



DOMINATORS

- **NAYAN SHIVHARE**
- **GABRIEL PROCTOR**
- **JUNSHI WEI**

HALMA PROJECT

**ADVANCED INTELLIGENT SYSTEMS
(SPRING 2021 M16) 001**

18-04-2021

• Overview:

Brain phase:

As in phase 1, we created an interface and all the functionalities for the halma game so that Two players can play the game. We created a gridded interface and some small functionalities which return the important information such as valid moves and jumps. Now in the brain phase, we created a brain for halma so that a player can play the game with AI. By using all the previously built functions it becomes an easy task to create a brain because we have mostly everything that we need to make a brain for AI.

As we created a gridded interface we have the coordinates of every tile so we added some more functionalities like Calculating the distance from tile to the goal by using the distance formula. This is used by AI to calculate the distance for its piece/token from a tile to the goal tile.

We created a Minimax algorithm that is the main brain of our AI player. We used minimax as it's a kind of backtracking algorithm that finds the optimal move in games such as Chess, Halma. It has a maximizer and a minimizer. Maximizer tries to get the highest score and minimizer tries to get the lowest score. Using the minimax algorithm our AI generates the best optimal move over all the possible valid moves. Once AI gets the best move it returns the move and our AI plays that move within a time limit.

Once we implemented min-max we added Alpha-beta pruning which is an optimization for our minimax algorithm. It helps in reducing the calculation, saves time, and gets the moves as fast as possible.

Alpha Beta pruning shows us very promising results as we tested when making the first move when alpha-beta is off AI will take about 6 to 7 seconds to make the move.

And when we turned on Alpha-beta pruning AI takes about 3 to 4 seconds to make the same move.

When player 1 is making a move then player 1 has some time limit to make a move. As player 1 is done with a move. Then our algorithm shifts the turn to the AI if ai is playing. Then AI gets all the possible moves recursively using minimax gets the best move and makes a move. Once done with the move its status is shown in a bar that we added newly.

This new bar that we added shows the status of Ai such as from which tile to which tile AI played the game.

Whoever wins the game. The Player's name is displayed as the winner and the game closes automatically.

NOTE: When alpha-beta pruning turns off AI takes 6 to 7 seconds to make the first move. And when Alpha-beta is turned on AI will take about 3 to 4 seconds to make the same move.

```
#initializing our main class to play game
#Parameters:
#Board size:8,10,16
#Timelimit : in seconds
#player1="RED"or "GREEN"
#player2="RED or GREEN"
#AI="AI" if you want Bot to play as player 2 else None for Human
#AlphaBeta= "ON" or "OFF"
# Always the second player is assigned as AI
game = HalmaGame(8, 10, "RED", "GREEN","AI","ON") #For Human vs AI
# game= HalmaGame(8,10,"RED","GREEN",None,"ON") # for human vs Human
```

The creation of our intelligent Halma agent(Phase 2 in Game Class):

distance_calculation()

This function calculates the distance from the current node to the goal node. This is used when AI is playing so that AI can calculate all the possible moves and returns the best move.

Calcuategoal()

This function calculates all the distance from a node to the all by calling the distance to calculate the function internally for every valid token.

CharTurnCord()

This function takes all the valid moves in input and returns the coordinates of each move.

Minimax()

This is our main algorithm. By using this algorithm our AI plays the game by calculating the best move. AI explores all the moves and returns the best move. It also uses alpha-beta pruning inside which can help in reducing a lot of time.

Player_Char_moves()

This function returns all the possible moves for a player. This function is used by AI to get all the possible moves.

Char_Move_Temp()

This function temporarily moves the token internally just for calculation it is used by ai to find the best possible moves.so it calculates all the possible moves by moving a token.

AiPlay()

This function handles all the functionality of the AI from start to end. It calls all the functions that are used by AI to play a move. Once the best move is returned it plays that move. And return the status of the movement.

Phase1:

class HalmaGame: This is the main class of the game. It assigns players and it is responsible to create the board internally to perform the calculations such as moves from one tile to another.

Play(): play function runs a loop until the game is not finished. Each player gets one chance to play one move if a user forgets to play his move within a time limit then he cannot make his move until his next turn.

Endgame(): This function checks which player won the game. And responsible to terminate the game.

Halmaguimode class: Gui Class which is made using Tkinter. This class handles all the functionality of GUI to show the moves, Show whose turn is currently. It also shows all the information if AI is playing that is from which tile to which tile ai has moved.

Timeren() This function is the timer of our game which is initialized every time when a player gets his turn. Users can change the timer limit at the start of the game.

Makegrid() This function is used to create a grid for visualization and calculation purposes.

Makechar() This function creates the tokens for each player.

Onpress() Function works on clicking a token of a game. if a player press one of his piece of token this function shows all the possible moves in the interface with a boundary Highlighted

And if a player presses another tile to move.

This function is responsible for making a move using the interface.

Game File

Game Class: Our Game Class handles and Generates moves in the back. It is also responsible to generate Jumps while making moves.

Checkjump()

Function to Check Jump on the board based On the current position of a player jumps can be made if other player is in a neighboring tile and the next tile to the other player is Empty

Validatemove()

this function checks that if a player is making a move then it is validated that player is making a correct move or not.

char_movement()

This is the Function that is used to move the token from one tile to another tile. Its response is recorded and referred in the Gui.

CharacterUtil Class.

CharacterUtil Class: This class is used to assign players a specific color, home, and goal coordinates.

Starposchar()

This function gets all the starting positions of the Players i.e goal, home coordinates.

all_char_opp()

This function checks that a game is finished or not. The game is finished when all the tokens of player 1 reach the goal. That is the home coordinate of player 2

charlocCheck()

This function checks the current location of the token.

charHomeCheck() Function to check the home coordinates.

charGoalCheck() Function to check the goal coordinates.

Effort Description.

Junshi Wei:

Minimax()

Created and implemented the Alpha-Beta Pruning called Minimax.

35% work done

Nayan Shivhare:

Combine all the things we created. And worked on implementing them on GUI with Gabriel.

35% work done

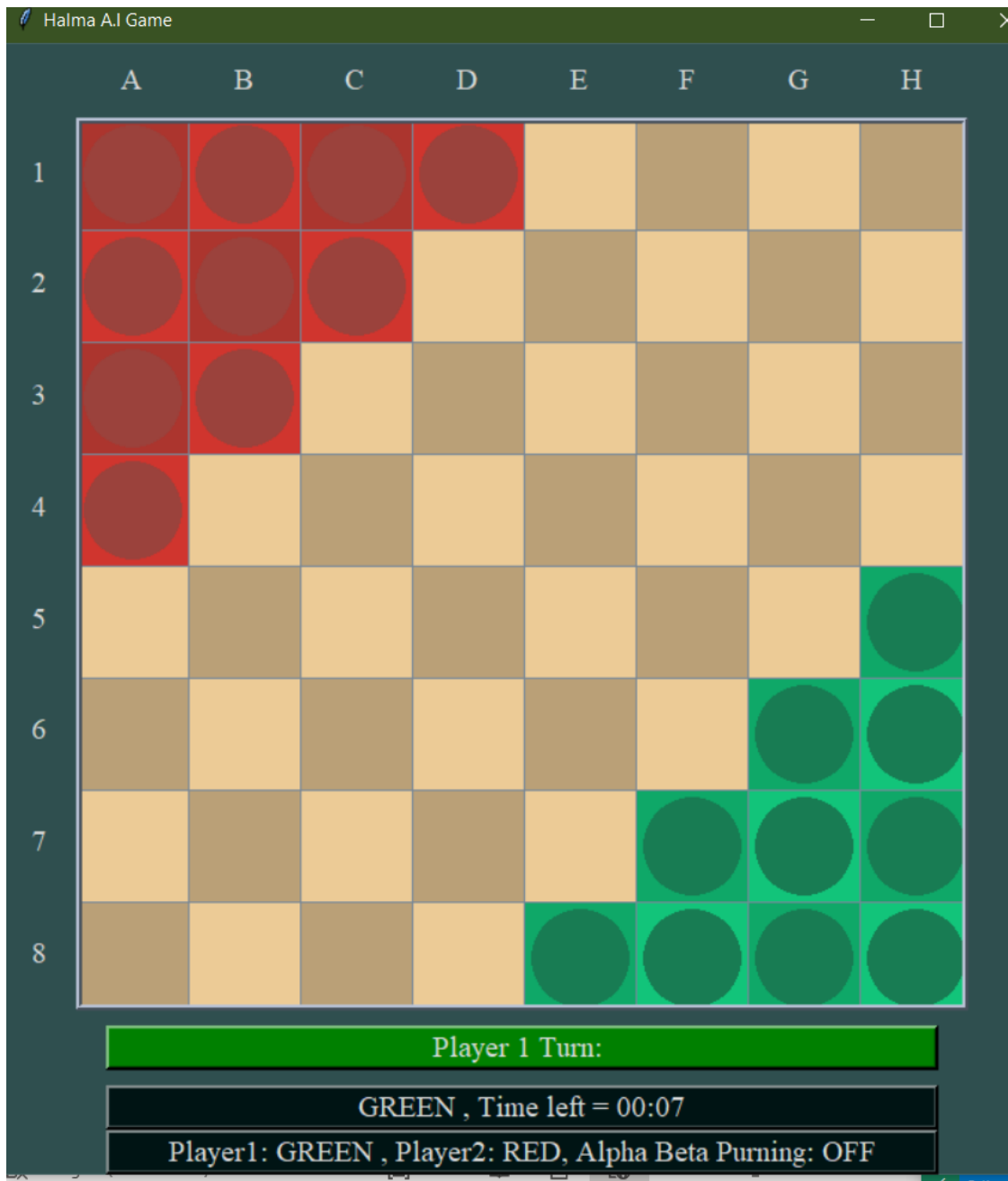
Gabriel Proctor:

Worked on GUI class to get the board extra functionality that is needed for phase 2.

30% work done

Functionality Checklist.

When Ab pruning is off: Red is AI and when an AI moves its turn it shows how much time is being taken on the grid as Time Taken.



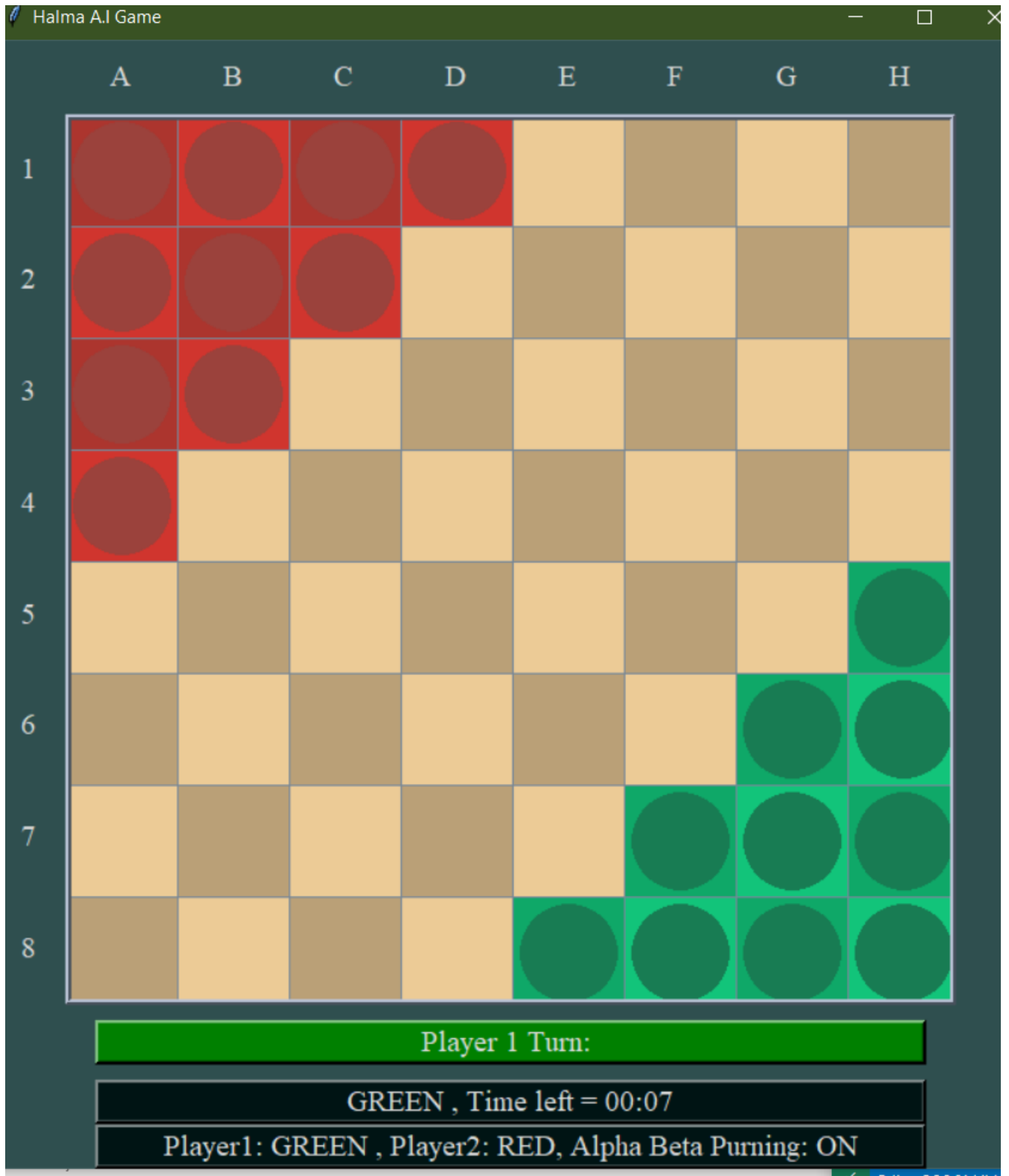
	A	B	C	D	E	F	G	H
1	Red	Red	Red	Red	Yellow	Green	Yellow	Green
2	Red	Red	Red	Yellow	Green	Yellow	Green	Yellow
3	Red	Red	Red	Green	Yellow	Green	Yellow	Green
4	Red	Yellow	Green	Yellow	Green	Yellow	Green	Yellow
5	Yellow	Green	Yellow	Green	Yellow	Green	Yellow	Green
6	Green	Yellow	Green	Yellow	Green	Yellow	Green	Yellow
7	Yellow	Green	Yellow	Green	Yellow	Green	Yellow	Green
8	Green	Yellow	Green	Yellow	Green	Yellow	Green	Yellow

Player 1 Turn:

GREEN , Time left = 00:07

AI moved From (1, 1) To (3, 3), Time Taken: 7 seconds.

