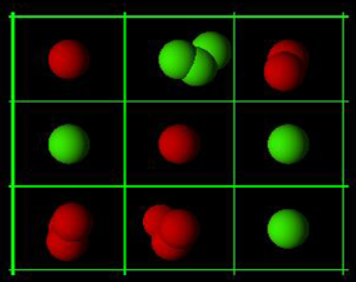
**CHAIN REACTION**



PROJECT REPORT ON

**Chain Reaction Game**

UNDER THE GUIDANCE OF

**Ms. KRUPA KAMDAR**

SUBMITTED BY

**Tanisha Gondke**

**TYCS18**

UNIVERSITY OF MUMBAI

T.Y.B.Sc (COMPUTER SCIENCE)

ACADEMIC YEAR: 2021-2022



BHARTIYA VIDYA BHAVAN’S

MM COLLEGE OF ARTS, NM COLLEGE OF COMMERCE &

HRJ COLLEGE OF COMMERCE BHAVAN’S COLLEGE

MUNSHI NAGAR, ANDHERI (WEST),

MUMBAI –400058



BHARTIYA VIDYA BHAVAN’S

MM COLLEGE OF ARTS, NM COLLEGE OF COMMERCE &

HRJ COLLEGE OF COMMERCE

BHAVAN’S COLLEGE MUNSHI NAGAR,

ANDHERI (WEST), MUMBAI – 400058

BHARATIYA VIDYA BHAVAN

BHAVAN’S COLLEGE, ANDHERI (W)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**CERTIFICATE**

This is to certify that **Ms. Tanisha Gondke**, Seat Number **TYCS18** of the class T. Y. B. Sc. Computer Science has satisfactorily completed the practical course in Group 6 as prescribed by the University of Mumbai during the academic year 2021 – 22.

Signature Signature

Staff in charge Computer Science Coordinators

Signature

Examiner College Stamp

**ACKNOWLEDGEMENT**

I would like to express my sincere gratitude towards the Computer Science Department of Bhavan’s College.

After months of hard work, finally I am very happy to present my final year Project. The Project making was full of new experiences and learning and difficult one too. Though a difficult job it was made simpler by the timely guidance received, which helped me greatly in the completion of my project. But it wouldn’t be right to do so without thanking to those who have helped me in converting our thought into reality. So I would like to take full advantage of this opportunity to thank each and every person who has helped me throughout the completion of our project.

I am obliged to my parents & family members who always support me greatly and encouraged me in each and every step. I give my special thanks and sincere gratitude towards the Principal Dr (Mrs.) Z. P. Bhathena, Head of SFC Prof. (Mrs.) Lata Dolke and Head of Department (Computer Science) Prof. Mrs. Krupa Balsara-Kamdar.

I owe my sincere thanks to our Project guide Mrs. Krupa Kamdar for her constant support and encouragement without which the successful completion of this project would have been impossible. She has been instrumental for making me concentrate and focus my effort in this project. And last but not the least the entire college staff specially Mr. Prathamesh N.Rane.

I would like to give thanks to my Project guide Prof. Mrs. Krupa Balsara-Kamdar for helping me in my project.

Finally I would like to thank each and every individual who was directly or indirectly contributing for this project.

-Thank you.

**TYCS SEM-VI PROJECT INDEX**

|  |  |  |  |
| --- | --- | --- | --- |
| Sr. No | Topic | Page No. | Sign and Remark |
|  |  | 6 |  |
| 1 | Abstract and Keywords |  |  |
|  |  |  |  |
| 2 | Introduction | 7 |  |
| 2.1 | Problem Statement | 7 |  |
| 2.2 | Literature Review / Description of Present System | 7 |  |
| 2.3 | Background / Limitations | 7 |  |
| 2.4 | Aim and Objective | 7 |  |
| 2.5 | Project Motivation | 7 |  |
|  |  |  |  |
| 3 | Description of Proposed Work | 8 |  |
| 3.1 | Number of Modules | 8 |  |
| 3.2 | Algorithm | 8 |  |
| 3.3 | Working | 8 |  |
| 3.4 | Design / Block Diagram / Flow chart / Graph / Deployment Diagram / Architectural Diagram | 9 |  |
| 3.5 | Plagiarism Report | 10 |  |
| 3.6 | Coding | 10 |  |
| 3.7 | Screen Layouts | 17 |  |
|  |  |  |  |
| 4 | Technology / Language / Development Tools / Hardware | 18 |  |
|  |  |  |  |
| 5 | Conclusion and Future Scope | 18 |  |
|  |  |  |  |
| 6 | References / Resource Material / Data Collection | 18 |  |
|  |  |  |  |

