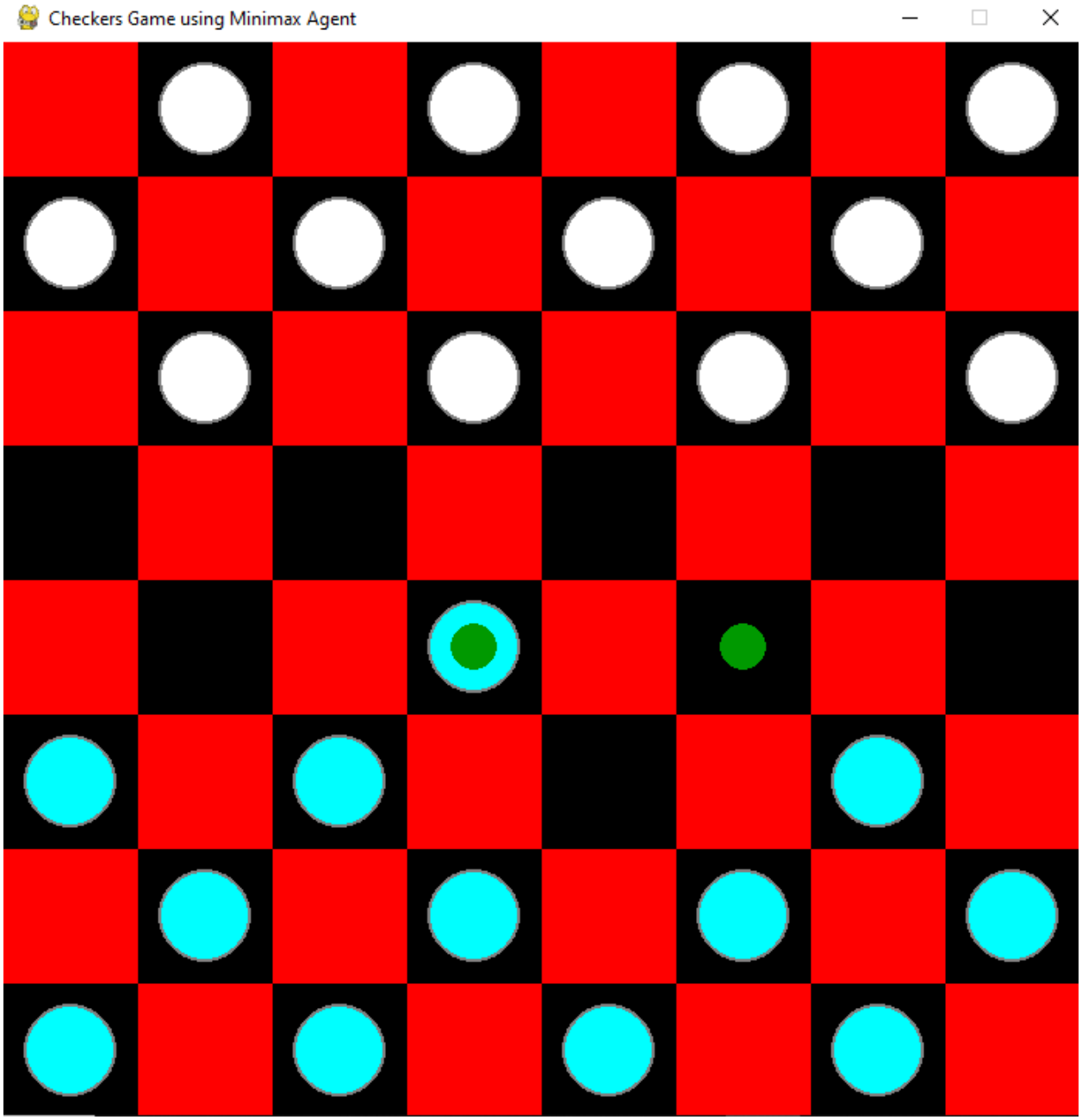
   Actual/Expected result: - Piece cannot be placed there.




The user cannot move the pieces to the invalid positions. It has to follow the green valid points for piece movements.

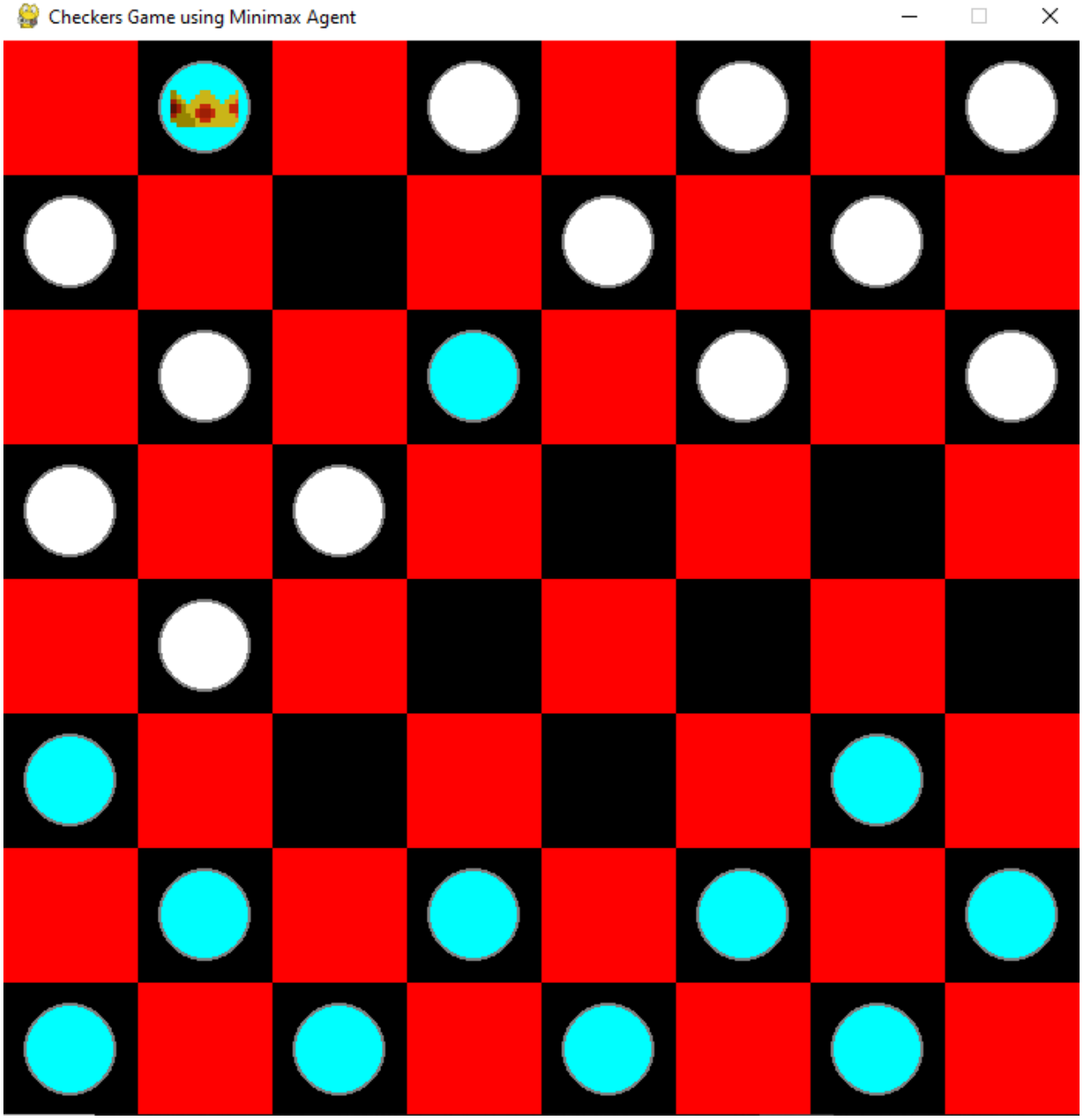
2. Input :- Piece movement to a valid square

   Actual/Expected result: Piece moves to the valid square.