When Ab pruning is on:-



Halma AI Game

	A	B	C	D	E	F	G	H
1								
2								
3								
4								
5								
6								
7								
8								

Player 1 Turn:

GREEN , Time left = 00:09

AI moved From (1, 1) To (3, 3), Time Taken: 4 seconds.

```

PLYER WHILE <charutils_py.CharacterUtil object at 0x000002C8E9B95C70>
YO 8
IF
GET CURENT (8, 8)
YO 8
Length 1
X,y, (6, 6)
YO 8
PLYER WHILE <charutils_py.CharacterUtil object at 0x000002C8E9B95C70>
deep 0
YO 8
RED
YO 8
RED
YO 8
RED
RED
YO 8
RED
RED
YO 8
RED
YO 8
RED
YO 8
RED
YO 8
RED
YO 8
RED
YO 8
AlphaBetaaaaaaaaaaaaaaaaa
YO 8
deep 1
YO 8
IF
YO 8
GET CURENT (4, 7)
Length 8
YO 8
X,y, (5, 7)
PLYER WHILE <charutils_py.CharacterUtil object at 0x000002C8E9B95C70>
X,y, (5, 8)
YO 8

```

```
YO 8
PLYER WHILE <charutils_py.CharacterUtil object at 0x000002C8E9B95C70>
YO 8
X,y, (4, 8)
PLYER WHILE <charutils_py.CharacterUtil object at 0x000002C8E9B95C70>
X,y, (3, 8)
YO 8
PLYER WHILE <charutils_py.CharacterUtil object at 0x000002C8E9B95C70>
X,y, (3, 7)
PLYER WHILE <charutils_py.CharacterUtil object at 0x000002C8E9B95C70>
X,y, (3, 6)
YO 8
PLYER WHILE <charutils_py.CharacterUtil object at 0x000002C8E9B95C70>
YO 8
X,y, (4, 6)
YO 8
PLYER WHILE <charutils_py.CharacterUtil object at 0x000002C8E9B95C70>
YO 8
X,y, (5, 6)
YO 8
PLYER WHILE <charutils_py.CharacterUtil object at 0x000002C8E9B95C70>
IF
YO 8
GET CURENT (6, 7)
YO 8
Length 4
X,y, (5, 8)
YO 8
PLYER WHILE <charutils_py.CharacterUtil object at 0x000002C8E9B95C70>
YO 8
X,y, (5, 7)
YO 8
PLYER WHILE <charutils_py.CharacterUtil object at 0x000002C8E9B95C70>
YO 8
X,y, (5, 6)
PLYER WHILE <charutils_py.CharacterUtil object at 0x000002C8E9B95C70>
YO 8
X,y, (6, 6)
YO 8
PLYER WHILE <charutils_py.CharacterUtil object at 0x000002C8E9B95C70>
IF
YO 8
```

Demos:

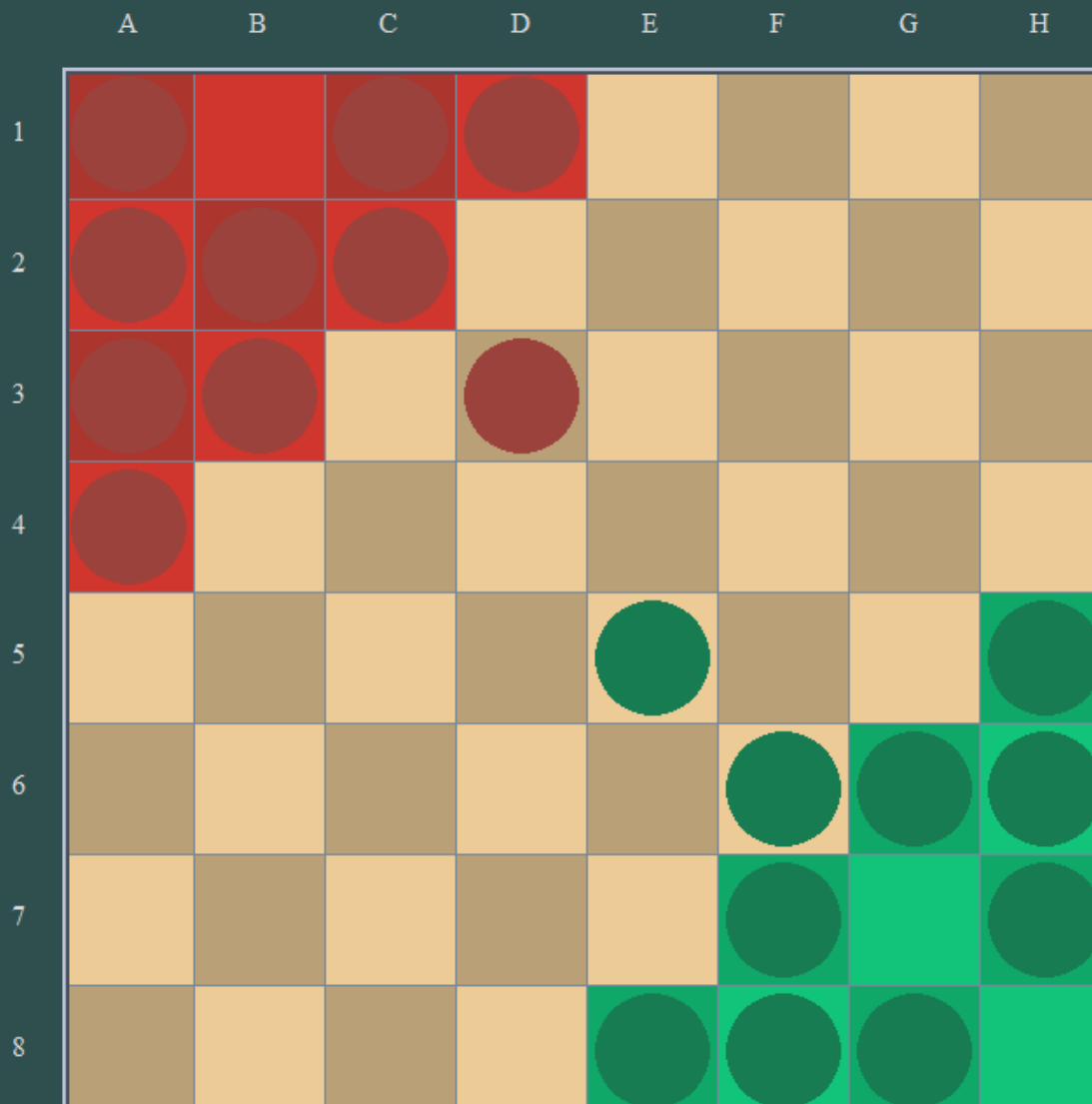
Halma A.I Game

	A	B	C	D	E	F	G	H
1	Red	Red	Red	Red	Yellow	Brown	Yellow	Brown
2	Red	Red	Red	Yellow	Brown	Yellow	Brown	Yellow
3	Red	Red	Yellow	Brown	Yellow	Brown	Yellow	Brown
4	Red	Yellow	Brown	Yellow	Brown	Yellow	Brown	Yellow
5	Yellow	Brown	Yellow	Brown	Yellow	Brown	Yellow	Green
6	Brown	Yellow	Brown	Yellow	Brown	Green	Green	Green
7	Yellow	Brown	Yellow	Brown	Yellow	Green	Green	Green
8	Brown	Yellow	Brown	Yellow	Green	Green	Green	Green

Player 1 Turn:

RED , Time left = 00:58

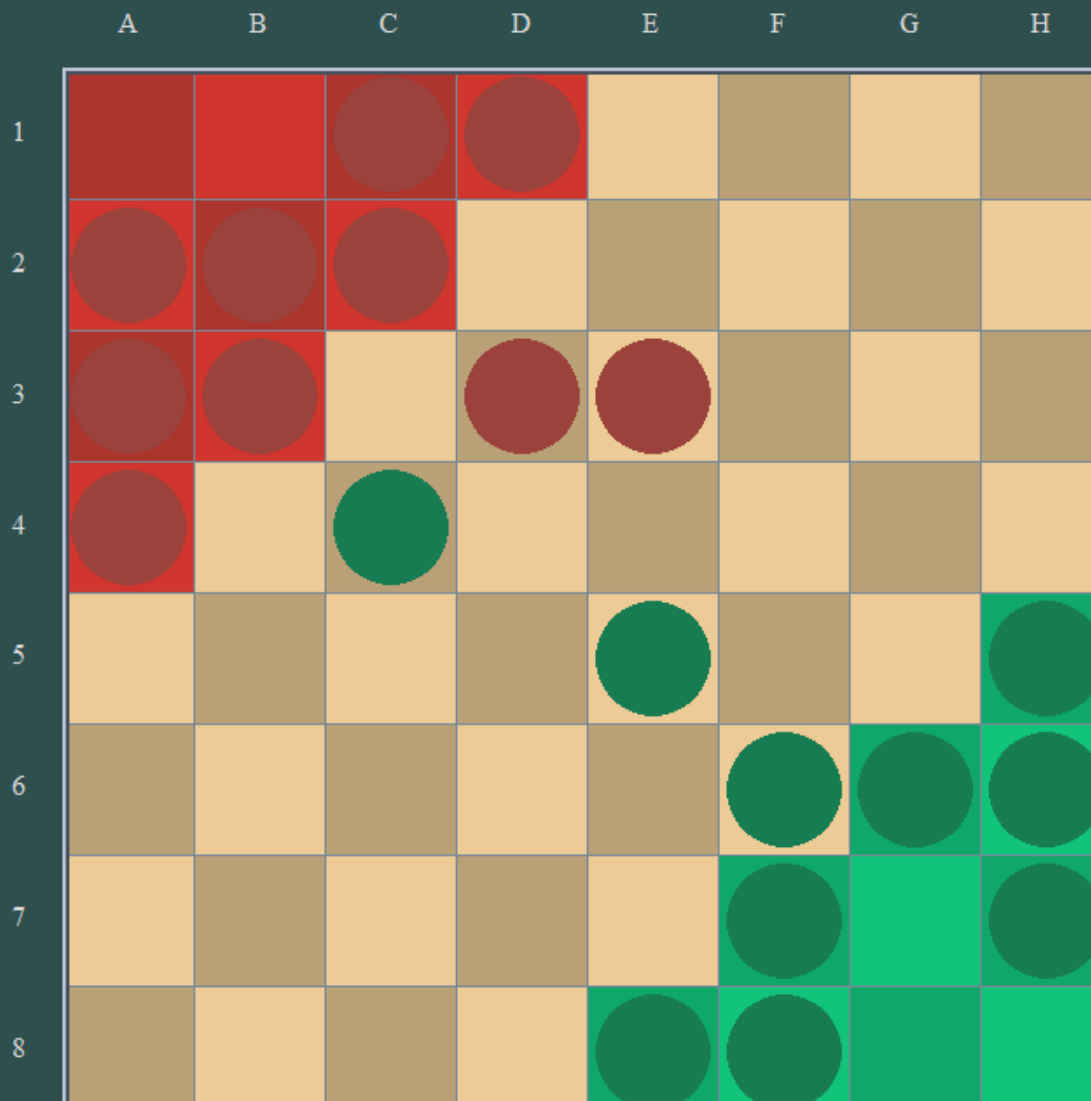
AI moved From (8, 8) To (6, 6), Time Taken: -48 seconds.



Player 1 Turn:

RED , Time left = 00:59

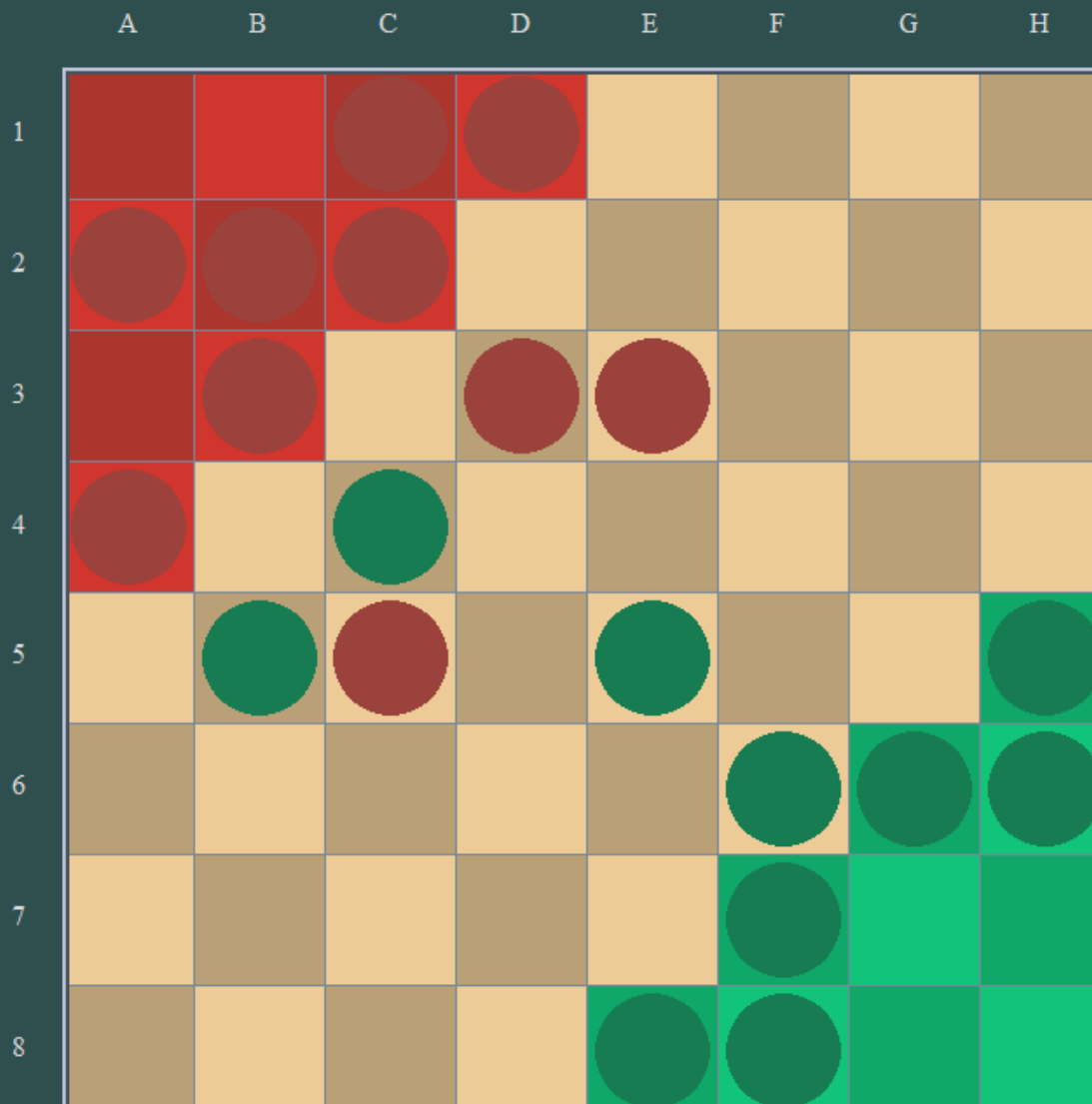
AI moved From (7, 7) To (5, 5), Time Taken: -48 seconds.