**1. ABSTRACT AND KEYWORDS**

Main objective of the game chain reaction is to take control of the entire board by eliminating your opponents' orbs. The cell, when reached its mass critical stage; the orb explodes into its surrounding area adding an extra orb. This claims that particular cell for the player. Interesting part of this game is that it’s unpredictable. We should focus to find out the right movements and combinations of chain reactions to win the game. There are 3 types of agents we can play with, this reflects to their difficulty level in the game. Each agent has its own algorithm to play the game.

**Keywords:**

Chain Reaction, python, pygame, Random, MCTS, Minimax

**2. Introduction**

**2.1 Problem Statement**

The game chain reaction is a game where the player has to get rid of all the opponent orbs in the board. The game is usually played with two human players. We should focus to find out the right movements and combinations of chain reactions to win the game.

**2.2 Description of the present system / Literature Review**

Chain reaction is a board game. The player places the orb in the board to claim that cell. When the maximum cell space is reached, the orbs explodes and moves into the other adjacent cell. Similarly, both the players have to claim their cells on the board and then one who gets rid of all the opponent orbs wins the game. A bot is introduced here.

**2.3 Background / Limitation**

The traditional chain reaction game is played on the board. The number of players are 2 and both are humans.

**2.4 Aim and Objective**

To enhance the game’s performance, the opponent player will be a bot based on AI. So, a single human player can play against AI. There is a freedom of playing against an AI or a human. The game will have 3 agents (bot) each based on 3 algorithms each. The algorithms will be made such that the level of difficulty of the game increases as we play with the respective agent. This makes the game more interesting as were playing with the bot.

**2.5 Project Motivation**

The motivation of this project is to boost / increase the competition of the players in the game called chain reaction.

**3. Description of the Proposed Work / Project**

**3.1 Number of Modules**

**1. Chain reaction basic game**: This consists of the traditional chain reaction game having 2 human players.

**2. Random**: This contains a bot (agent) which plays its move on the board at random.

**3. MCTS**: This contains a bot (agent) which plays its move on the board using MCTS algorithm.

**4. Minimax**: This contains a bot (agent) which plays its move on the board using Minimax algorithm.

**3.2 Algorithm**

**1. Random**

This is made using random module which was imported in the code. It uses the normal valid board moves logic but just plays them at random.

**2. MCTS**

Monte Carlo Tree Search is a method usually used in games to predict the path (moves) that should be taken by the policy to reach the final winning solution. It is a slow learner. It has to observe all the states in present and chooses the optimal one not knowing if the state will really lead to an optimal solution or not. It follows a sequence of steps until it reaches the final solution, also learning the policy of the game; and those steps are: Selection, Expansion, Simulation, Back Propagation.