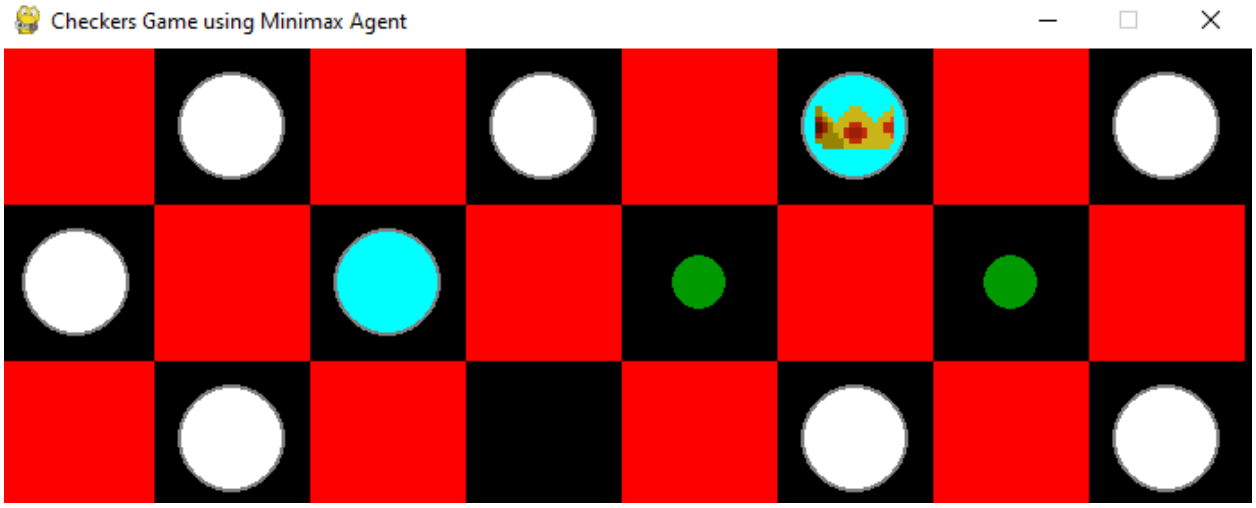

When the user clicks on the green valid position, then only the system allows the piece to change its position and get a new one.

3. Input: - Piece reached at the end of the opponent's area in the board.
   Actual/Expected result: - Piece is marked as a king.



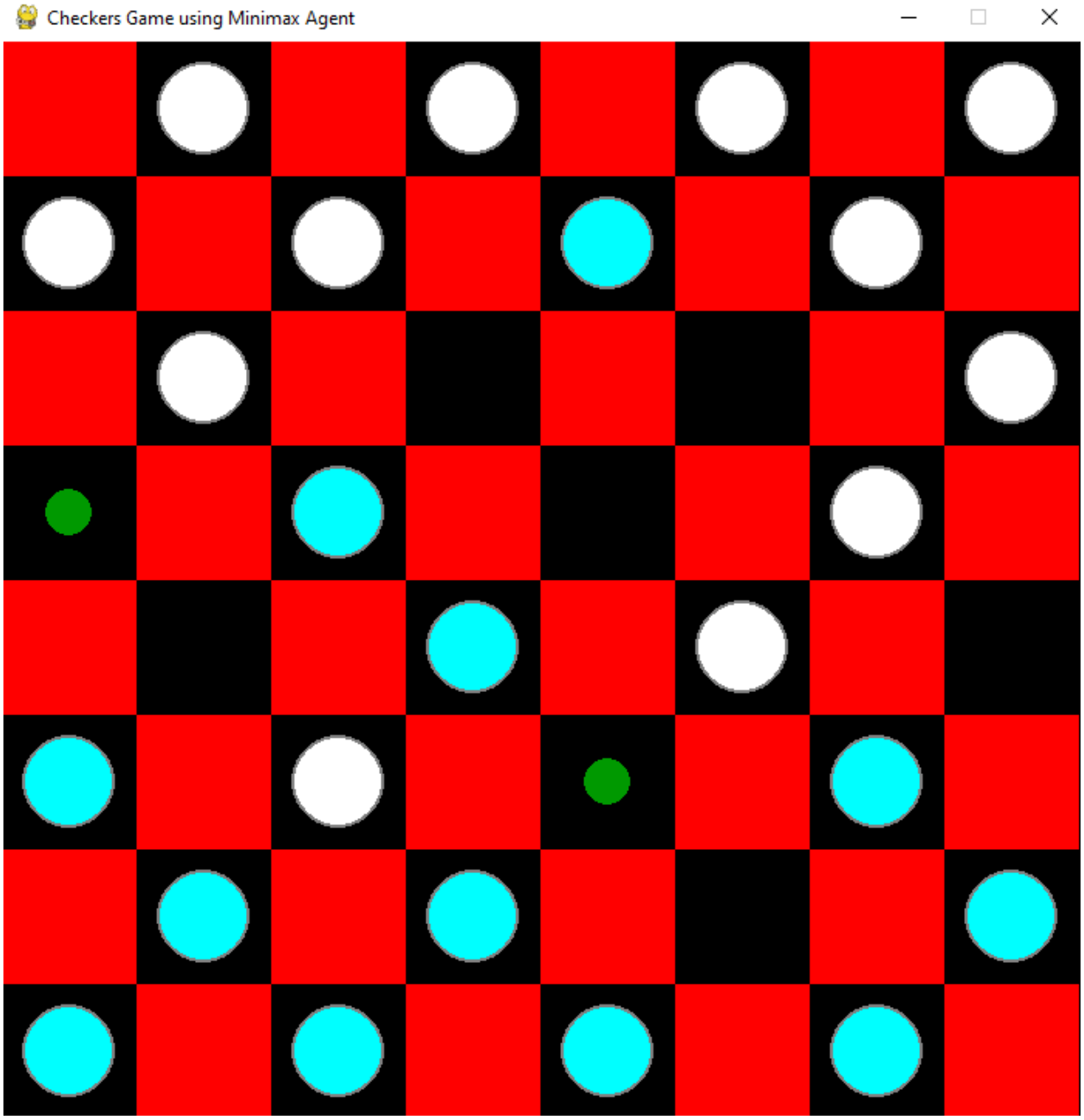

When the blue piece reaches at the end of the opponent's side of the board, it gets marked as the King piece by placing a crown on top of it.

4. Input: - King piece performs backward movement and attacking with single, multi or a triple jump.
   Actual/Expected result: - The AI pieces are attacked by king.



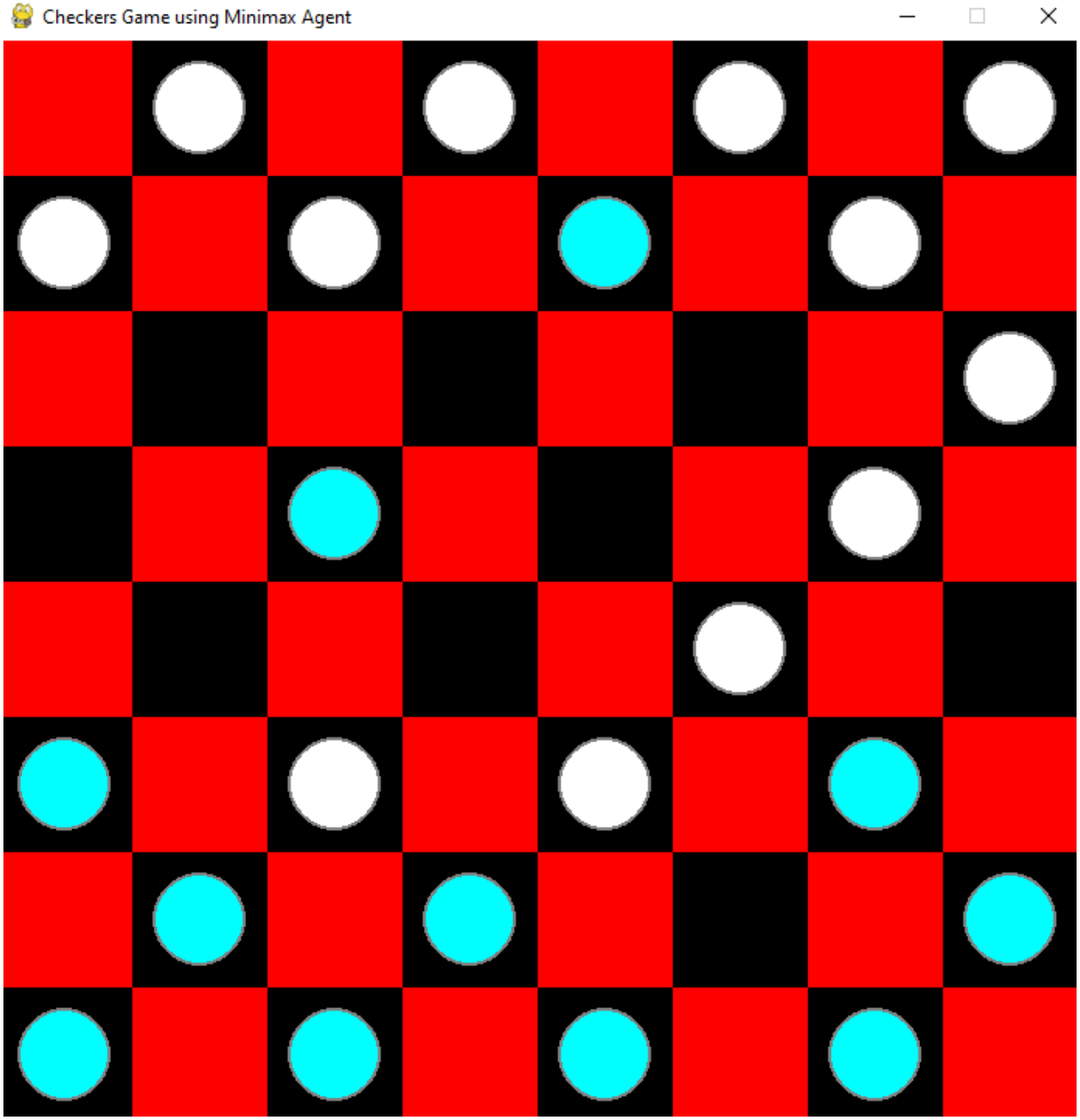

The king can perform backward attacks and movements.