Player 1 Turn:

RED , Time left = 00:58

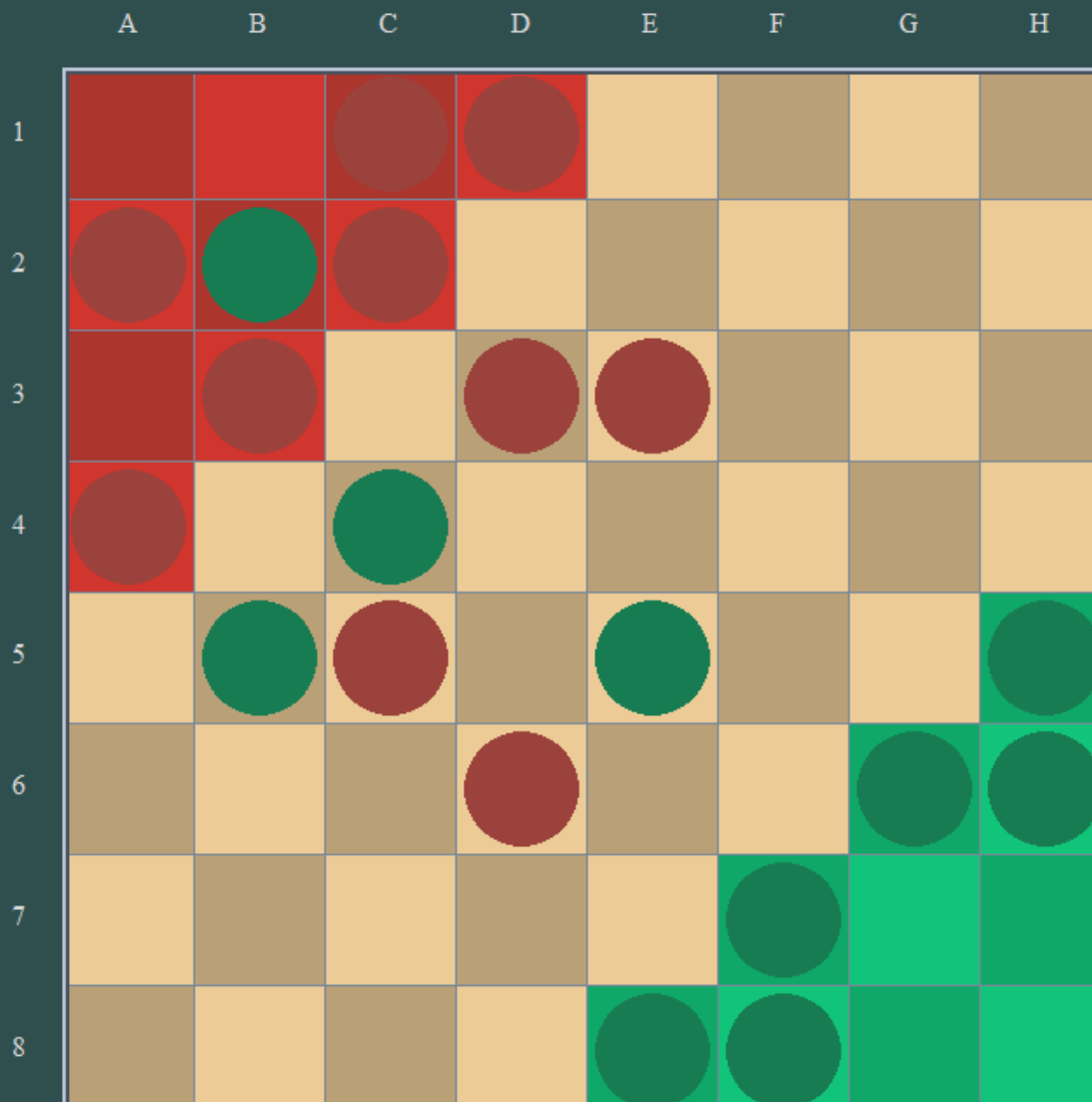
AI moved From (7, 8) To (3, 4), Time Taken: -48 seconds.



Player 1 Turn:

RED , Time left = 00:58

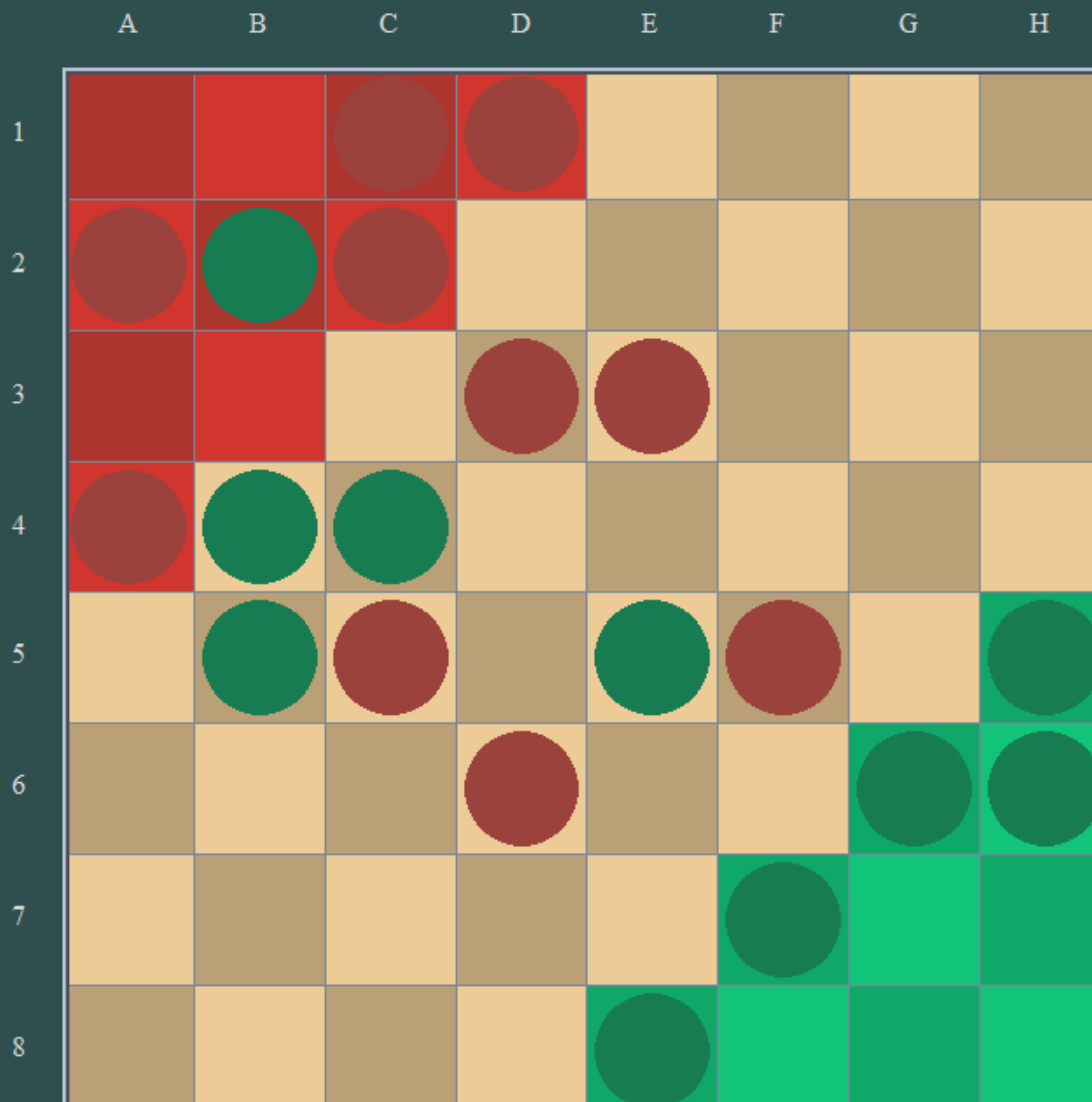
AI moved From (8, 7) To (2, 5), Time Taken: -48 seconds.



Player 1 Turn:

RED , Time left = 01:00

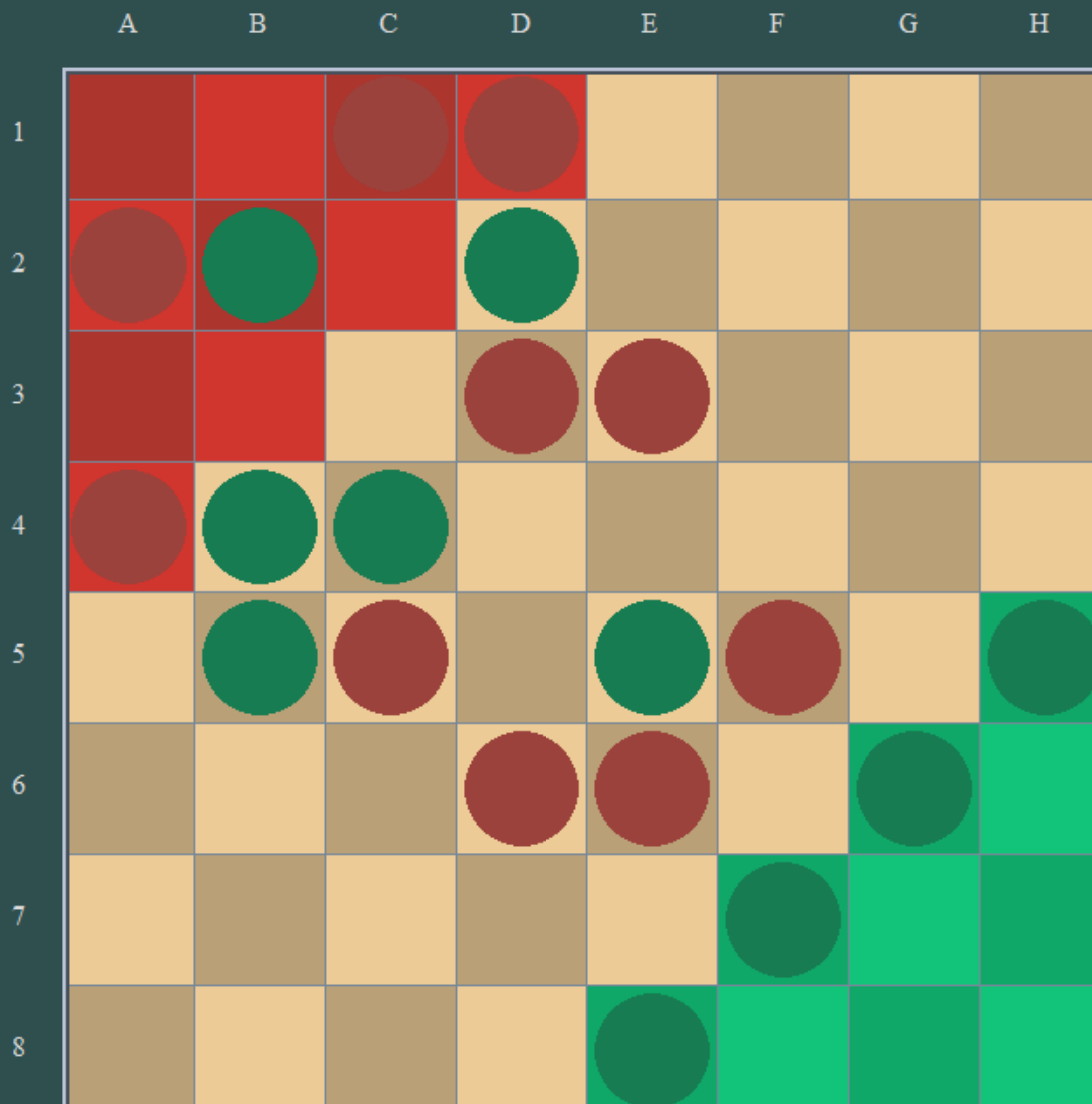
AI moved From (6, 6) To (2, 2), Time Taken: -48 seconds.



Player 1 Turn:

RED , Time left = 00:58

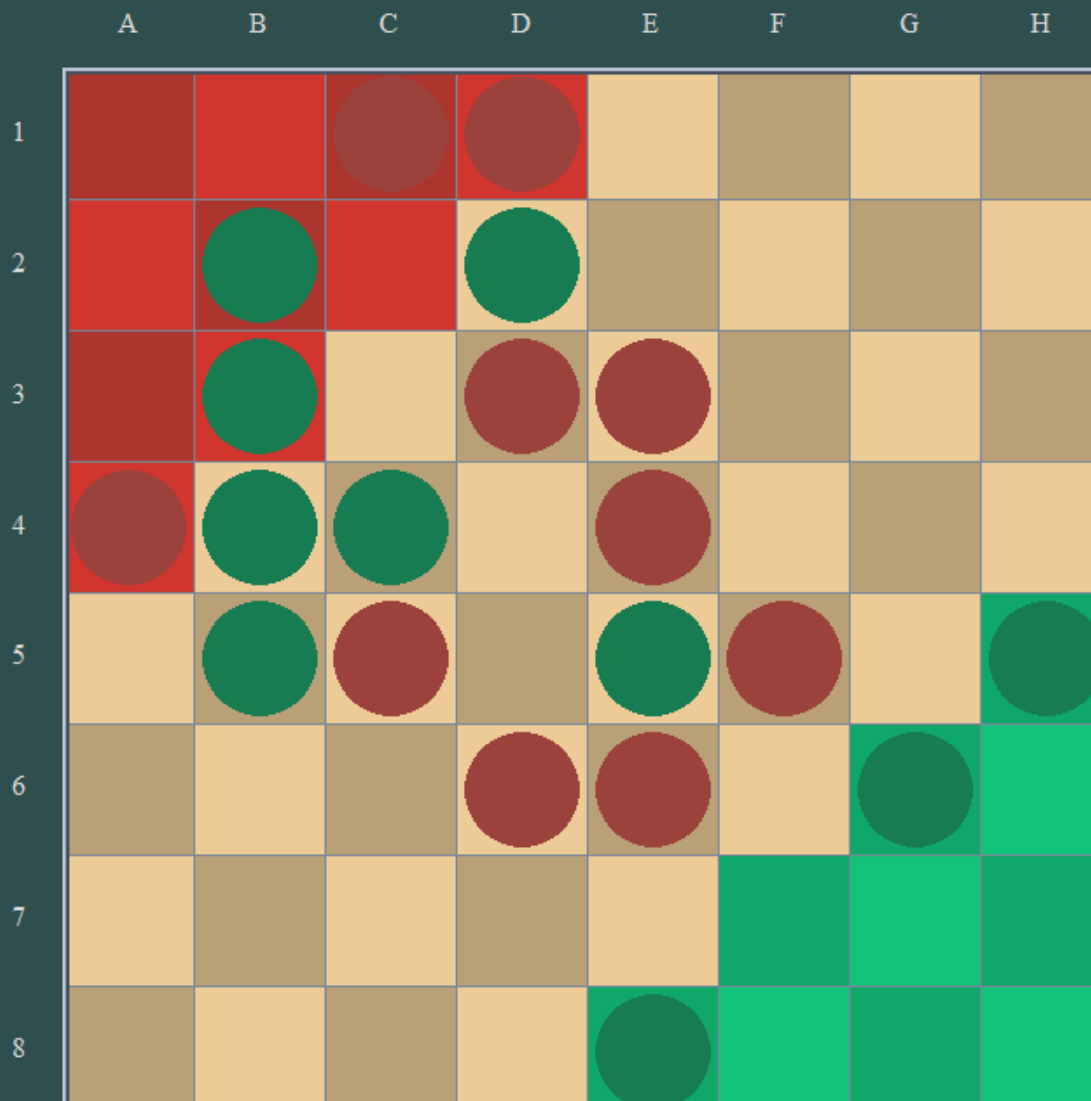
AI moved From (6, 8) To (2, 4), Time Taken: -48 seconds.