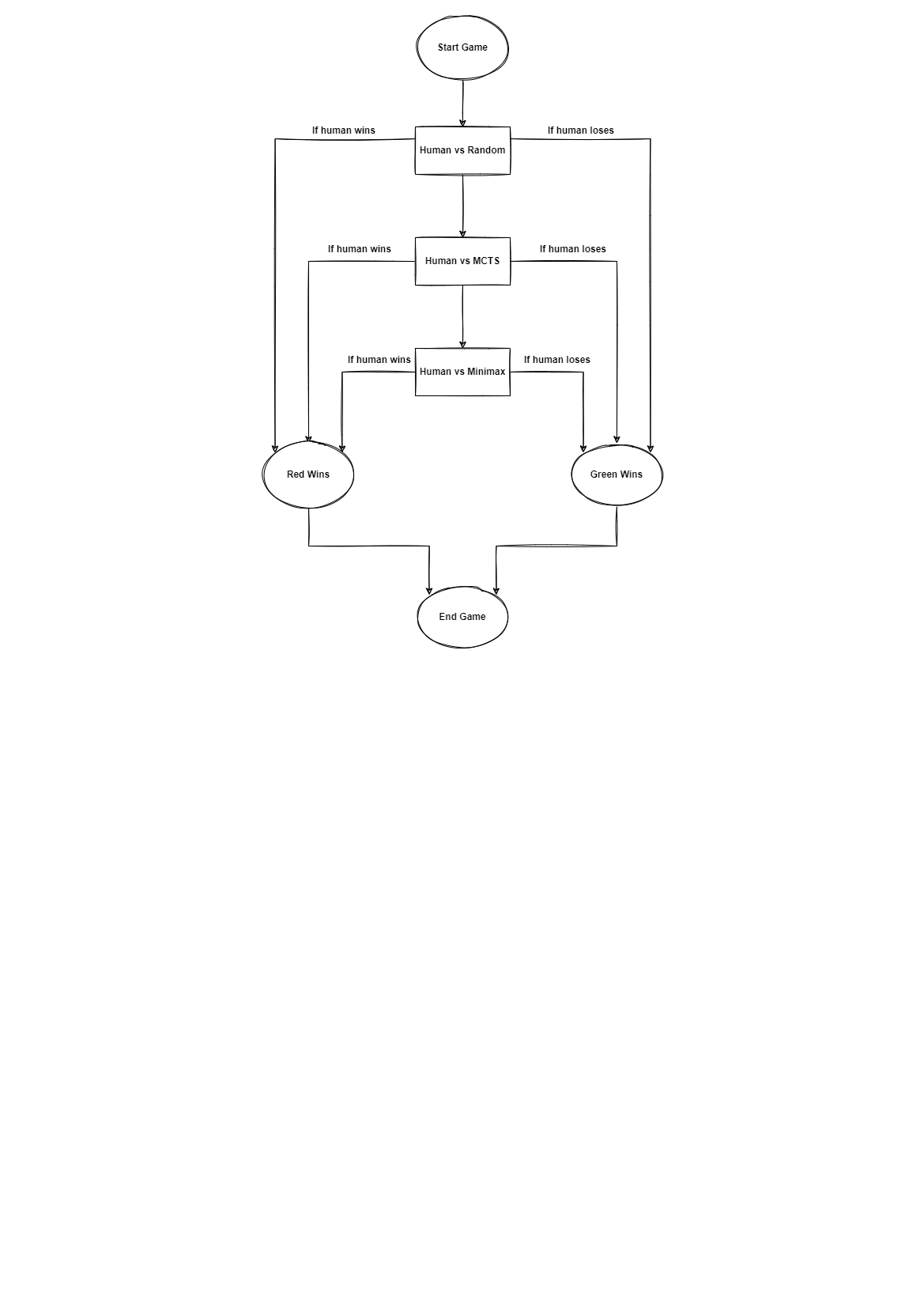
**3. Minimax**

This algorithm follows the look ahead decision strategy. It assumes that the opposite player is playing optimally, thus it tries to play even better with the intention of defeating the enemy player. This algorithm is used when there are two player games this type of game is called zero sum game concept. It contains a minimizer and maximize. They choose the best move and worst move for the agent respectively. It also consists of Alpha Beta Pruning where the tree branch that is useless is eradicated. It traverses in a DFS fashion down the tree. It selects the move which gives the best utility value.