5. Input: - Pieces perform single, multi jump to attack the AI piece to a **non valid square**. Actual/Expected result: - Piece cannot be moved, jumped to the desired attacked positions.




When the user performs single, multi or triple jump on a non valid square, it cannot do so.

6. Input: - Pieces perform single, multi jump to attack the AI piece to a **valid square**.
   Actual/Expected result: - Piece moved, jumped to the desired attacked positions.




When the user performs single, multi or triple jump on a valid square and attack the opponent it can do so.

## 6.2 User Documentation

When the user installs or plays this game, he/she are provided with the main screen of the board game and starting with their turn first(Human Player), then when the user plays his/her move the AI Agent (opponent) plays the next optimal move. The Human player can attack the AI player by only performing the valid positions. The Human player can perform single, multiple, or triple jumps too in order to attack the opponent or jump their pieces to reach the end of the other side (opponents) of the board. When the piece reaches at the end of the users or opponents side of the game the pieces are assigned as king pieces and they are identified by a crown. When the king piece performs backward movements or attacks it can and thus it can also perform single, multi or triple jumps in order to attack the opponent. When one of the players pieces are maximum on the board then they are declared as the winner.

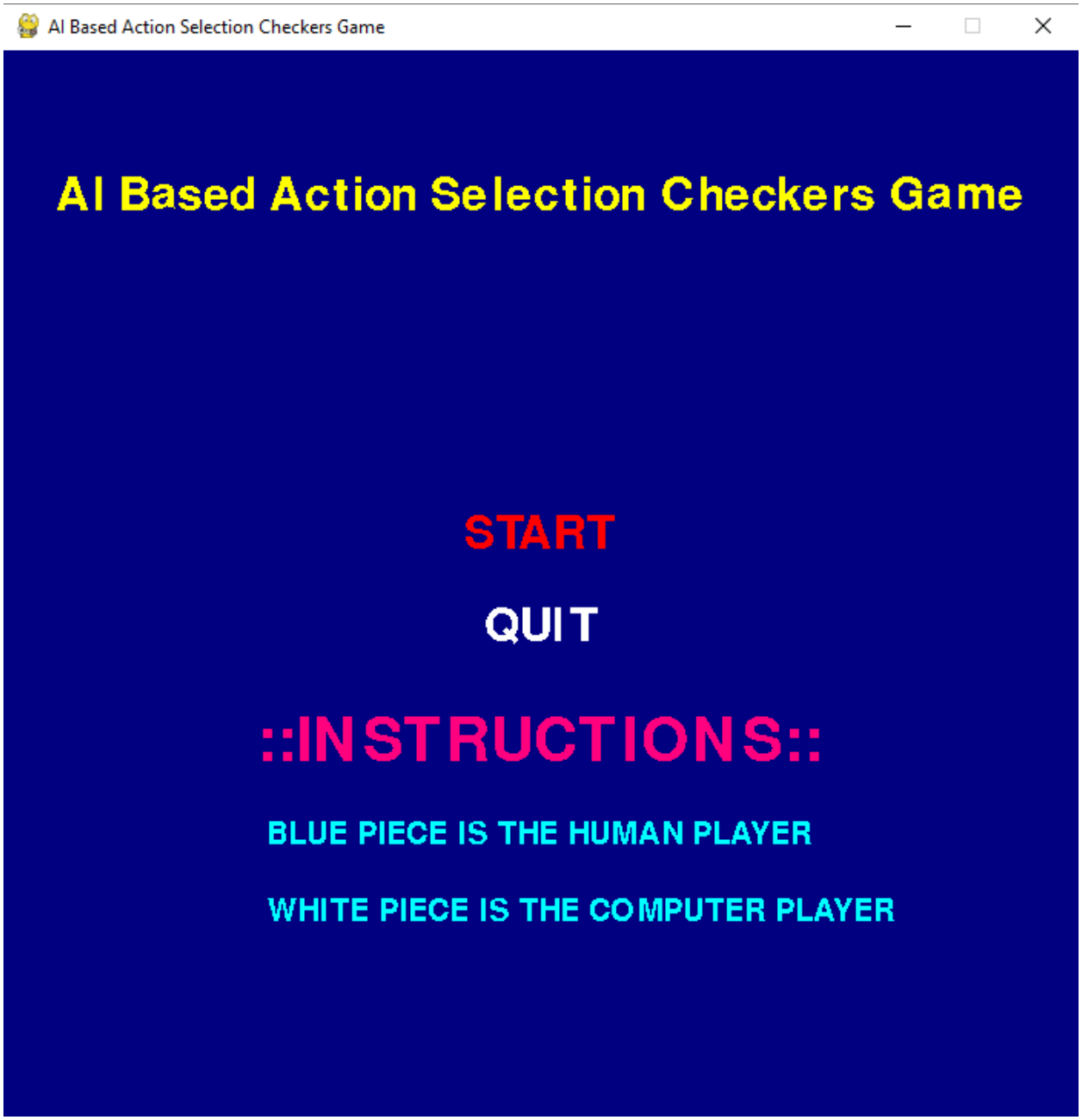



**Figure: 16 Main Menu screen**

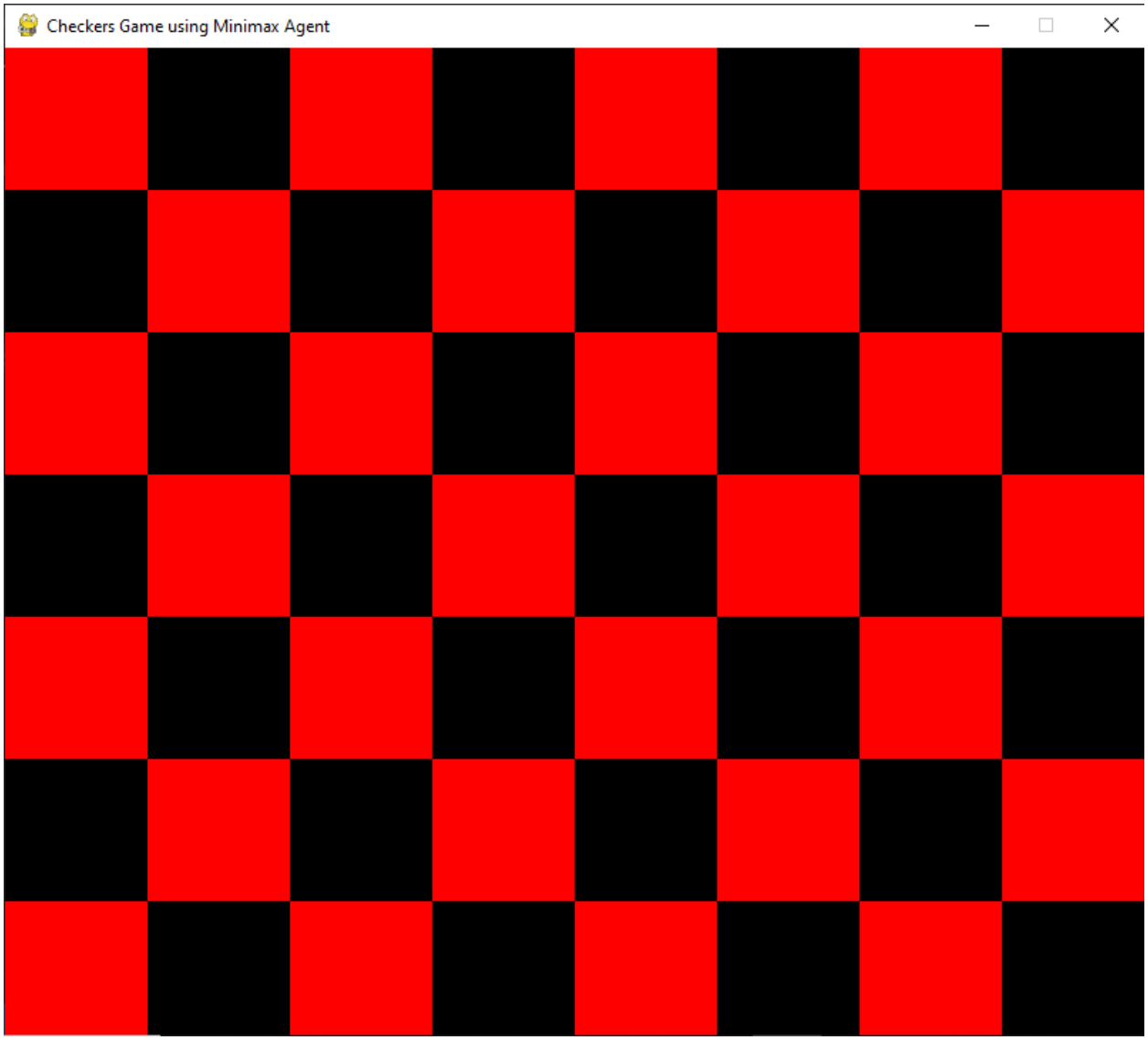



**Figure: 17 Board without pieces screen**

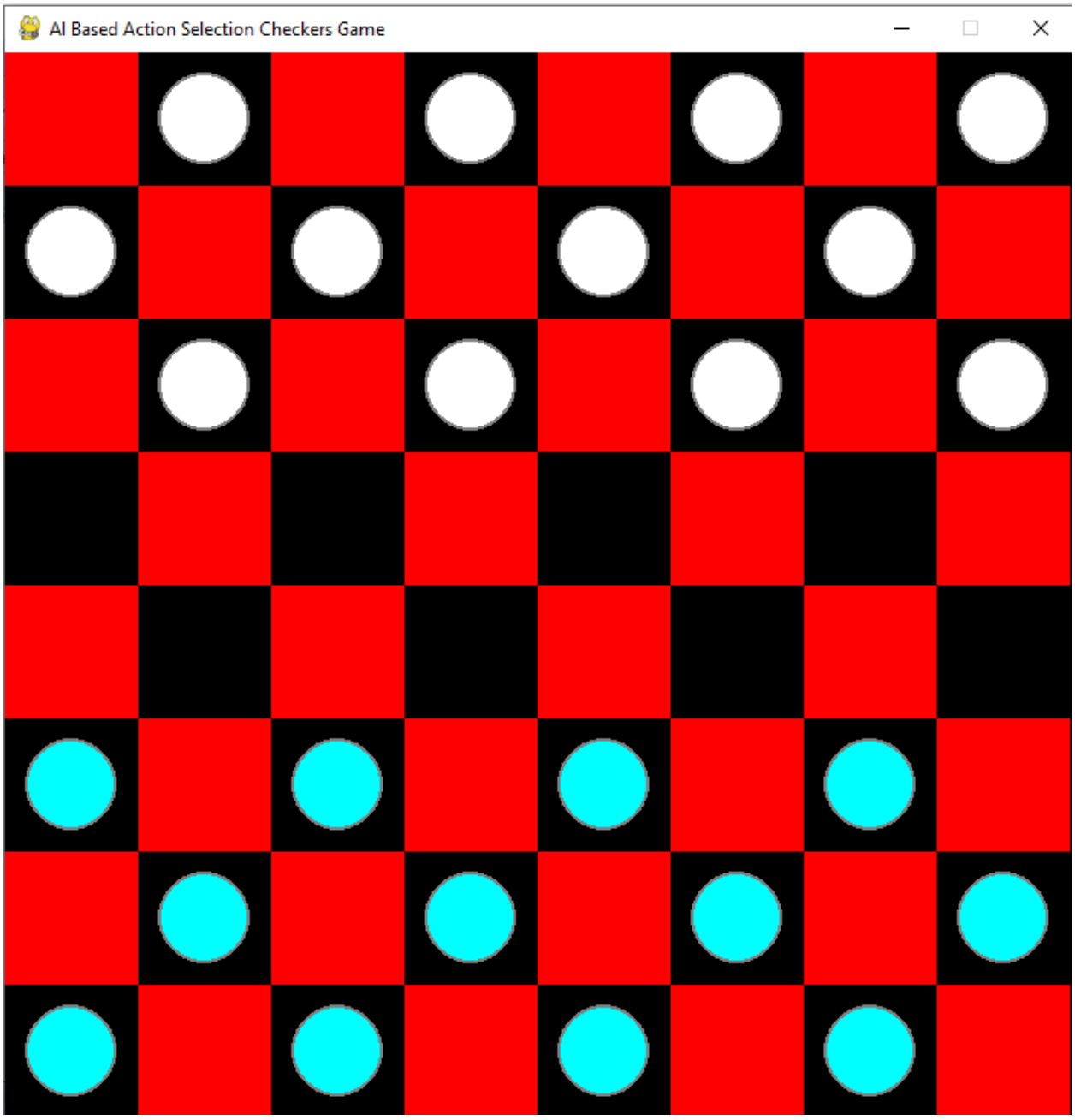