Player 1 Turn:

RED , Time left = 00:59

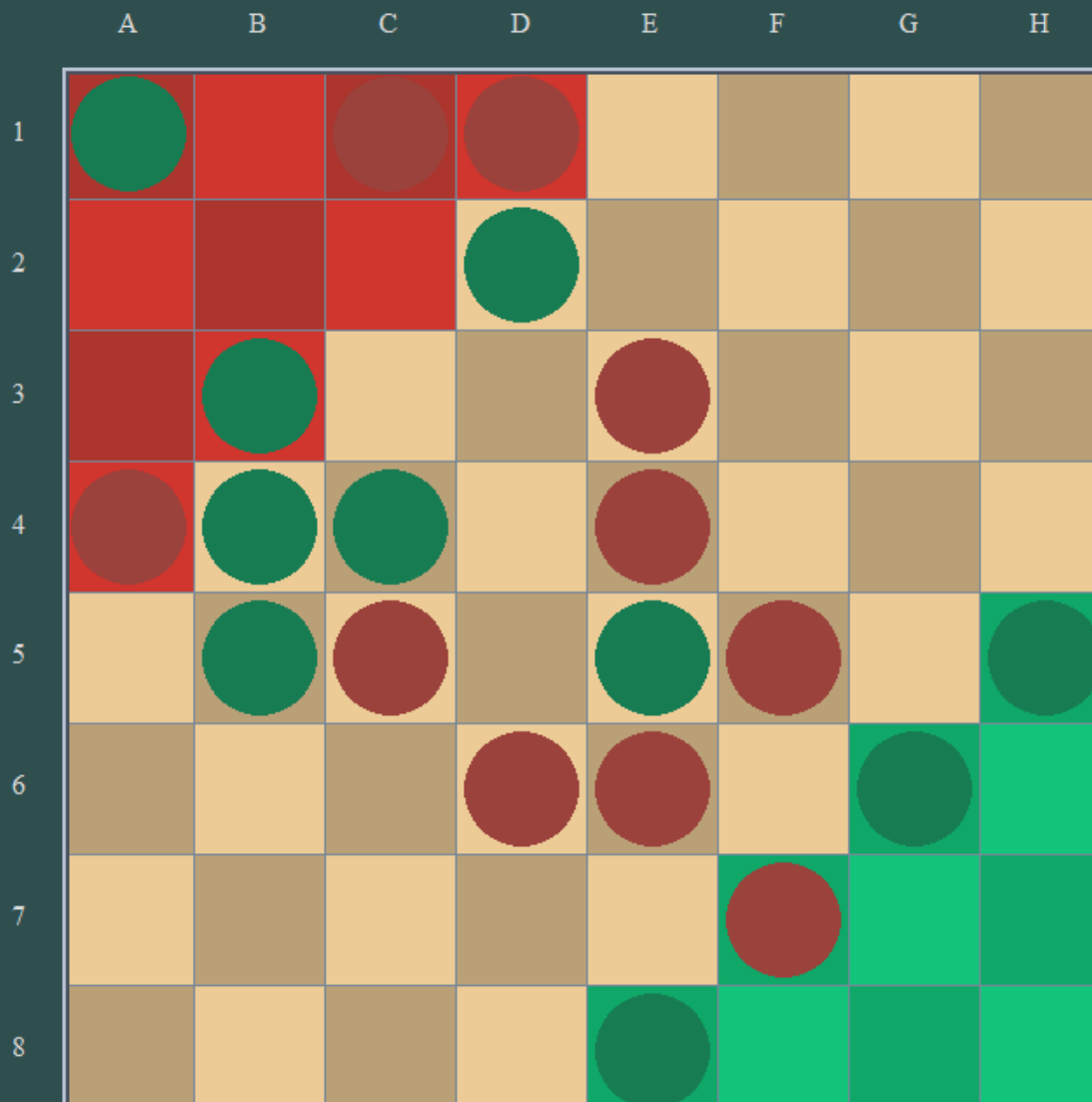
AI moved From (8, 6) To (4, 2), Time Taken: -48 seconds.



Player 1 Turn:

RED , Time left = 00:59

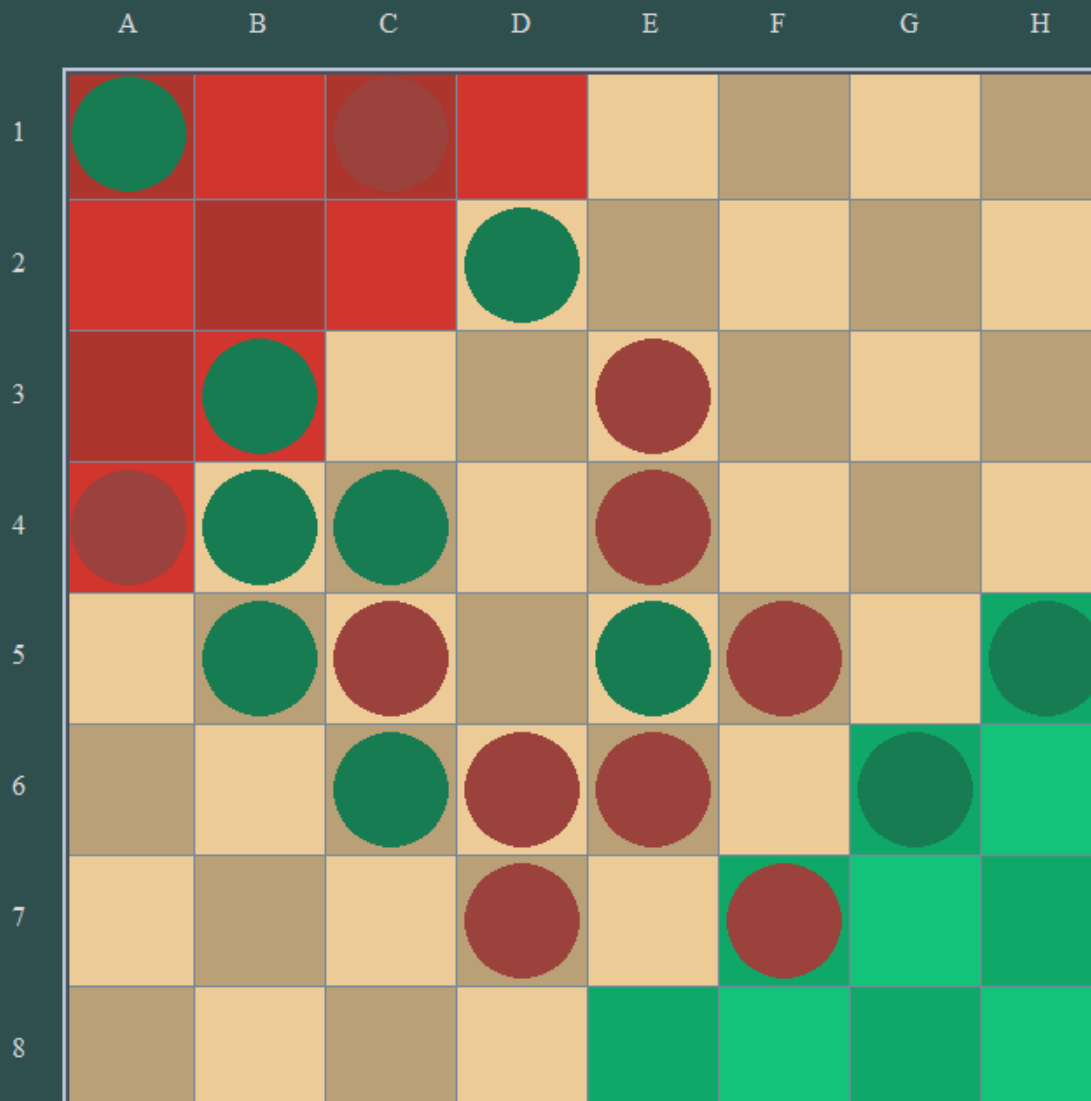
AI moved From (6, 7) To (2, 3), Time Taken: -48 seconds.



Player 1 Turn:

RED , Time left = 00:58

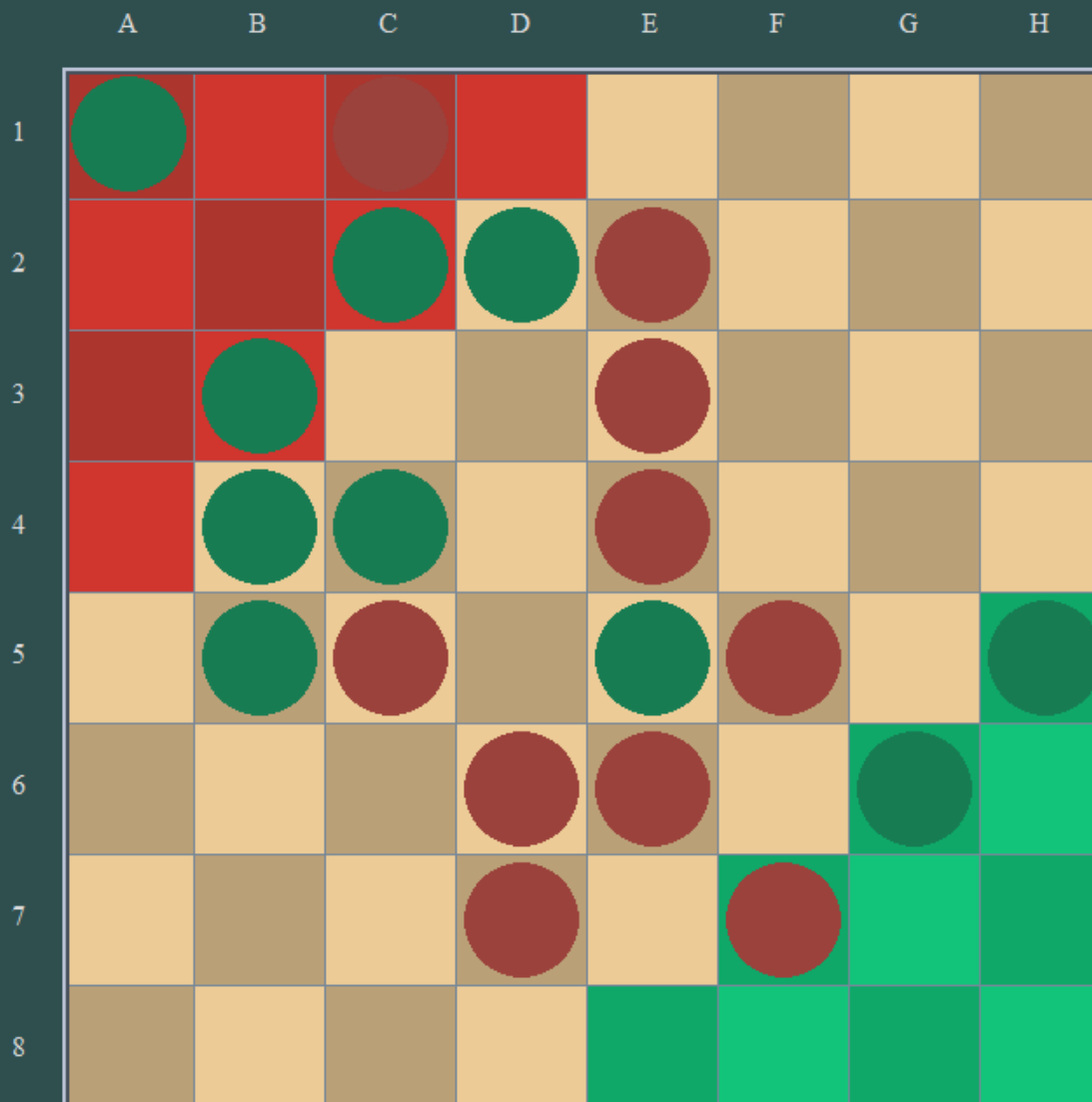
AI moved From (2, 2) To (1, 1), Time Taken: -48 seconds.