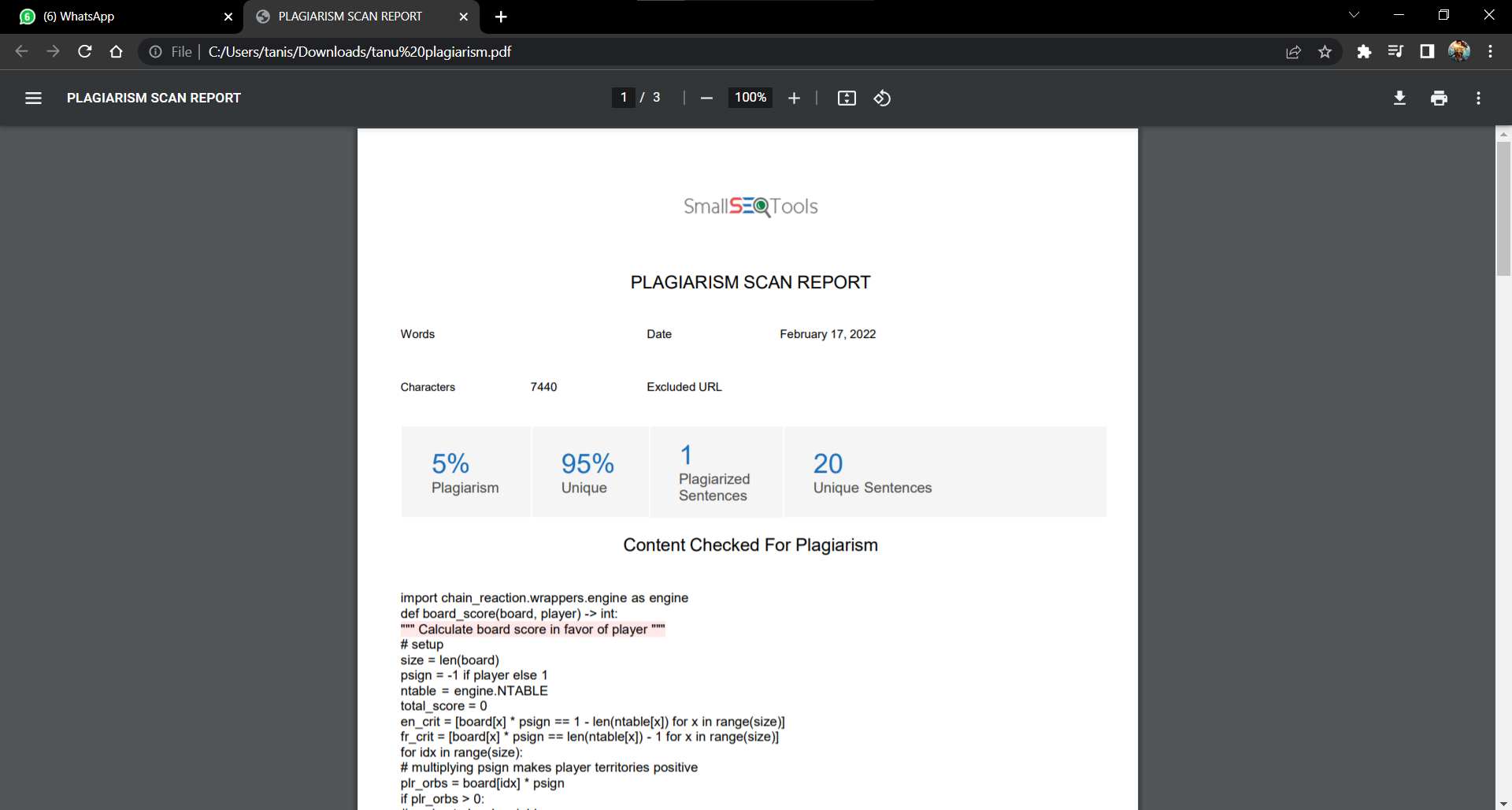
**3.3 Working**

The user can change the grid dimensions by changing the shape attribute in the code. The player1 and player2 can have values: “human, random, mcts, minimax”. The game then starts by the settings you’ve set. The difficulty increases as the levels increase. Level 1 is the random agent. Level 2 is MCTS and level 3 is Minimax. The players have to place their atoms in the grid in alternate turns and try to eliminate the atoms of the other players. These atoms have valency upto 3 but it depends on what position they’re placed in. If it’s a corner the atom has valency 1; if an edge then valency is 2, if in between then the valency is 3. When these valency get full, these atoms burst in the board in x and y plane (2d). The player (we) have to defeat the ai agent by placing our atoms on the board and eliminating all the enemies’ atoms. The atoms burst and occupy adjacent cells leading to a chain reaction. Finally, a wintext frame will appear at the end of the game when a player wins.

**3.4 Design / Block diagram / flowchart / graph / deployment diagram / architectural design**

****

**3.5 Plagiarism Report**

****

**3.6 Coding**

**1. MCTS**

import time

import math

import random

import chain\_reaction.wrappers.engine as engine

def uct\_score(node, c\_param) -> int:

exploit = node.qscore / node.visits

explore = math.sqrt(math.log(node.parent.visits, 10) / node.visits)

return exploit + c\_param \* explore

def forward\_roll\_once(state, player) -> tuple:

valid\_moves = engine.valid\_board\_moves(state, player)

chosen\_move = random.choice(valid\_moves)

next\_state = state[:]

game\_over = engine.interact\_inplace(next\_state, chosen\_move, player)

if game\_over:

return (None, player)

else:

return (next\_state, 1 - player)

class MCTSVisitedNode:

def \_\_init\_\_(self, state, parent, index, player):

self.state = state

self.index = index

self.player = player

self.parent = parent

self.children = []

self.unvisited = engine.valid\_board\_moves(state, player) if state else []

self.is\_terminal = False if state else True

self.visits = 0