**Figure: 18 Board with pieces**

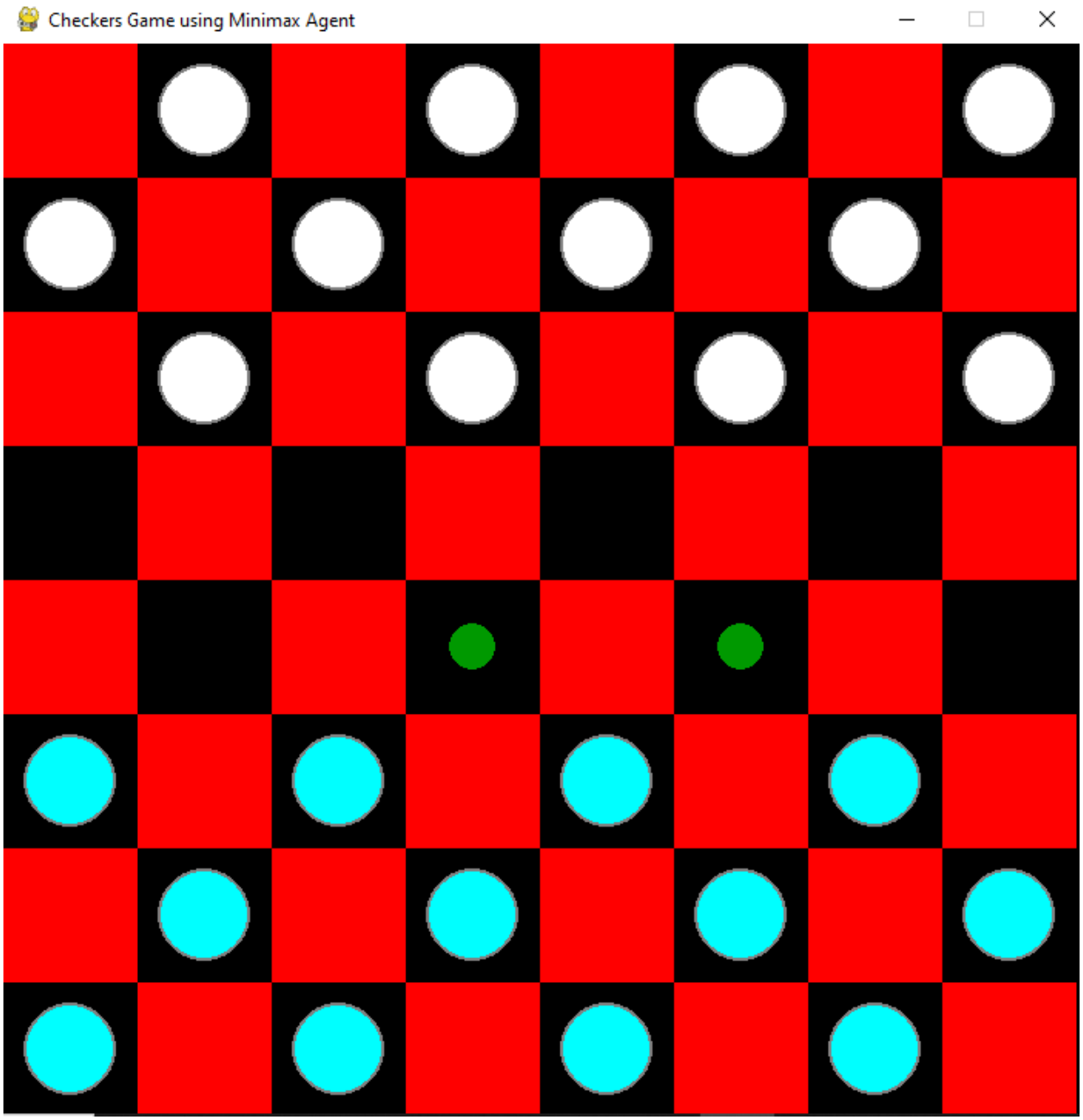



**Figure: 19 Human piece Movement**

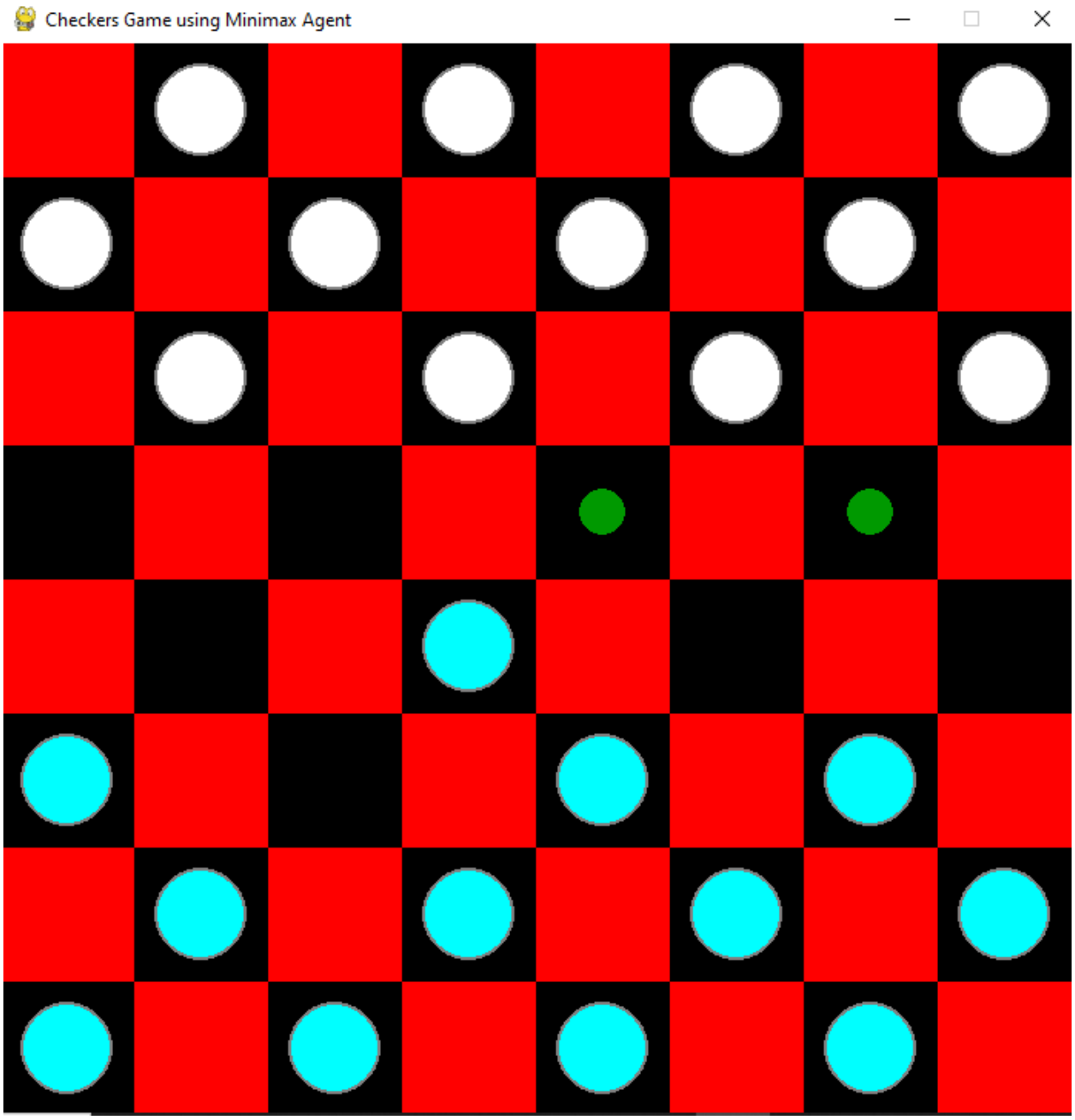



**Figure: 20 Human piece Movement**

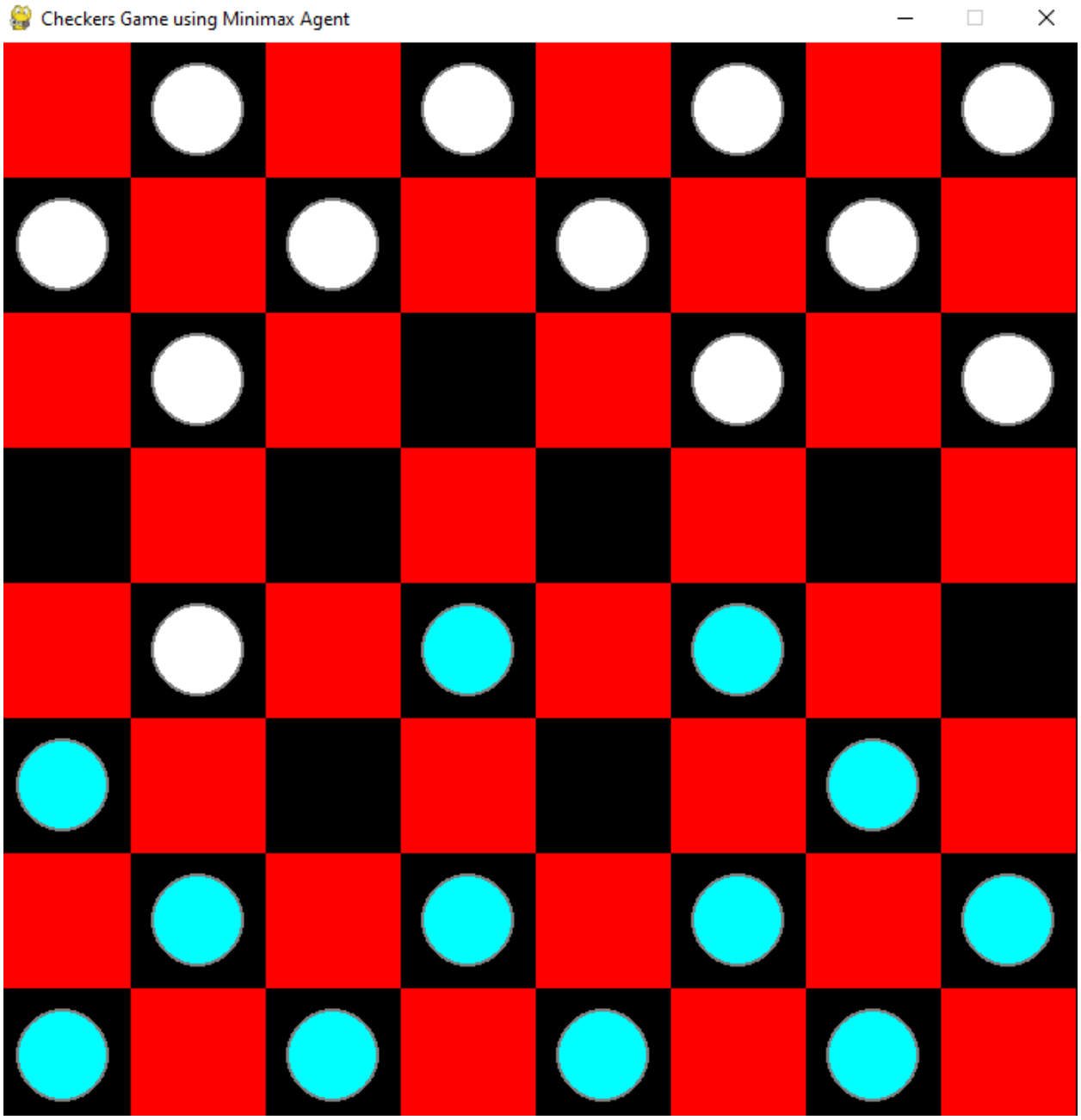



**Figure: 21 AI Agent piece movement**

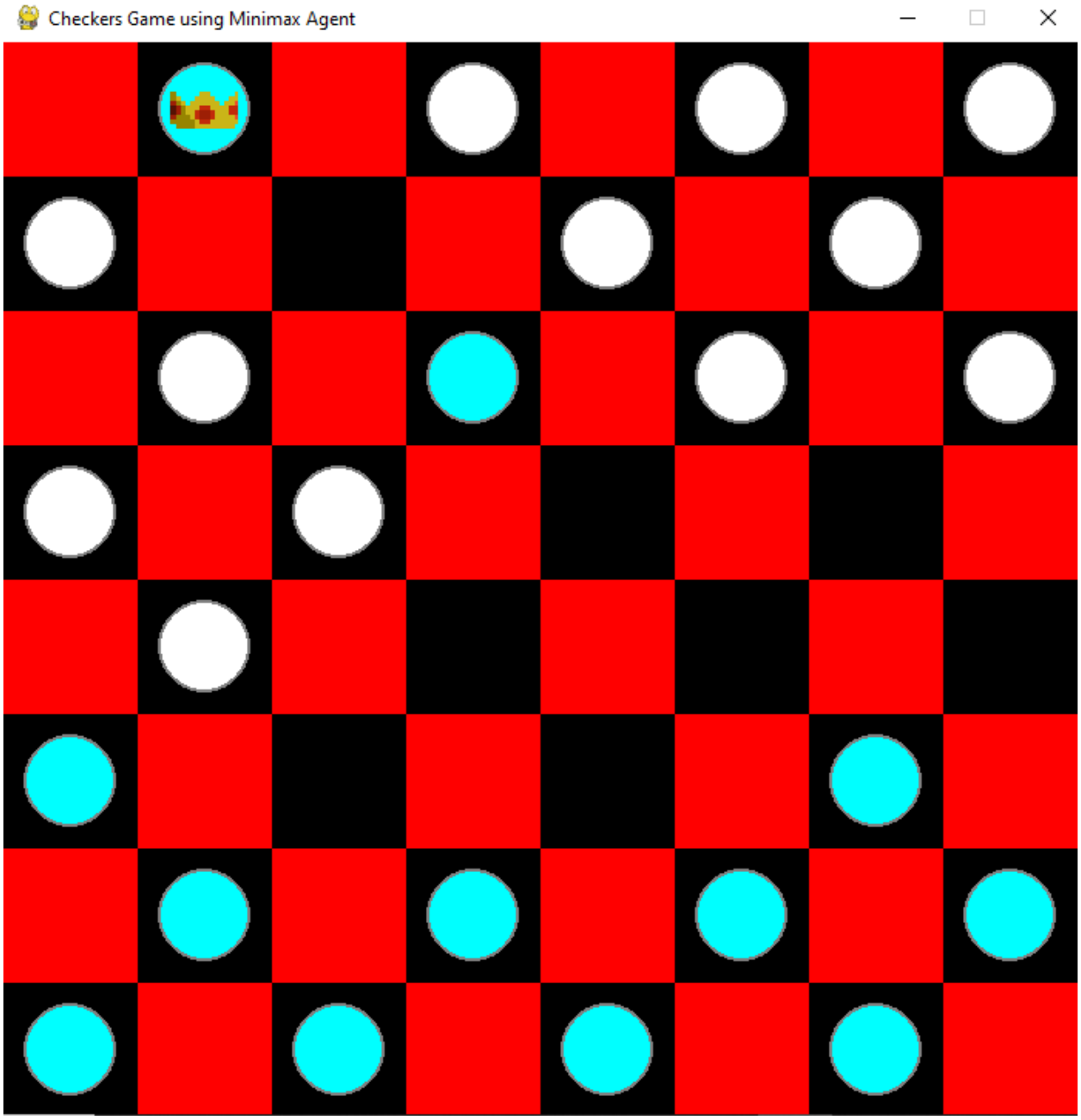



**Figure: 22 King piece assigned**

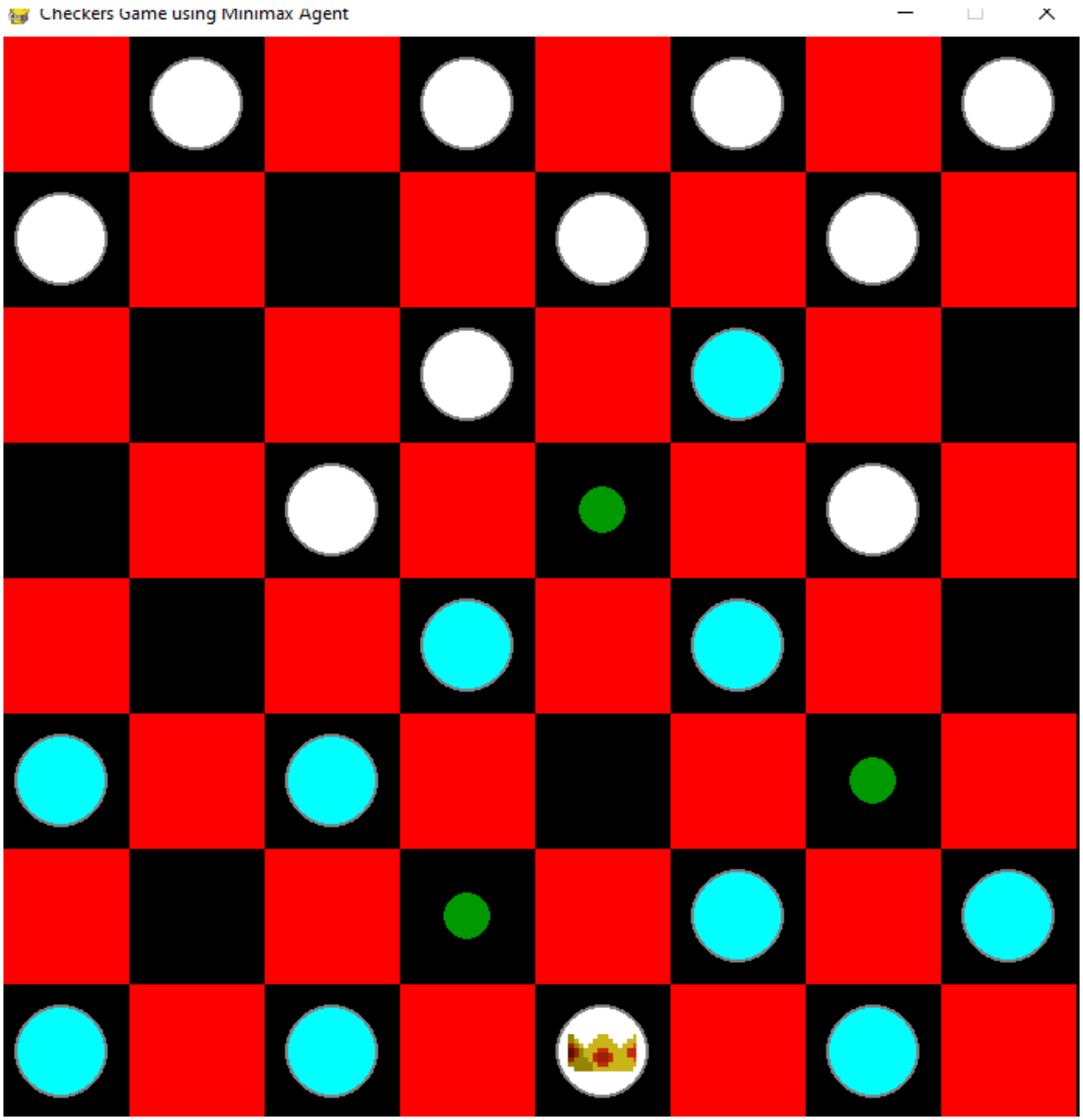



**Figure: 23 King piece attack with triple jump**

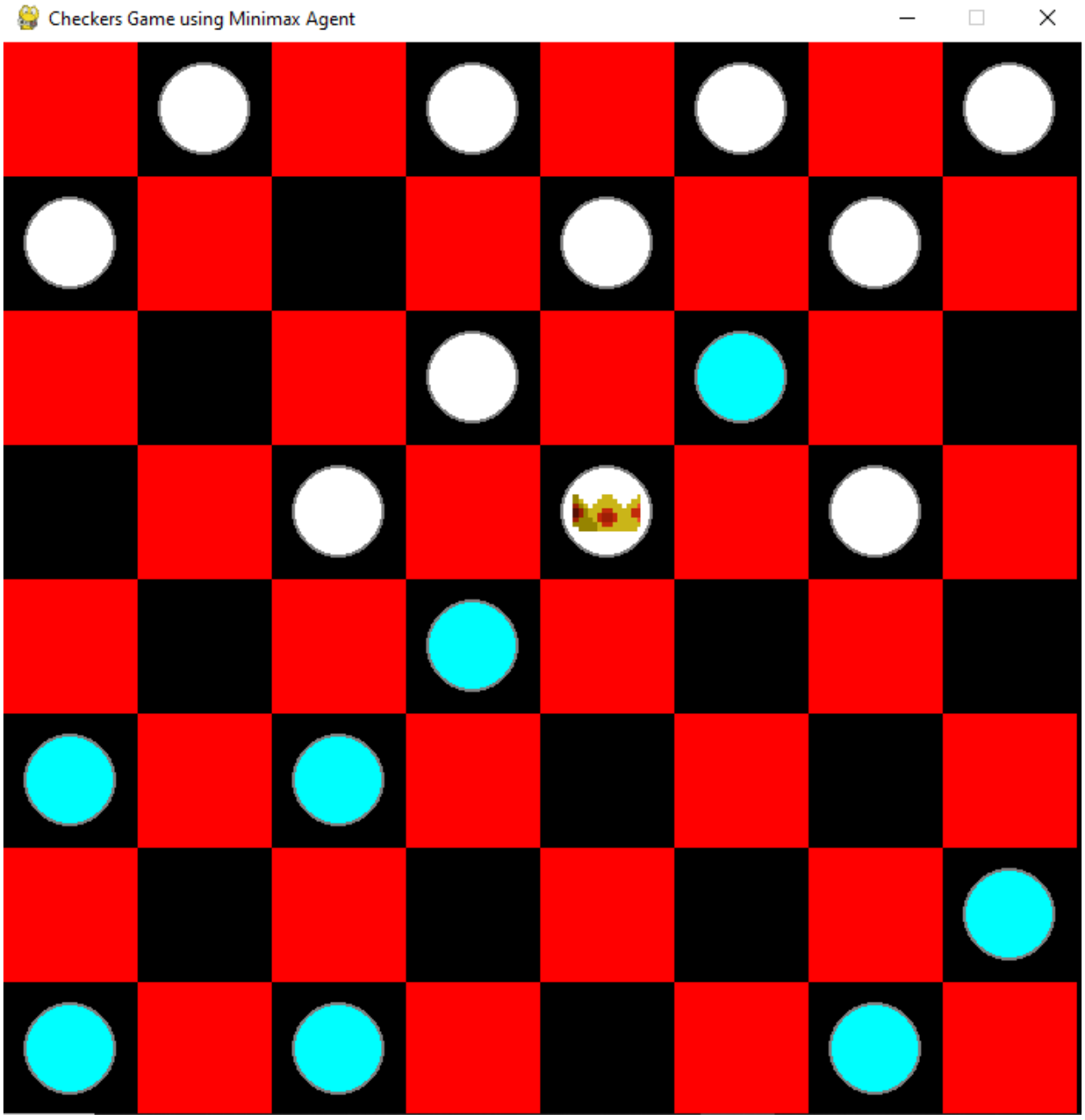



**Figure: 24 King piece attacked**

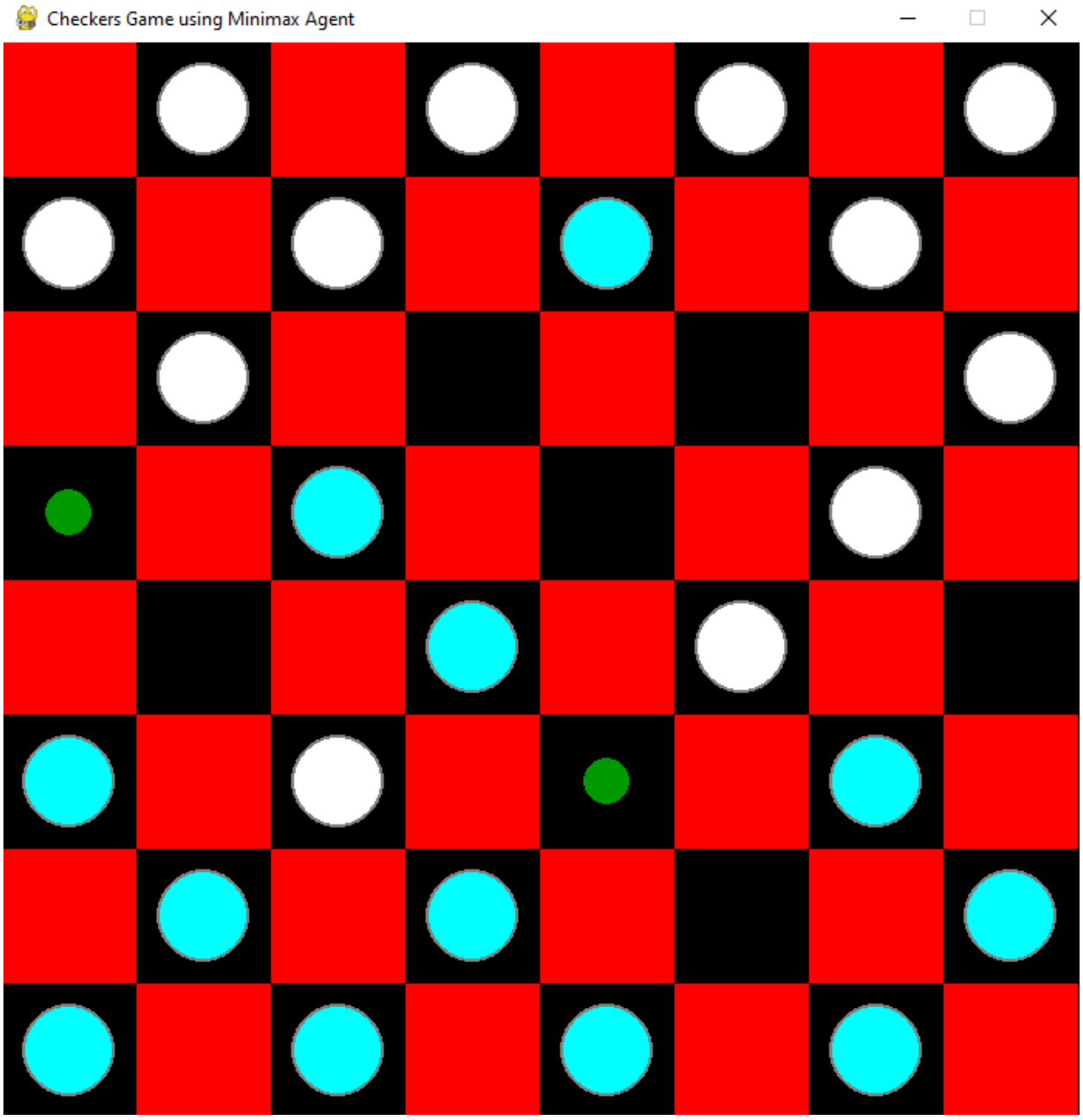



**Figure: 25 Normal pieces double jump attack**

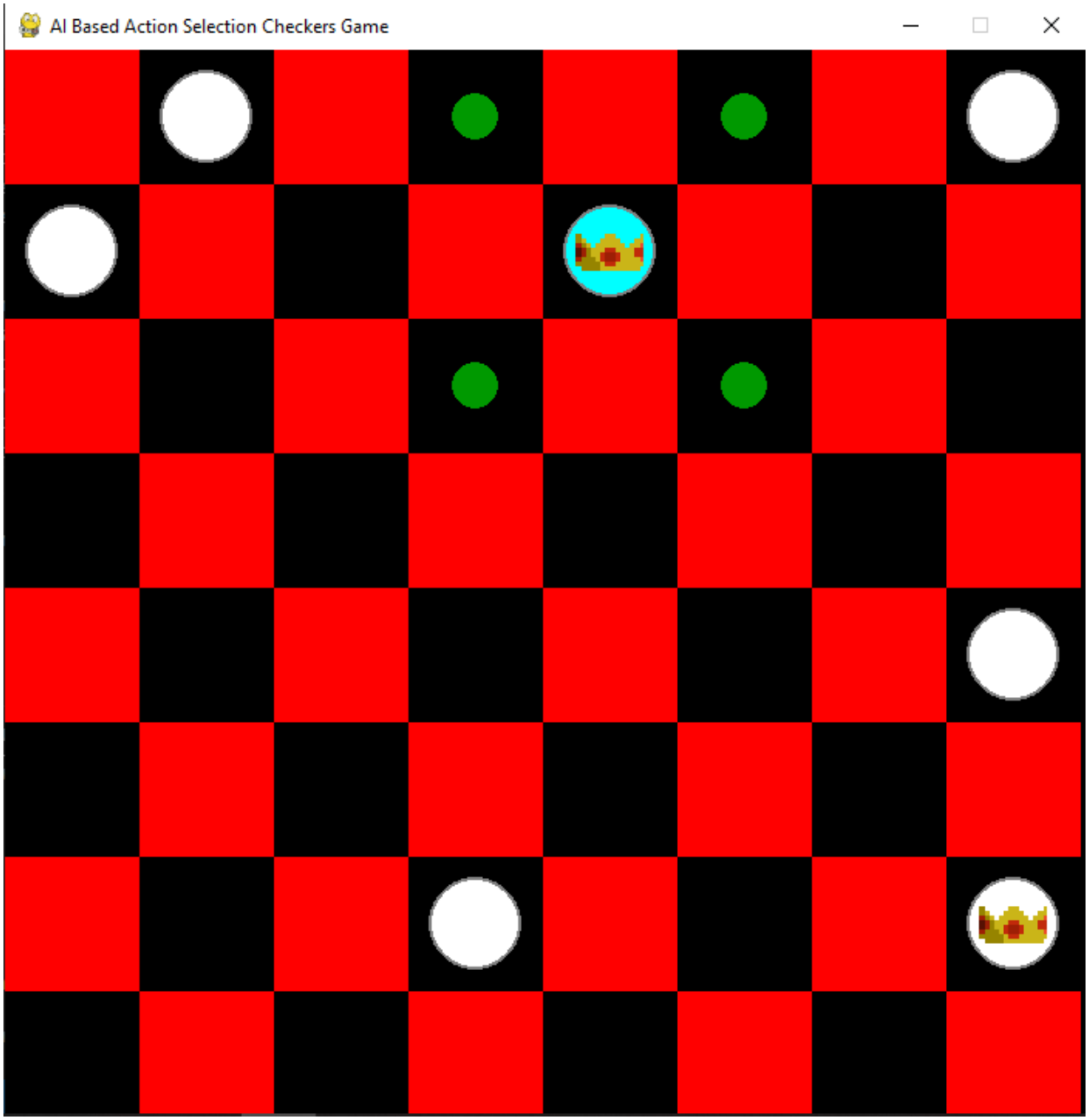



**Figure: 26 King has 4 options to attack**

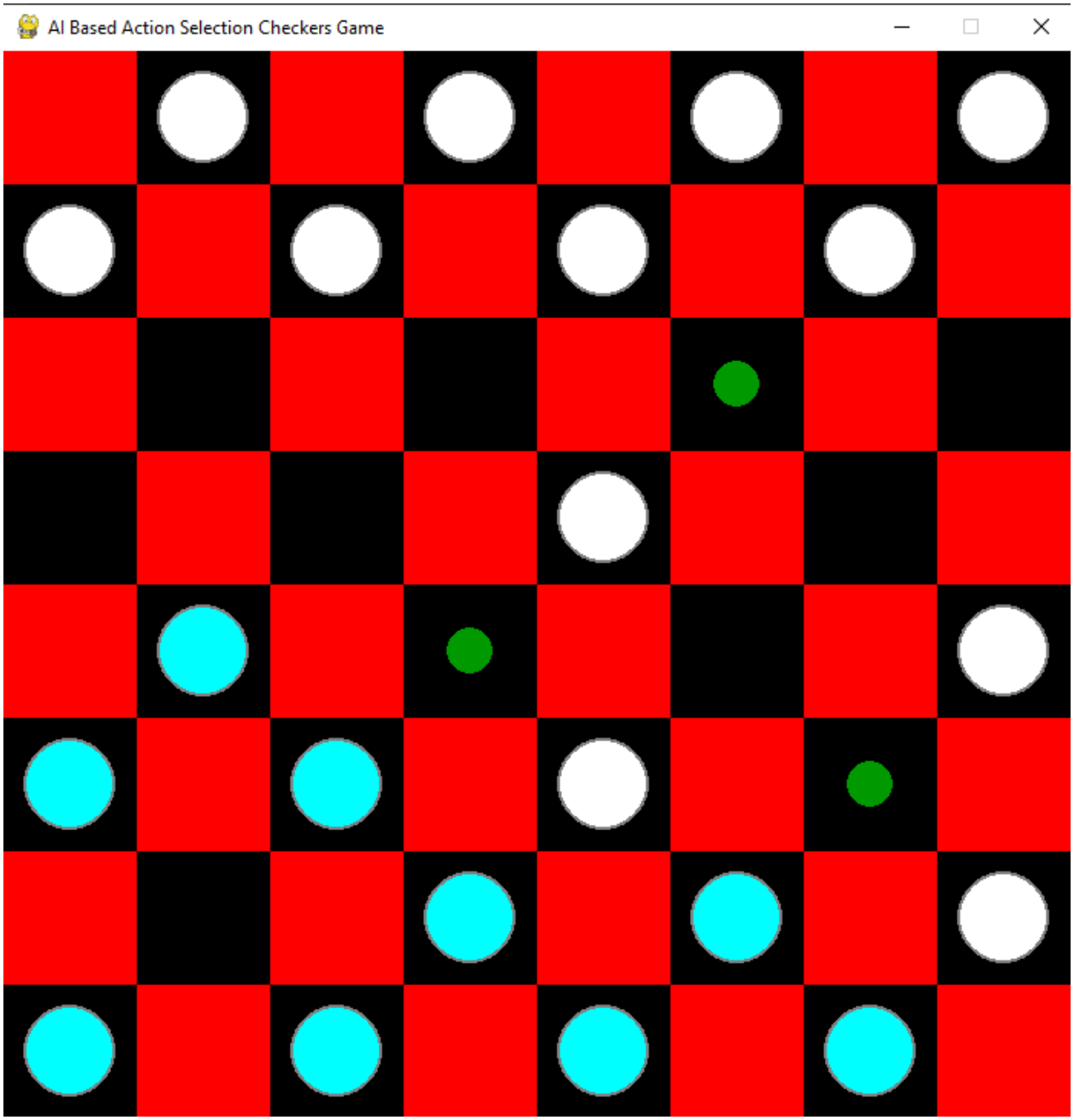



**Figure: 27 Human player has triple jump attack**

# 7 CONCLUSION AND FUTURE WORK

## 7.1 Conclusion

I have done my project according