Player 1 Turn:

RED , Time left = 00:58

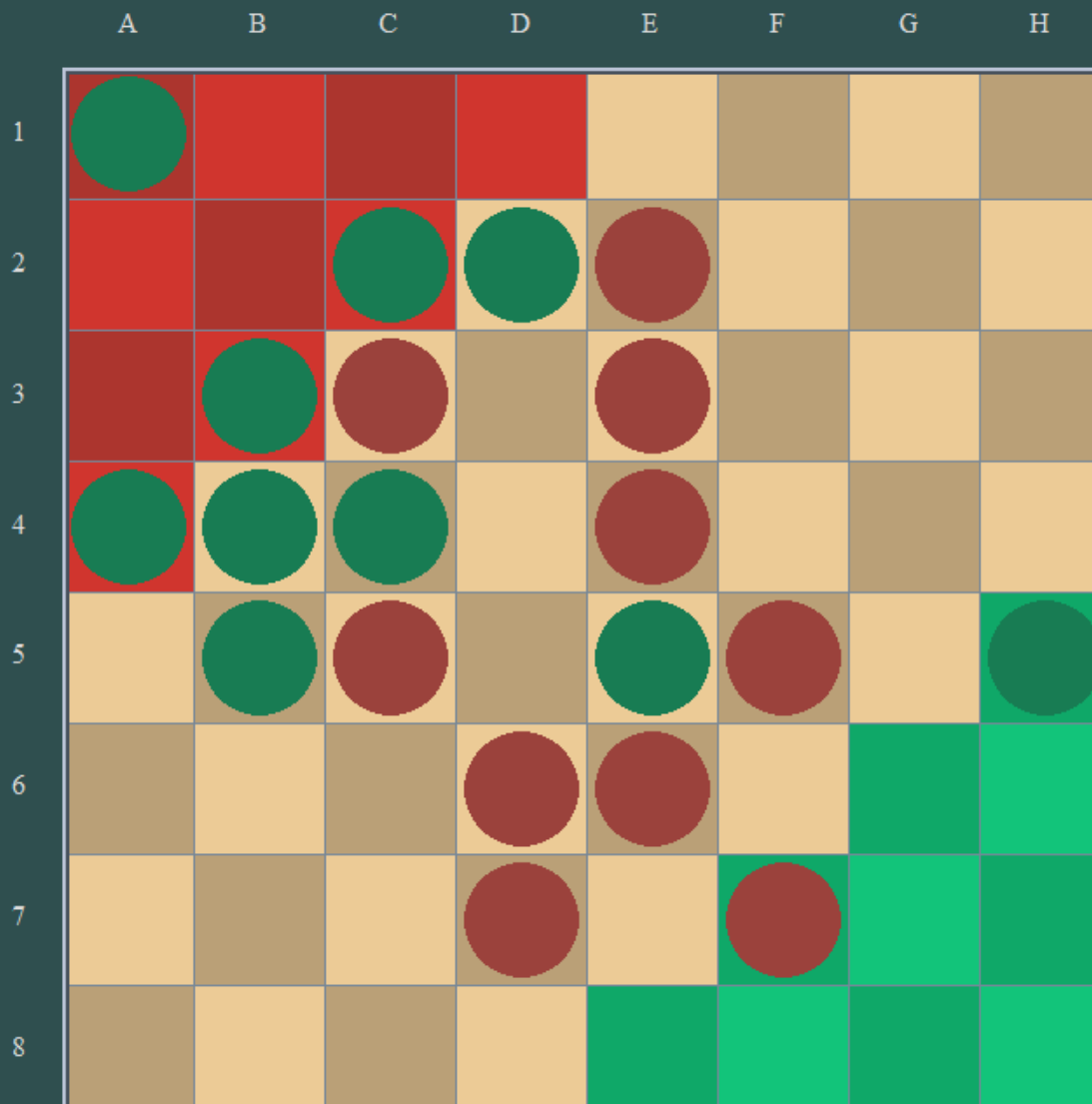
AI moved From (5, 8) To (3, 6), Time Taken: -48 seconds.



Player 1 Turn:

RED , Time left = 00:55

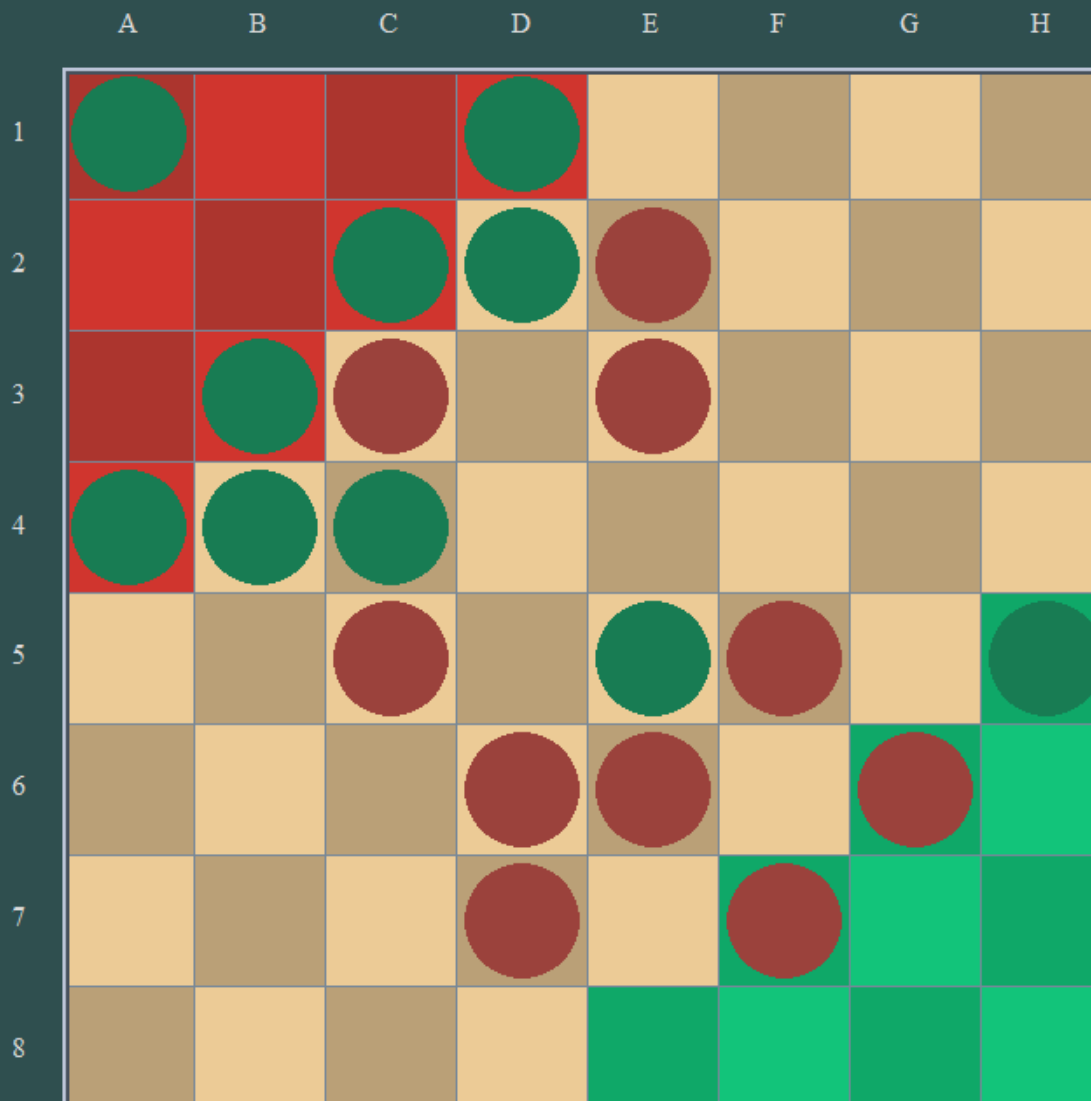
AI moved From (3, 6) To (3, 2), Time Taken: -48 seconds.



Player 1 Turn:

RED , Time left = 00:56

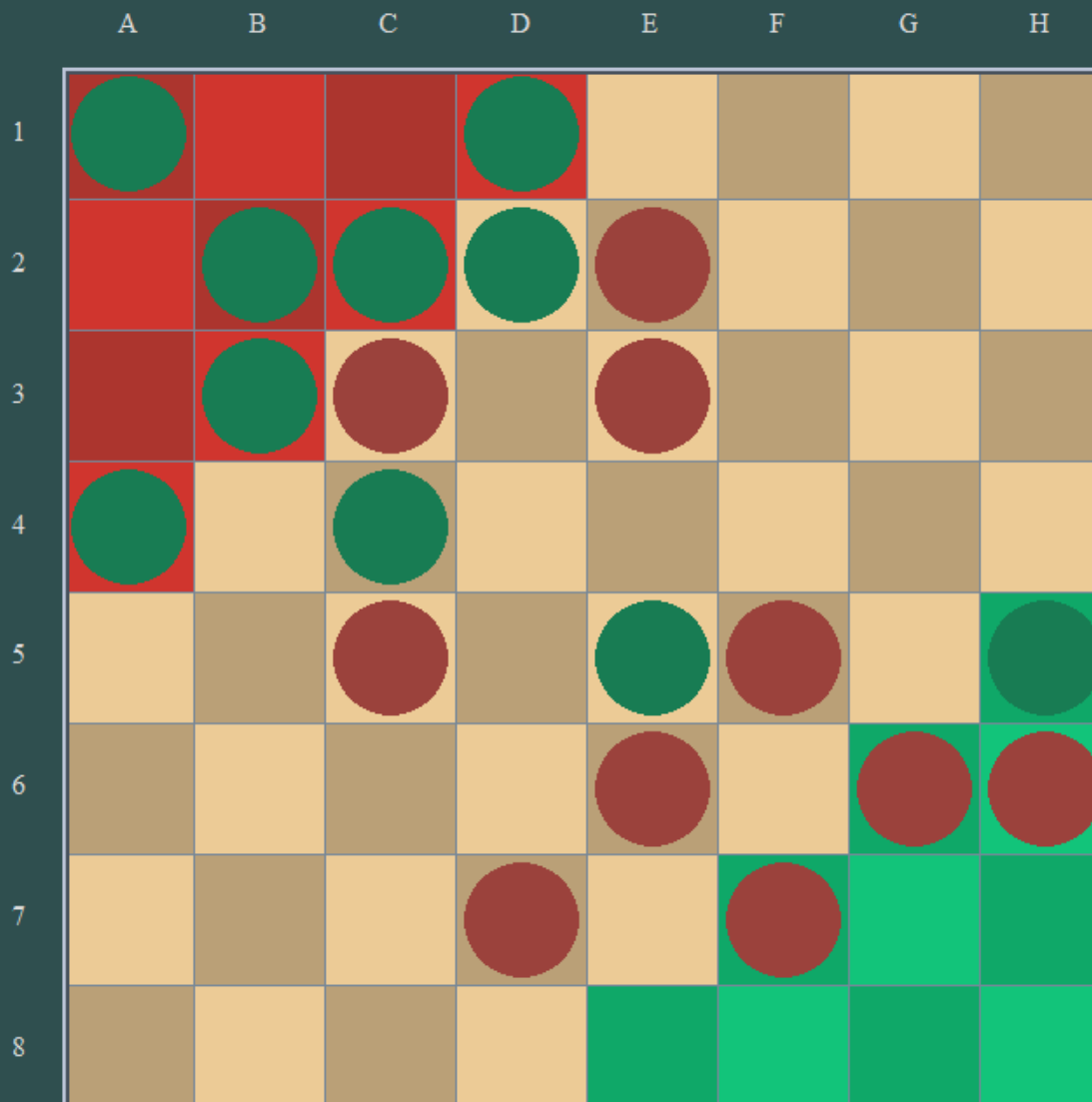
AI moved From (7, 6) To (1, 4), Time Taken: -48 seconds.



Player 1 Turn:

RED , Time left = 00:57

AI moved From (2, 5) To (4, 1), Time Taken: -48 seconds.



Player 1 Turn:

RED , Time left = 00:59

AI moved From (2, 4) To (2, 2), Time Taken: -48 seconds.

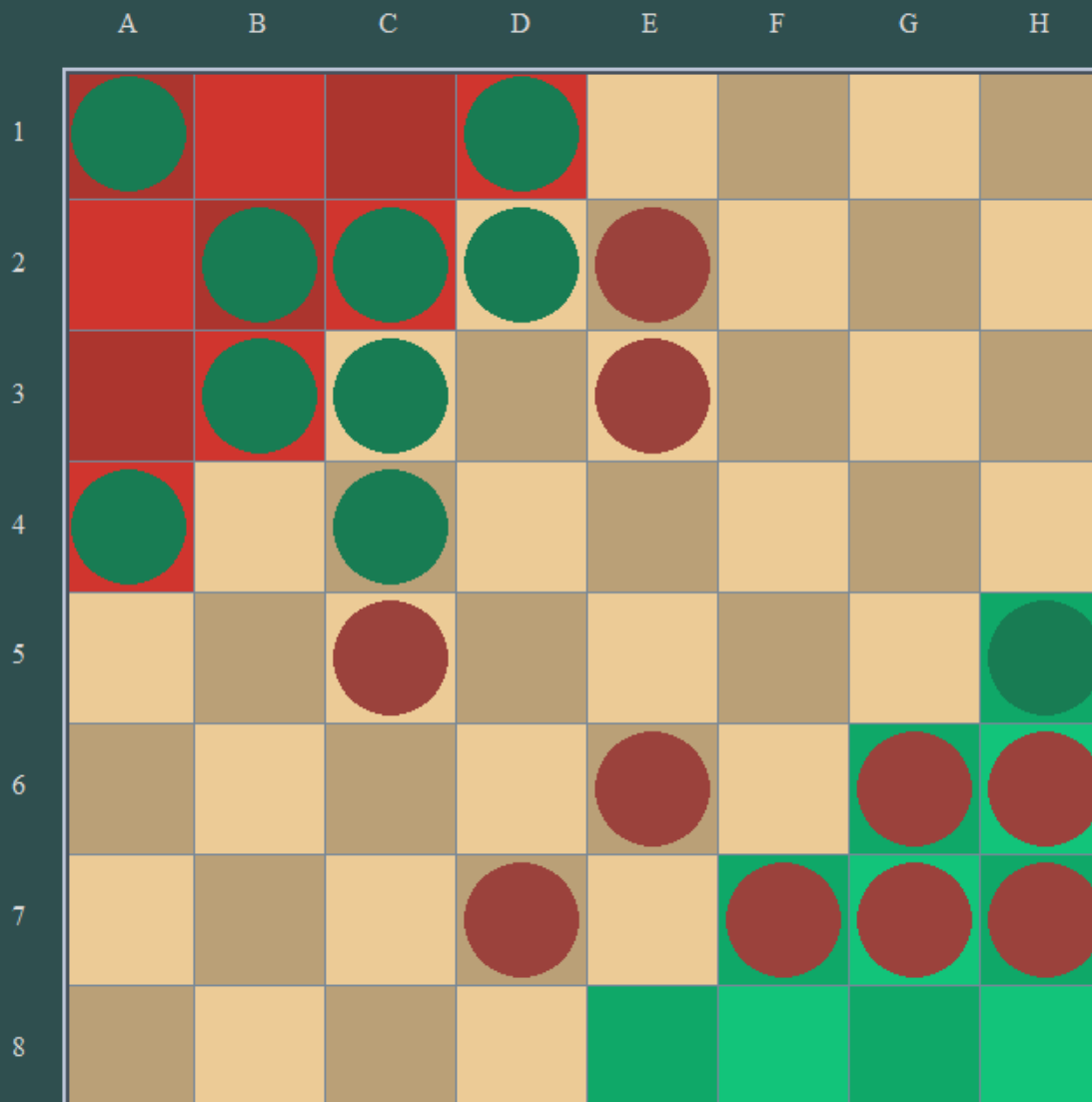
Halma A.I Game

	A	B	C	D	E	F	G	H
1								
2								
3								
4								
5								
6								
7								
8								

Player 1 Turn:

RED , Time left = 00:53

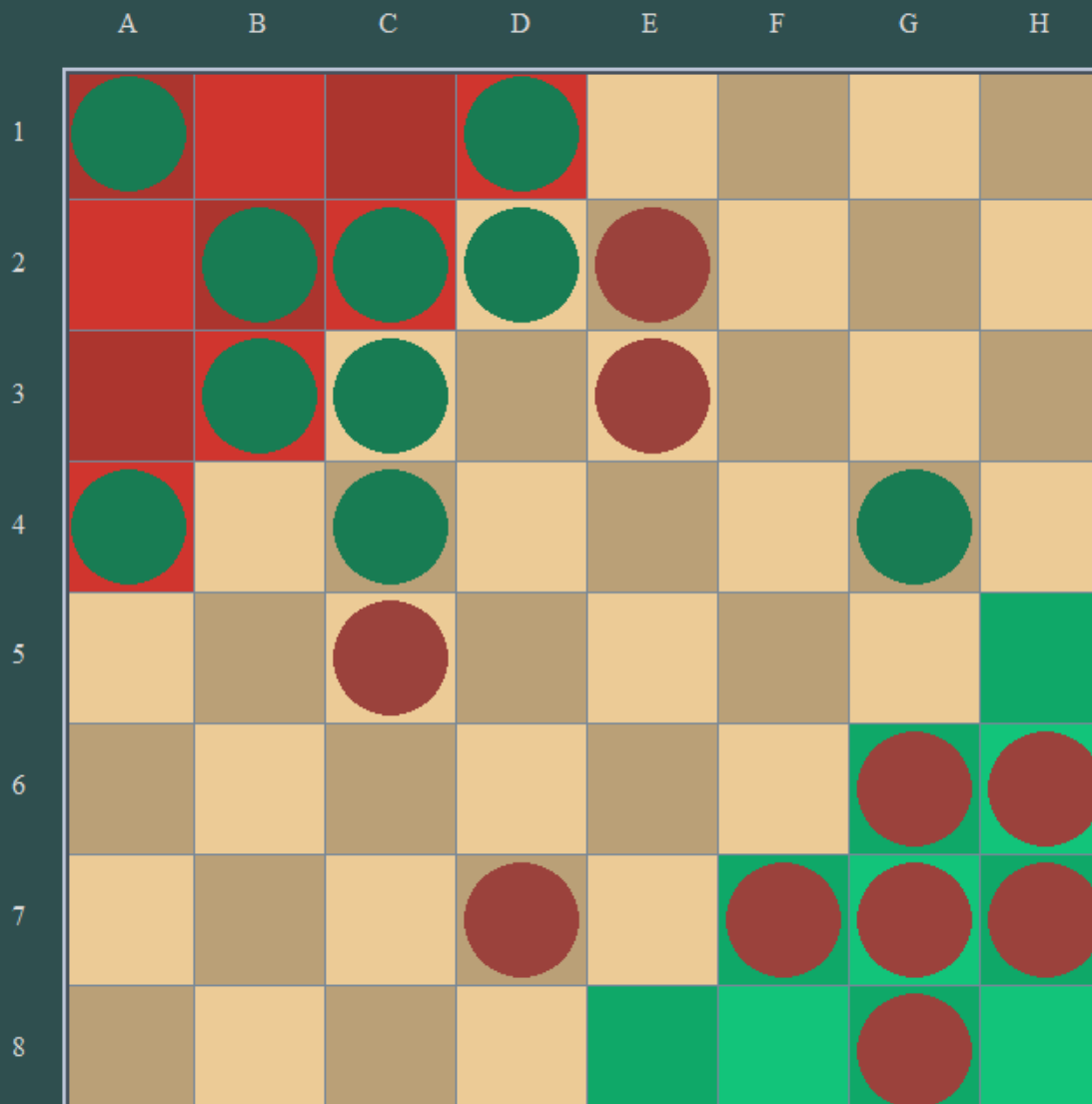
AI moved From (5, 5) To (4, 4), Time Taken: -48 seconds.



Player 1 Turn:

RED , Time left = 00:59

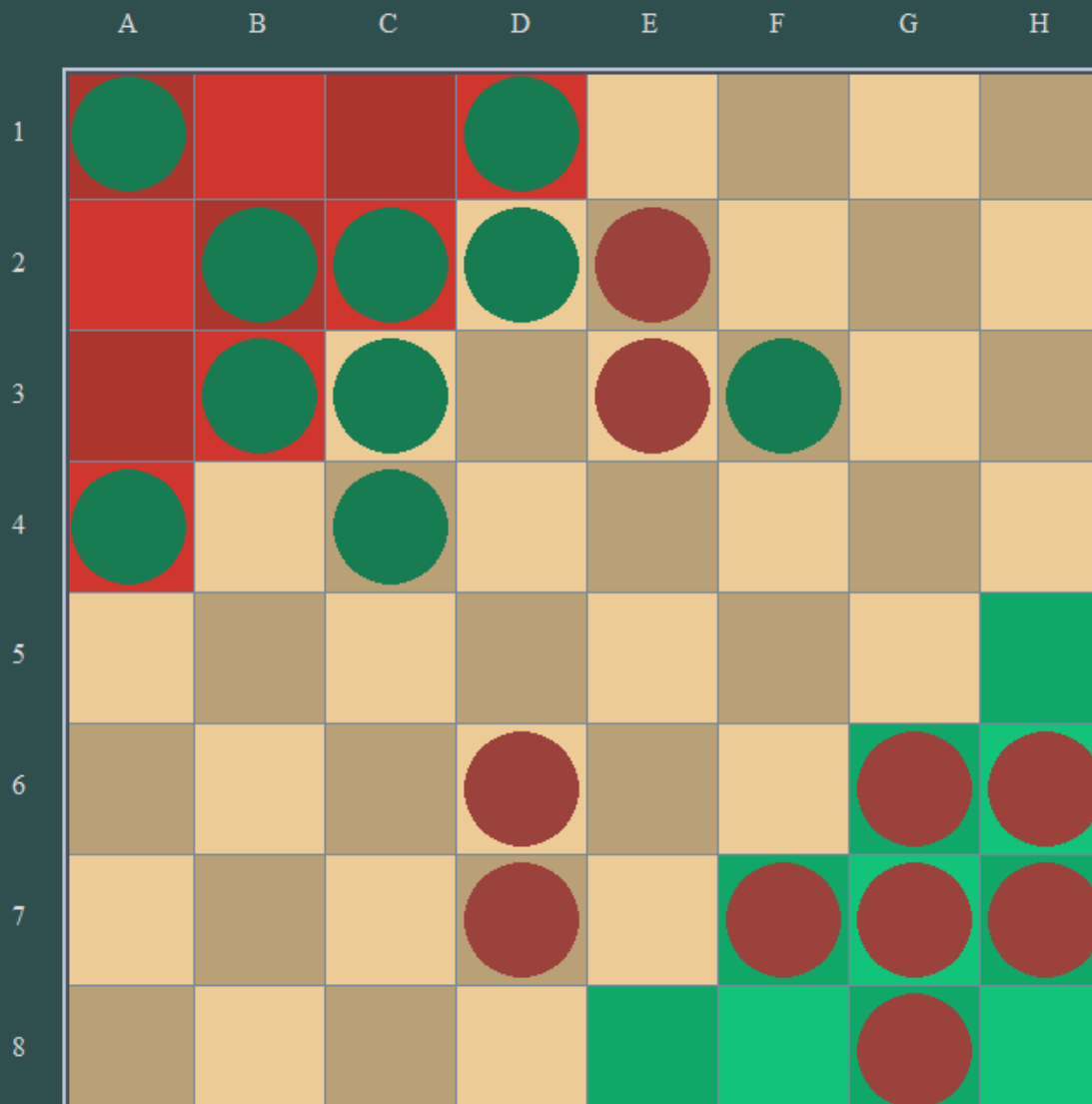
AI moved From (4, 4) To (3, 3), Time Taken: -48 seconds.



Player 1 Turn:

RED , Time left = 00:56

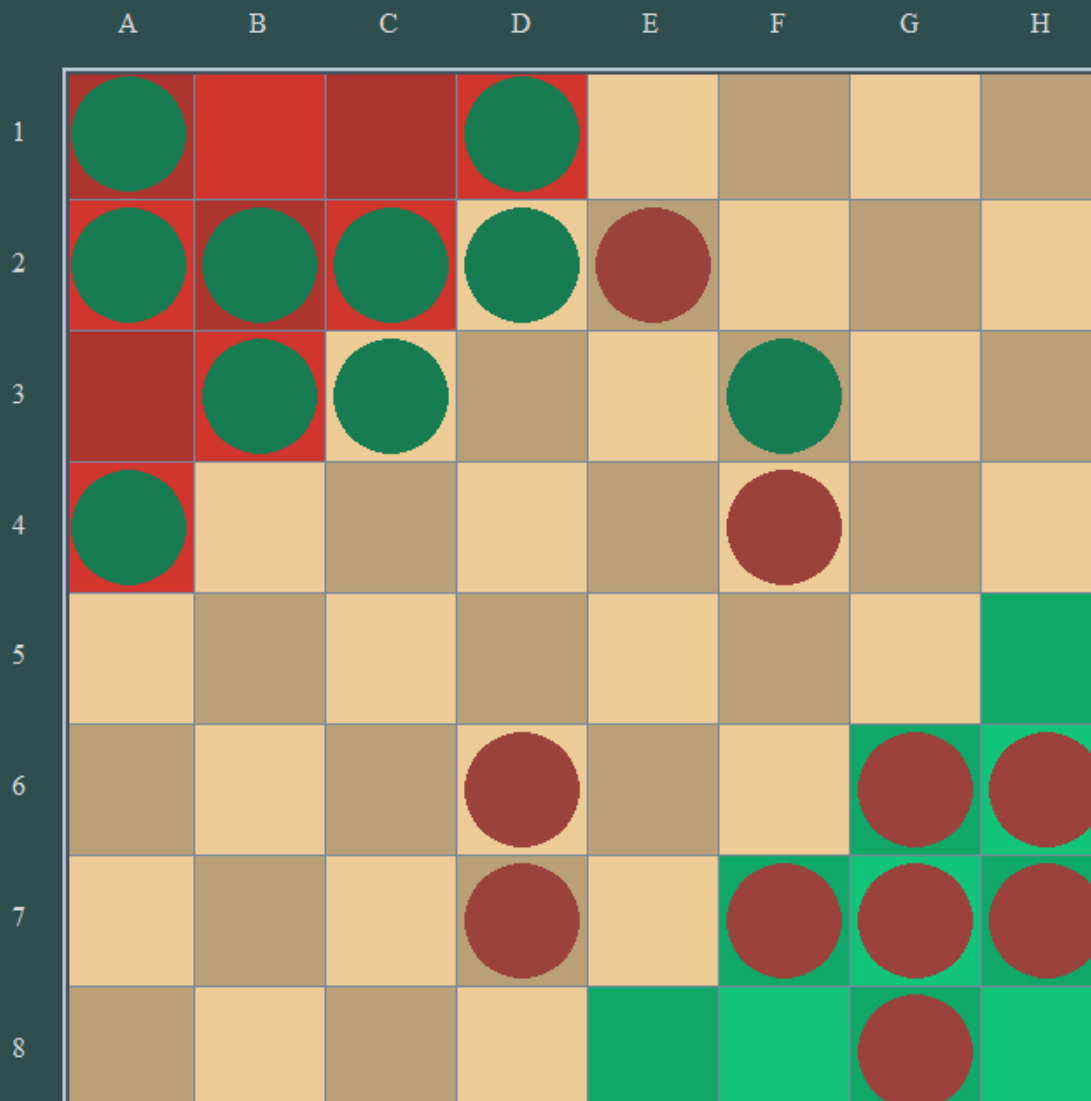
AI moved From (8, 5) To (7, 4), Time Taken: -48 seconds.



Player 1 Turn:

RED , Time left = 00:57







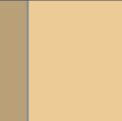













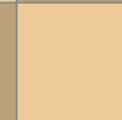






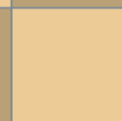






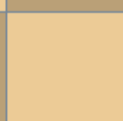
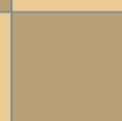









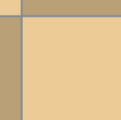




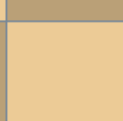







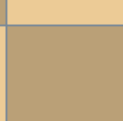

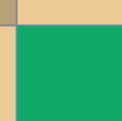



AI moved From (7, 4) To (6, 3), Time Taken: -48 seconds.



Player 1 Turn:

RED , Time left = 00:59



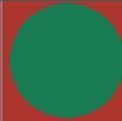



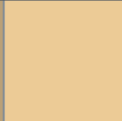













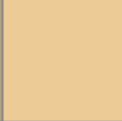

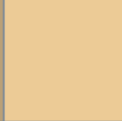








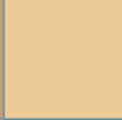
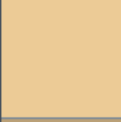











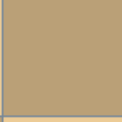



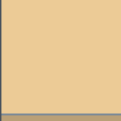














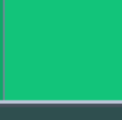
AI moved From (3, 4) To (1, 2), Time Taken: -48 seconds.