self.qscore = 0

def is\_fully\_expanded(self):

return len(self.unvisited) == 0

def expand(self):

action = self.unvisited.pop()

next\_state = self.state[:]

game\_over = engine.interact\_inplace(next\_state, action, self.player)

next\_state = None if game\_over else next\_state

child = MCTSVisitedNode(next\_state, self, action, 1 - self.player)

self.children.append(child)

return child

def best\_child(self, c\_param):

b\_score = -math.inf

b\_child = None

for child in self.children:

score = uct\_score(child, c\_param)

if score > b\_score:

b\_score = score

b\_child = child

return b\_child

def simulate(self):

state = self.state

player = self.player

while state is not None:

state, player = forward\_roll\_once(state, player)

return player

def backpropagate(self, reward):

node = self

while node is not None:

node.visits += 1

node.qscore += 1 if node.player == reward else -1

node = node.parent

class MCTSRootNode(MCTSVisitedNode):

def \_\_init\_\_(self, state, player):

super().\_\_init\_\_(state, None, None, player)

def best\_action(self):

return self.best\_child(c\_param=0.0).index

def tree\_policy(self, c\_param):

node = self

while not node.is\_terminal:

if not node.is\_fully\_expanded():

return node.expand()

else:

node = node.best\_child(c\_param)

return node

def best\_action(board: list, player, time\_limit, c\_param) -> int:

time\_start = time.perf\_counter()

rootnode = MCTSRootNode(board, player)

while time.perf\_counter() - time\_start < time\_limit:

leafnode = rootnode.tree\_policy(c\_param)

reward = leafnode.simulate()

leafnode.backpropagate(reward)

return rootnode.best\_action()

**2. Minimax**

import chain\_reaction.wrappers.engine as engine

def board\_score(board, player) -> int:

size = len(board)

psign = -1 if player else 1

ntable = engine.NTABLE

total\_score = 0

en\_crit = [board[x] \* psign == 1 - len(ntable[x]) for x in range(size)]

fr\_crit = [board[x] \* psign == len(ntable[x]) - 1 for x in range(size)]

for idx in range(size):

plr\_orbs = board[idx] \* psign

if plr\_orbs > 0:

neighbrs = ntable[idx]

is\_critc = fr\_crit[idx]

maxcp = len(neighbrs)