to the objectives of my project. I have Firstly defined my all the modules and then one by one I have completed them. So, In My Project Mainly I am giving the best Interface to users to play and the main menu will have the instructions of the pieces and hence the users can start playing the game. And Even I am providing the Facility to Human player to perform the most optimistic and best attack against its opponent (the AI) by having single, multi, or triple jumps, backward movements after it has been a king piece.

## 7.2 Limitations of the system

- When a king piece is assigned it can perform backward jumps but a normal piece can also attack the king piece.
- Another Limitation is that Human Player have to keep track of their own turns.
- When any of the players have won the game, the game will be closed itself.

## 7.3 Future Scope of the project

- First Future Enhancement will be to add modify the user interface and attraction so the users can have more fun while playing.
- Another future enhancement will be that the users will be able to play with human players online on the web.
- The system can make the game do pause, play and resume options in the middle of the game

# REFERENCES


- SourceCodester. (2021, January 9). Pygame Menu Selection Tutorial Demo. YouTube. https://www.youtube.com/watch?v=EHn3u6-yUHs
- RKT. (2019, December 23). *How to Code a Simple Main Menu for your Python 3 games!* YouTube. https://www.youtube.com/watch?v=ZpxtH5TjjVM
- Coded Harsh. (2020, August 26). *Pygame Tutorial 2: Creating Game Screen and Adding Background Image*. YouTube. https://www.youtube.com/watch?v=H_7cXZgH4S8
- Tech With Tim. (2020, September 5). *Python/Pygame Checkers Tutorial (Part 1) - Drawing the Board*. YouTube. https://www.youtube.com/watch?v=vnd3RfeG3NM
- Sebastian Lague. (2018, April 20). *Algorithms Explained – minimax and alpha-beta pruning*. YouTube. https://www.youtube.com/watch?v=l-hh51ncgDI
- Keith Galli. (2018, July 14). *How does a Board Game AI Work? (Connect 4, Othello, Chess, Checkers) - Minimax Algorithm Explained*. YouTube. https://www.youtube.com/watch?v=y7AKtWGOPAE
- Indian Pythonista. (2019, March 27). *Creating Menu Display for Terminal | Intro to curses in Python (Part-2)*. YouTube. https://www.youtube.com/watch?v=zwMsmBsC1GM
- Phil Adams. (2015, May 25). *20. Creating a menu-based program using functions in Python*. YouTube. https://www.youtube.com/watch?v=f3D-w6XMTN8&feature=youtu.be
- Coding With Russ. (2020, October 2). *PyGame Button tutorial with text - Coding in Python*. YouTube. https://www.youtube.com/watch?v=jyrP0dDGqgY&feature=youtu.be