	A	B	C	D	E	F	G	H
1								
2								
3								
4								
5								
6								
7								
8								

Player 1 Turn:

RED , Time left = 00:55

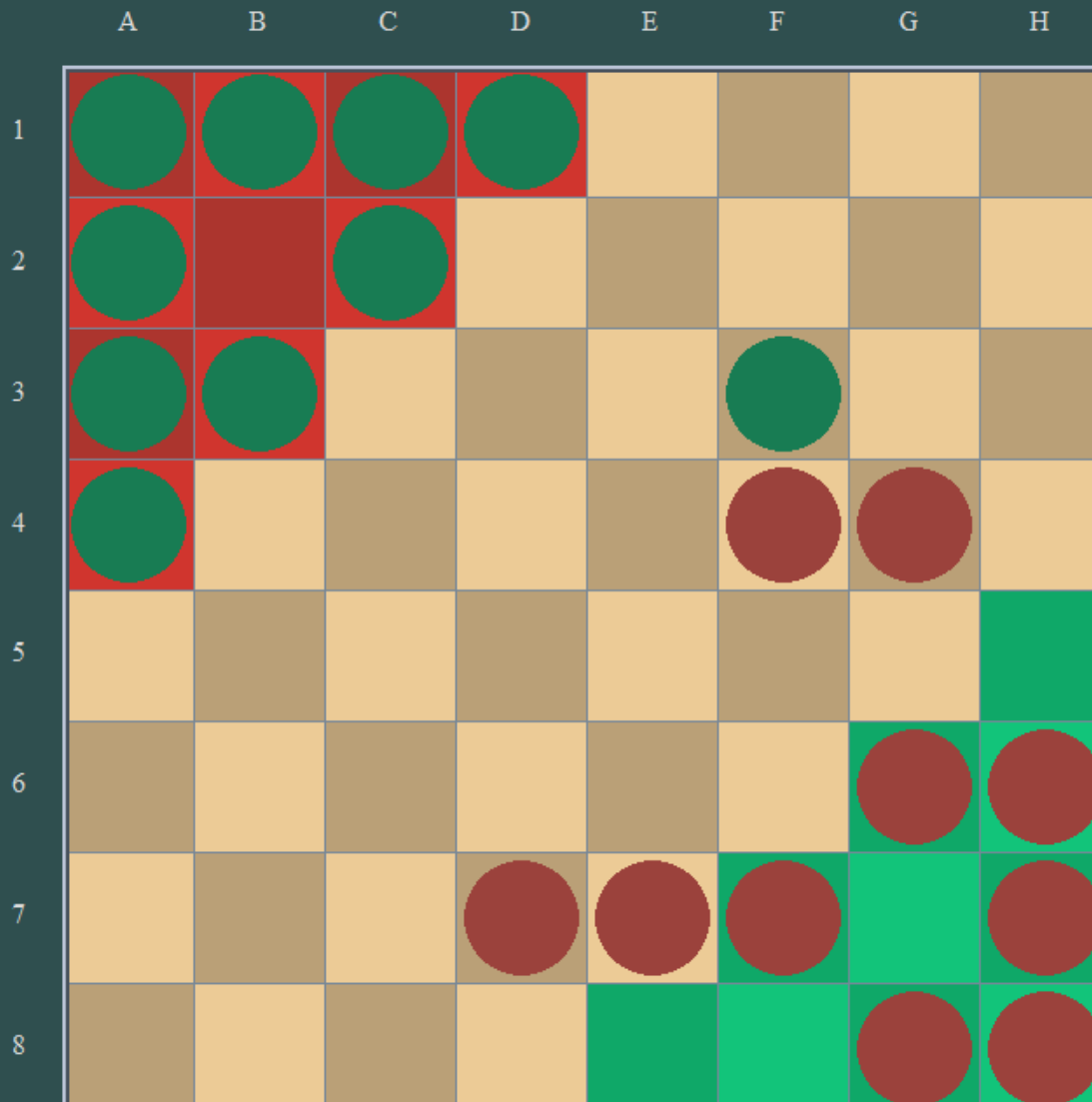
AI moved From (3, 3) To (1, 3), Time Taken: -48 seconds.

	A	B	C	D	E	F	G	H
1								
2								
3								
4								
5								
6								
7								
8								

Player 1 Turn:

RED , Time left = 00:58

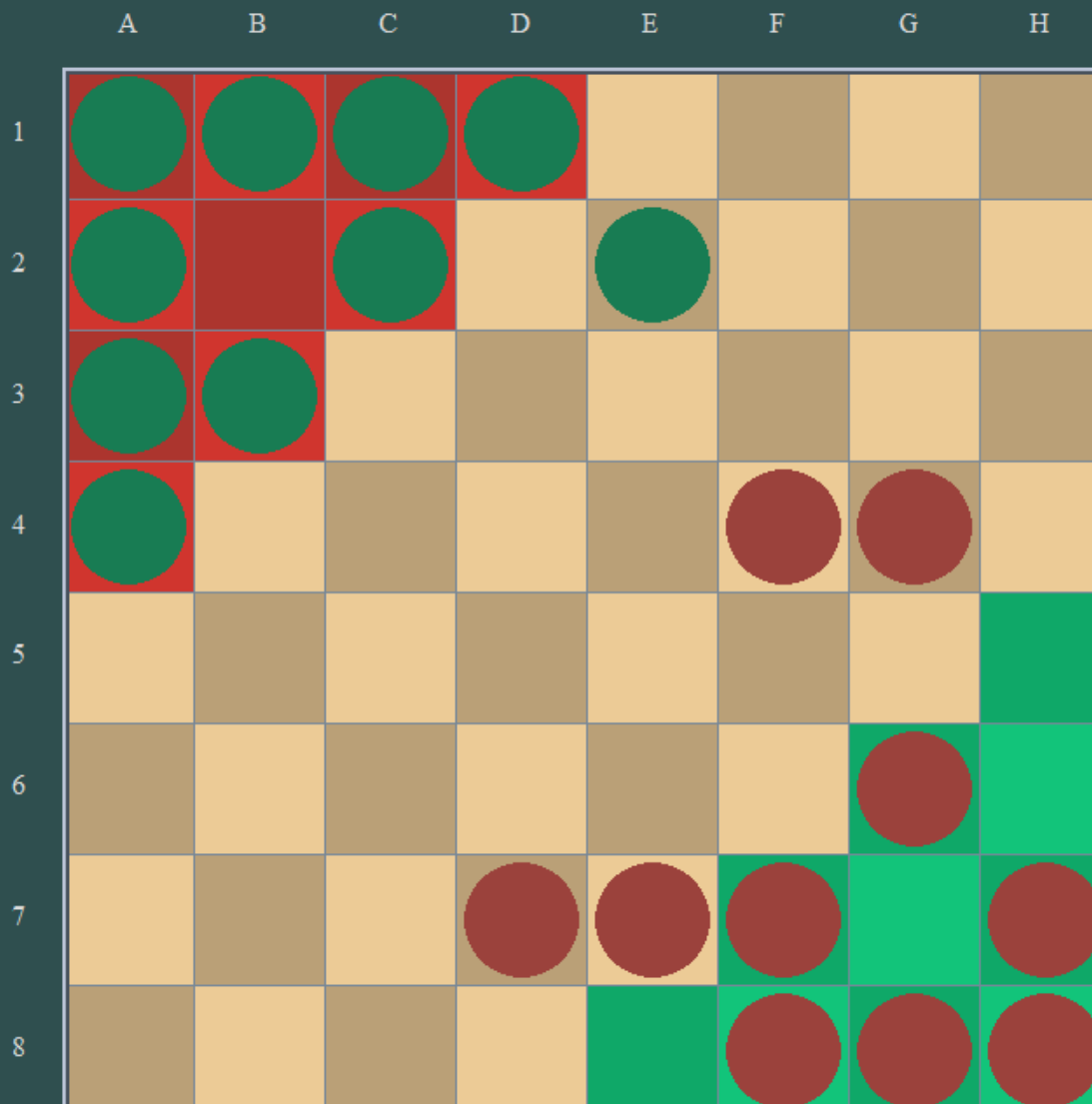
AI moved From (4, 2) To (3, 1), Time Taken: -48 seconds.



Player 1 Turn:

RED , Time left = 00:54

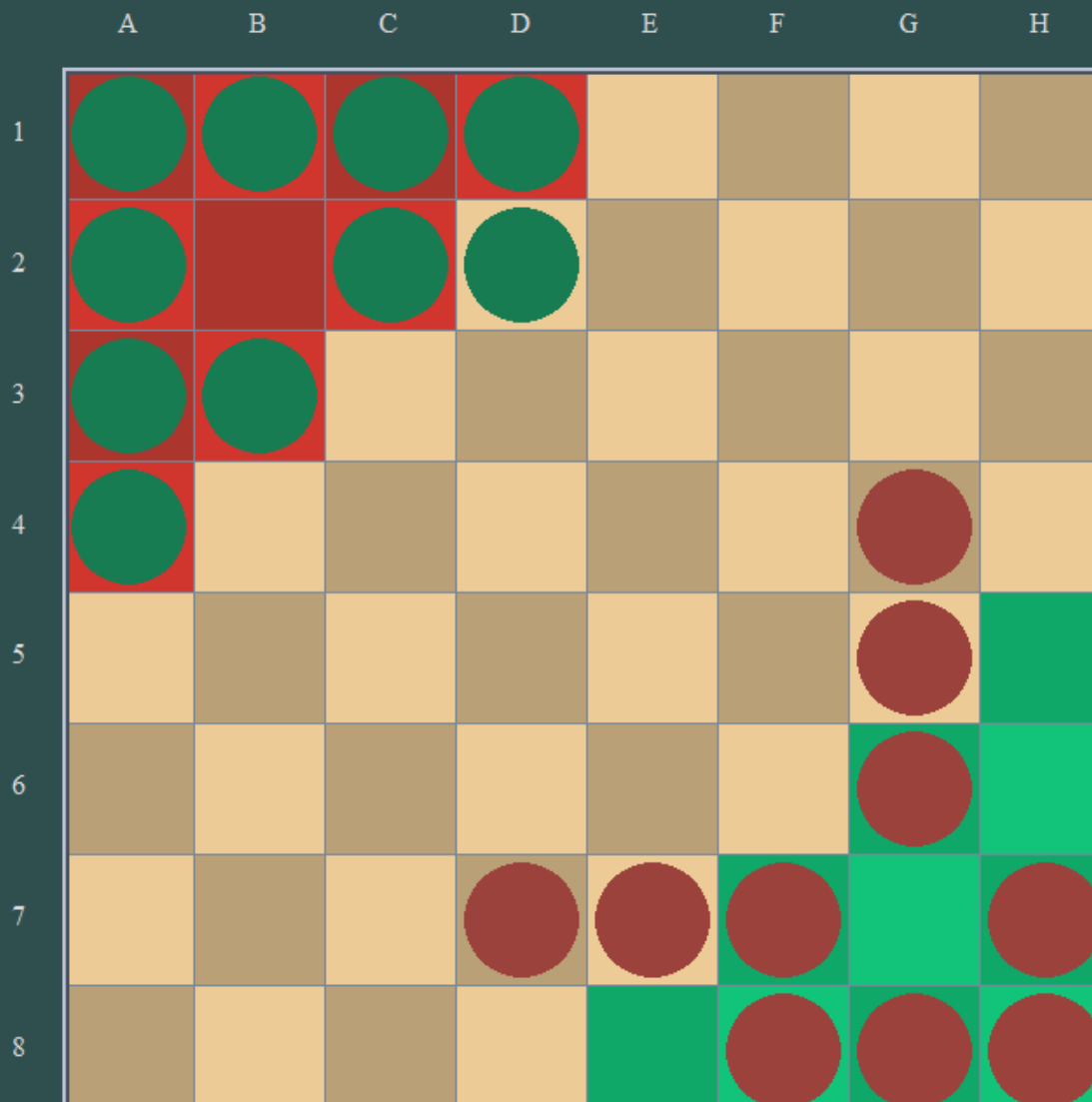
AI moved From (2, 2) To (2, 1), Time Taken: -48 seconds.



Player 1 Turn:

RED , Time left = 00:58

AI moved From (6, 3) To (5, 2), Time Taken: -48 seconds.



Player 1 Turn:

RED , Time left = 00:59

AI moved From (5, 2) To (4, 2), Time Taken: -48 seconds.

	A	B	C	D	E	F	G	H
1	Green	Green	Green	Green	Yellow	Brown	Yellow	Brown
2	Green	Green	Green	Yellow	Brown	Yellow	Brown	Yellow
3	Green	Green	Yellow	Brown	Yellow	Brown	Yellow	Brown
4	Green	Yellow	Brown	Yellow	Brown	Yellow	Red	Yellow
5	Yellow	Brown	Yellow	Brown	Yellow	Brown	Red	Green
6	Brown	Yellow	Brown	Yellow	Brown	Yellow	Red	Green
7	Yellow	Brown	Yellow	Brown	Red	Green	Green	Red
8	Brown	Yellow	Brown	Yellow	Red	Green	Green	Red

Player 2 Won The Game.

GREEN , Time left = 01:00

AI moved From (4, 2) To (2, 2), Time Taken: -48 seconds.

Source code.

Phase 2

charutils.py.py

```
1  import copy
2
3  # Characted Util Class which is used to create
4  #Objects of players when the game begins.
5
6  class CharacterUtil:
7      def __init__(self, color, boardSize):
8          # Setting each player with its resepective colors
9          self.color = color
10         self.starposchar(boardSize)
11         # Home and Goal Cordinates
12         # home cordinate is Goal cordinate for the other player
13         #and vice versa
14         self.homeCoord = []
15         self.goalCoord = []
16         self.move = False
17     # Functon to get the starting postion of the charcters
18     def starposchar(self, boardSize):
19         # Creating empty list varibale to track the home
20         # Goal and Tokens
21         self.charss = []
22         self.home = []
23         self.goal = []
24         # ASsigning Iteration based on the board size.
25         if boardSize == 8:
26             lenmax = 5
27         elif boardSize == 10:
28             lenmax = 6
29         else: # default is 16 x 16
30             lenmax = 7
31         # Iterating though our borad to set the home GOal
32
33         for i in range(1, lenmax):
34             for j in range(1, lenmax):
35                 # and the positon of the player. so that we can keep a track later.
36                 if (i + j <= lenmax and i < lenmax and j < lenmax):
37                     self.charss.append(Utils(boardSize - j + 1, boardSize - i + 1))
38                     self.home.append((boardSize - j + 1, boardSize - i + 1))
39                     self.goal.append((j, i))
40         # For debugging PUrpose
41         print(self.charss)
42         print(self.home)
43         print(self.goal)
44
45
46         if self.color == 'RED':
47             # The other Player also have similar Cordinates the
48             # only change is that
49             temp = copy.deepcopy(self.home)
50             self.home = copy.deepcopy(self.goal)
51             # The goal cordinate for player 1 is Home Corindate of player 2
52
53             self.goal = temp
54             # and the Home Cordinate of player 1 is the Goal cordinate of player 2.
55             for i in range(len(self.charss)):
56                 self.charss[i].x = self.home[i][0]
57                 self.charss[i].y = self.home[i][1]
```

```

57         self.charss[i].y = self.home[i][1]
58
59
60
61     # FUnction to Check that the game is FINised Or not
62     # if all the players reached oppositon Home
63     # then the player wins the game.
64     def all_char_opp(self):
65         status = True
66         for p in self.charss:
67             if (p.x, p.y) in self.goal:
68                 status = True
69             else:
70                 status = False
71                 break
72         return status
73
74     # Fucntion to Chekc the location of the chratcer
75     def charlocCheck(self,x,y):
76         # if the charcater is in X and Y cordinate it returns True
77         for i in range(len(self.charss)):
78             if(self.charss[i].x == x and self.charss[i].y== y ):
79                 return True
80                 break
81             else:
82                 # Else It return false
83                 pass
84         return False
85     # FUnction To check the Home Cordinate of the Chracter.
86     def charHomeCheck(self,x,y):
87         koor =(x, y)
88         # If Givenn Cordinate is in our H0me list that we alredy Trakced then it return true
89         if koor in self.home:
90             return True
91         else:
92             return False
93
94     #Function to Check the goal cordinate
95     def charGoalCheck(self,x,y):
96         koor =(x, y)
97         # If given Cord is Goal cord then true else false
98         if koor in self.goal:
99             return True
100         else:
101             return False
102     # FUnction to get the Charatcer
103     #Token from row and Column
104     def getChar(self, row, column):
105         i = 0
106         found = False
107         # If a chracter token is found withing our row and Columns
108
109         while i < len(self.charss) and not(found):
110             # it returns the chracters if it is avaiible