crit\_enemies = sum([en\_crit[nid] for nid in neighbrs])

crit\_friends = sum([fr\_crit[nid] for nid in neighbrs])

total\_score += plr\_orbs

total\_score -= (5 - maxcp) \* crit\_enemies

if not crit\_enemies:

total\_score += 3 if maxcp == 2 else 0

total\_score += 2 if maxcp == 3 else 0

total\_score += 2 if is\_critc else 0

if is\_critc and crit\_friends:

total\_score += 2

return total\_score

def score\_minimizer(board, player, alpha, beta) -> int:

enemy = 1 - player

esign = -1 if enemy else 1

score = 10000

for idx in range(len(board)):

if board[idx] \* esign < 0:

continue

cboard = board[:]

if engine.interact\_inplace(cboard, idx, enemy):

return -10000

cscore = board\_score(cboard, player)

score = min(score, cscore)

beta = min(beta, score)

if alpha >= beta:

return score

return score

def pruned\_minimizer(board, player, alpha, beta, depth) -> int:

enemy = 1 - player

esign = -1 if enemy else 1

score = 10000

if depth == 0:

return score\_minimizer(board, player, alpha, beta)

for idx in range(len(board)):

if board[idx] \* esign < 0:

continue

cboard = board[:]

if engine.interact\_inplace(cboard, idx, enemy):

return -10000

cscore = pruned\_maximizer(cboard, player, alpha, beta, depth)

score = min(score, cscore)

beta = min(beta, score)

if alpha >= beta:

return score

return score

def pruned\_maximizer(board, player, alpha, beta, depth) -> int:

score = -10000

psign = -1 if player else 1

for idx in range(len(board)):

if board[idx] \* psign < 0:

continue

cboard = board[:]

if engine.interact\_inplace(cboard, idx, player):

return 10000

score = max(

score, pruned\_minimizer(cboard, player, alpha, beta, depth - 1)

)

alpha = max(alpha, score)

if alpha >= beta:

return score

return score

def load\_scores(board, player, depth) -> list:

alpha = -10000

psign = -1 if player else 1

score\_list = [0] \* len(board)

for idx in range(len(board)):

if board[idx] \* psign < 0:

score\_list[idx] = -20000

continue

cboard = board[:]

game\_over = engine.interact\_inplace(cboard, idx, player)

if game\_over:

score\_list[idx] = 10000

return score\_list

score = pruned\_minimizer(cboard, player, alpha, 10000, depth - 1)

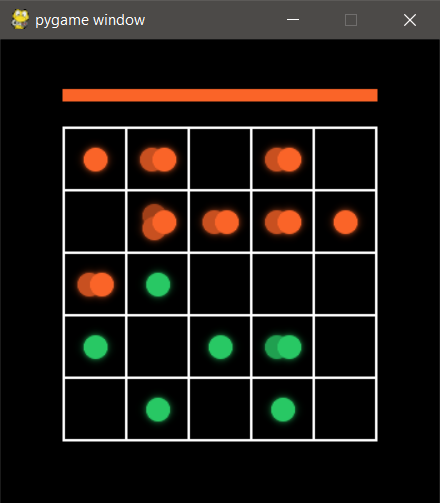
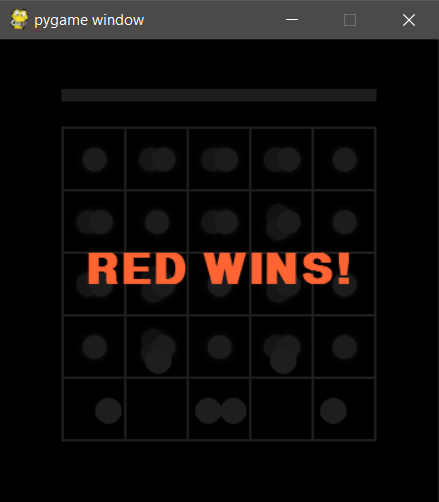
score\_list[idx] = score

alpha = max(alpha, score)

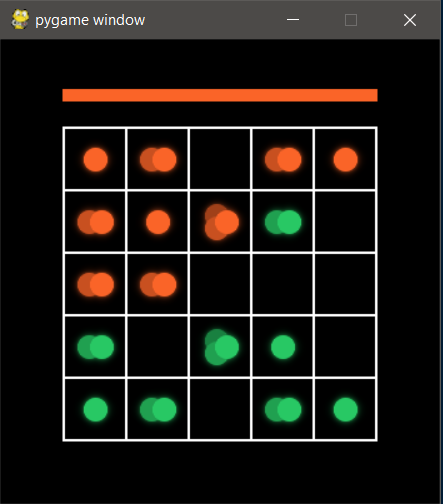
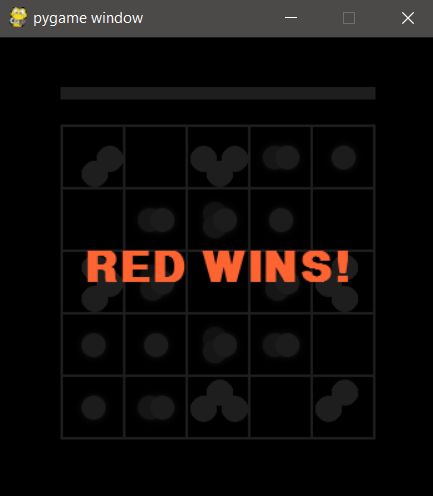
return score\_list

**3.7 Screen Layouts**

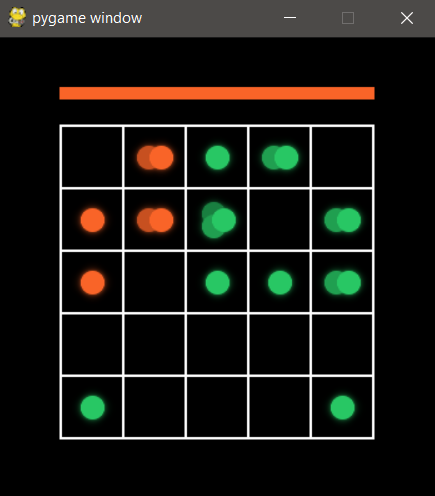
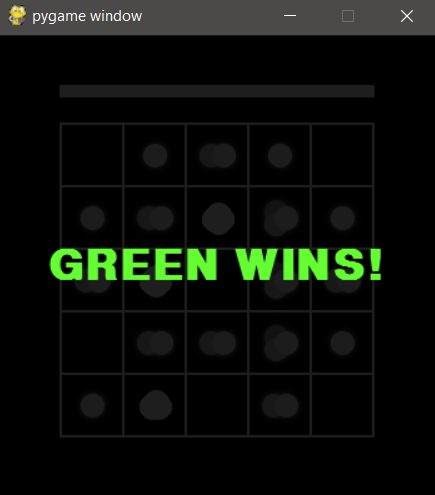
**1. Human (Red) vs Random (Green) – Easy to defeat random**

** **

**2. Human (Red) vs MCTS (Green) – Slow learner**

** **

**3. Human (Red) vs Minimax (Green) – Difficult to defeat minimax**

** **

**4. Technology / Language / Development Tools / Hardware**

1. Python - Pygame

**5. Conclusion and Future Report**

The traditional chain reaction game is modified and upgraded where we can have an ai agent playing against us human. There are 3 agents and each play with their own strategy in this game. In future, we can have an increased number of players in the board and can also play it online. Other entertainment utilities like song can also be added.

**6. References / Resource Material / Data collection**

<https://www.udemy.com/course/python-game-developmenttm-build-5-professional-games/>

<https://jatinmandav.wordpress.com/2017/09/27/chain-reaction-in-python/>

<https://www.baeldung.com/java-monte-carlo-tree-search>

<https://www.freecodecamp.org/news/minimax-algorithm-guide-how-to-create-an-unbeatable-ai/>