```

```

111         if (self.charss[i].x == row and self.charss[i].y == column):
112             character = self.charss[i]
113             return character
114             found = True
115         else:
116             i +=1
117
118     # A temporary movement function that is used by AI to move the Tokens
119     # temp so that the ai get the best possible moves.
120     def Char_Move_Temp(self, grid_from, grid_to):
121         (x, y) = grid_from
122         (x2, y2) = grid_to
123         # it takes the coordinate of tile to the goal tile and Move the token
124         # temporarily
125         for p in self.charss:
126             if p.x == x and p.y == y:
127                 p.x = x2
128                 p.y = y2
129                 self.charss = sorted(self.charss, key=lambda p: (p.x, p.y))
130                 break
131
132     # Function to move the character from one Grid Location
133     # TO the other grid location
134     def char_movement(self, gridfrom, gridto):
135         # get the current Location
136         (x, y) = gridfrom
137         (x2, y2) = gridto # get the destination
138         # Looping through our Tokens
139         for p in self.charss:
140             # if its x coordinates matches with the current coordinate
141             if p.x == x and p.y == y:
142                 # and if it is in Home Cord then we can make a movement
143                 if (x, y) in self.home and (x2, y2) not in self.home:
144                     p.char_gone = True
145                 # Similarly if it is goal then we cannot make a move
146                 elif (x, y) not in self.goal and (x2, y2) in self.goal:
147                     p.char_arr = True
148                 # Assigning GOal Coordinate
149                 p.x = x2
150                 p.y = y2
151                 self.charss = sorted(self.charss, key=lambda p: (p.x, p.y))
152                 self.move = True
153                 break
154
155
156     # Utils Class for More debugging Purpose to get each and ANY single variable.
157     class Utils:
158         def __init__(self, x, y):
159             #setting the Coordinates
160             self.setCoordinate(x, y)
161             #Print("CHAR INISITALIZE")
162             self.setChar_Gone(False)
163             self.setChar_arr(False)
164             # TO set the coordinates again
165         def setCoordinate(self,x, y):
166             self.x = x
167             self.y = y

```



```

167     self.y = y
168     # to set Char Dest
169     def setChar_Gone(self, char_gone):
170         self.char_gone = char_gone
171     # To set Char Currnet
172     def setChar_arr(self, char_arr):
173         self.char_arr = char_arr
174     # To get the X coordinates of Player
175     def getCoordinateX(self):
176         return (self.x)
177     # TO get Y coordinates
178     def getCoordinateY(self):
179         return (self.y)
180
181     def getCoordinate(self):
182         return (self.x, self.y)

```

Utilities.py

```

Utilities.py / Utilities / printCoordinate
1  #Utilities Class That helps in various stage to get the data
2  #related to our aGame
3  class Utilities:
4      # Initializing Constructor so that we can easily get the Color and Cordianates
5      # of our Player
6      def __init__(self, x, y, color="BLACK", character=0):
7          self.x = x
8          self.y = y
9          self.color = color
10         self.character = character
11         # setting the color of a player that he choosed.
12         def ColorSetup(self, color):
13             self.color = color
14         #setting the chracter tokens
15         def CharTokenSetup(self, character):
16             self.character = character
17     #Getting the coordinates of the player.
18     def printCoordinate(self):
19         print(str(self.x) + str(self.y) + self.color + str(self.character))

```

Game.py

```
game.py > ...
1  #importing other package
2  from Utilities import Utilities
3  import math
4  import time
5  #Our Game Class which handels and Genrate moves in the back
6  #It is also responsible to generate Jumps while making moves.
7  class Game:
8      #initializing our construter of the game.
9      def __init__(self, boardSize, timeLimit, p1, p2, human,bot,alphabeta):
10         #assigning all the parametres essential to play the game
11         self.boardSize = boardSize
12         self.timeLimit = timeLimit
13         self.player1 = p1
14         self.player2 = p2
15         self.alphabeta=alphabeta
16         #assigning therir respective color to the players
17         self.p1cop = p1 if p1.color == "GREEN" else p2
18         self.p2cop = p2 if p2.color == "RED" else p1
19         #assigning turn to a player
20         self.turn = 1
21         self.bot=bot
22         #getting the cordianes of our board
23         self.coordinate = [[Utilities(i, j) for i in range(self.boardSize)] for j in range(self.boardSize)]
24         self.deep = 2
25         #humans as both players are playing the game.
26         self.human = human
27         self.ai_move=None
28
29         #Assigning maximum iterations according to our board size.
30         if self.boardSize == 8:
31             lenmax = 4
32         elif self.boardSize == 10:
33             lenmax = 5
34         else:
35             lenmax = 6
36         #looping through all the iterations.
37         for i in range(lenmax):
38             for j in range(lenmax):
39                 if (i + j < lenmax and i < 6 and j < 6):
40                     #Making red player home cordinate red and
41                     self.coordinate[i][j].color = "RED"
42                     self.coordinate[i][j].character = 2
43                     #making green player home cordinates green
44                     #Appending all the home and goal cordinates of
45                     #player 1 and Player 2
46
47                     self.p2cop.homeCoord.append(self.coordinate[i][j])
48                     self.p1cop.goalCoord.append(self.coordinate[i][j])
49                     #similarly for Player 2 Assigning Color and Home goal cordinates
50                     self.coordinate[self.boardSize - 1 - i][self.boardSize - 1 - j].color = "GREEN"
51                     self.coordinate[self.boardSize - 1 - i][self.boardSize - 1 - j].character = 1
52                     self.p1cop.goalCoord.append(self.coordinate[self.boardSize - 1 - i][self.boardSize - 1 - j])
53                     self.p2cop.homeCoord.append(self.coordinate[self.boardSize - 1 - i][self.boardSize - 1 - j])
54
55
```

```

55
56 #function to track Board Size
57 def gamesiz(self):
58     return self.boardSize
59 #Function to Chehck that the cordinate is empty or not to make a move
60 def checkboxemp(self,x,y):
61     #if a box is empty then player can make a move there if eligible
62     #if empty it return True
63     if(self.player1.charlocCheck(x,y) or self.player2.charlocCheck(x,y)):
64         return False
65     #else It return false
66     else:
67         return True
68 # Fucntion to Check the HOme Coordinates of the player
69 def checkhomecord(self, player, x, y):
70     return(player.charHomeCheck(x,y))
71 #Function to check the goal coordinates of the player
72 # so that we can check that a player reached goal or not
73 def checkgoalcord(self, player, x, y):
74     return(player.charGoalCheck(x,y))
75 # Getting all the list of avaibalve positons to make a move.
76 def allemppositions(self, position, alone):
77     x, y = position
78     list_of_positions = []# to retrive all the locaitions from the bord.
79     # if player is in the center and no ther token of otehr player is in negibour
80     # then it can move one step to any direction
81     if (alone == 1):
82         list_of_positions.append((x+1, y))
83         list_of_positions.append((x+1, y+1))
84         list_of_positions.append((x, y+1))
85         list_of_positions.append((x-1, y+1))
86         list_of_positions.append((x-1, y))
87         list_of_positions.append((x-1, y-1))
88         list_of_positions.append((x, y-1))
89         list_of_positions.append((x+1, y-1))
90 #if other player lies in the neghibouruing tile then we need to
91 #calculate jumps all over the boards
92 #appending all the available positions to our positions list
93 else:
94     if (not(self.checkboxemp(x+1,y)) and (self.checkboxemp(x+2, y))):
95         list_of_positions.append((x+2, y))
96     if (not(self.checkboxemp(x-1,y)) and (self.checkboxemp(x-2, y))):
97         list_of_positions.append((x-2, y))
98     if (not(self.checkboxemp(x,y+1)) and (self.checkboxemp(x, y+2))):
99         list_of_positions.append((x, y+2))
100     if (not(self.checkboxemp(x,y-1)) and (self.checkboxemp(x, y-2))):
101         list_of_positions.append((x, y-2))
102     if (not(self.checkboxemp(x+1,y+1)) and (self.checkboxemp(x+2, y+2))):
103         list_of_positions.append((x+2, y+2))
104     if (not(self.checkboxemp(x+1,y-1)) and (self.checkboxemp(x+2, y-2))):
105         list_of_positions.append((x+2, y-2))
106     if (not(self.checkboxemp(x-1,y+1)) and (self.checkboxemp(x-2, y+2))):
107         list_of_positions.append((x-2, y+2))
108     if (not(self.checkboxemp(x-1,y-1)) and (self.checkboxemp(x-2, y-2))):
109         list_of_positions.append((x-2, y-2))
110

```

```

110
111     length = len(list_of_positions)
112     i = 0
113     #once we get all the position to make a move its time to get Valid Positons.
114     while (i < length):
115         (x, y) = list_of_positions[i]
116
117         if(x<1 or y<1 or x>self.boardSize or y>self.boardSize):
118             list_of_positions.remove(list_of_positions[i])
119             length -= 1
120
121         elif (alone == 1 and not(self.checkboxemp(x, y))):
122             list_of_positions.remove(list_of_positions[i])
123             length -= 1
124         else:
125             i += 1
126     #all valid positons are returned to the player to make a move
127
128
129
130     return list_of_positions
131 #Function to Check JUmP on the board based
132
133 def checkjump(self, position, jumps, last_position): # On the current Positon of a player
134     #jumps can be made is a player is in neghibouring tile and the next tile
135     # to the other player is Empty
136     lis_available_jump = self.allemppositions(position, 2)
137
138     try:
139         lis_available_jump.remove(last_position)
140     except:
141         pass
142
143     if (len(lis_available_jump) == 0):
144         return jumps
145     else:
146         # Getting all the valid tiles to make a jump
147         #from one plce to another
148
149         for i in range (len(lis_available_jump)):
150             # if a tile is not avaiable to jump it is removed.
151             if lis_available_jump[i] not in jumps:
152                 jumps.append(lis_available_jump[i])
153                 #with the help of Recursion we check eachjumps for tiles.
154                 self.checkjump(lis_available_jump[i], jumps, position)
155 # if a player is making a move then it is validated that player is making a correct move or not.
156 def validatemove(self, character):
157     if (self.player1.charlocCheck(character.x, character.y)):
158         player = self.player1
159         print("IF")
160     else:
161
162         player = self.player2
163         print(" ELse")
164

```

```

164
165     ##getting the current position of the Player
166     current_position = (character.x, character.y)
167     print("GET CURENT",current_position)
168
169
170     # Once we have all the available positons for the move. we
171     # can validte each move. and jumps
172     list_of_positions = self.allemppositions(current_position, 1)
173     all_jumps = self.allemppositions(current_position, 2)
174     # looping until we check all the availables
175     if (len(all_jumps) > 0):
176         #if a jump is available to move then it is appened to our list
177         for i in range (len(all_jumps)):
178             if (all_jumps[i] not in list_of_positions):
179                 list_of_positions.append(all_jumps[i])
180             jumps = []
181             # once all the jumps are retrived we can validate to the avaiable Jumps to make a move.
182             self.checkjump(all_jumps[i], jumps, current_position)
183             #looping untill we get all teh validated moves
184             if (len(jumps) > 0):
185                 for i in range (len(jumps)):
186                     if (jumps[i] not in list_of_positions):
187                         list_of_positions.append(jumps[i])
188
189     # same concept a above using List of Moves
190     length = len(list_of_positions)
191     print("Length",length)
192     i = 0
193
194     while (i < length):
195         (x, y) = list_of_positions[i]
196         print("X,y," ,list_of_positions[i])
197         print("PLYER WHILE",player)
198         #checking all the home and goal coordinates of a player if it is not then we removed that positon from
199         # the list of positos
200         if (character.char_arr and not(self.checkgoalcord(player, x, y))) or (character.char_gone and (self.checkhomecord(player, x ,y))):
201             list_of_positions.remove(list_of_positions[i])
202             length -= 1
203         else:
204             i += 1
205
206     # once we have alist we sorted it
207     list_of_positions = sorted(list_of_positions, key=lambda tup: (tup[0], tup[1]))
208     return list_of_positions
209
210 # fucntion in backend to move chracter from one player to another.
211 def char_movement(self, xpos, ypos):
212     #getting from cordiante
213
214     gridfrom = self.coordinate[xpos[0]-1][xpos[1]-1]
215     #getting the coordinate where we have to move the player
216     gridto = self.coordinate[ypos[0]-1][ypos[1]-1]
217     # Chekcing that the move is valid or not
218
219     if gridfrom.character == 0 or gridto.character != 0:

```

```

218     if gridfrom.character == 0 or gridto.character != 0:
219         print("Invalidddddddddddddd")
220         return
221     # only a valid move can be played
222     # once a move is validated
223     if gridfrom.character == 1:
224         self.p1cop.char_movement((gridfrom.x+1, gridfrom.y+1), (gridto.x+1, gridto.y+1))
225     # the move is played by our backend board which is then displayed on our scree
226     #using the interface that we created.
227     elif gridfrom.character == 2:
228         self.p2cop.char_movement((gridfrom.x+1, gridfrom.y+1), (gridto.x+1, gridto.y+1))
229     else:
230         # IF the move is invalid then it prints invlaid in the terminal
231         print("Invalidddddddddddddd")
232         return
233     #Copying our moves to other variables so that we can move
234     gridto.character = gridfrom.character
235     gridfrom.character = 0
236     # using the distance formual to calculate the distance
237
238     def distance_calculation(self,x1, y1, x2, y2):
239         # we built this function so that we can calculate the distance between current and the goal coordinates
240         # form all the locationsor grid that bot is currentl
241         dist=math.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)
242         # once the distance is calculates it is returned
243         return dist
244 # This function is used to calculate the goal distance by calling the
245 # distance function that we built earlier
246 # it gets the current cordiante of the player and
247 #calls the distance calculate function to calculate the distance between the points
248 #i.e X1 x2 y1 y2
249 def calculategoal(self, player):
250     hold=-1
251     temp_val = 0 # To hold the temp vlaue for the distance
252
253     for x in range(self.boardSize):
254         for y in range(self.boardSize):
255             # Looping though our board
256             cor = self.coordinate[y][x] # getting the current coordinates using the x and y
257             # that is A and 1 for example in our case.
258
259             if (player.color == "GREEN" and cor.character == 1):
260                 # if AI that is playing assiged the color Green.
261
262                 dist_to_goal = [self.distance_calculation(cor.x, cor.y, x - 1, y - 1) for (x, y) in player.goal if
263                 self.coordinate[y - 1][x - 1].character != 1]
264                 # we calualte the distance from the cur to goal location
265                 print("DIST TO GOAL",dist_to_goal)
266                 # if we got some distance from cord to goal then we choose the one with the maximum distance
267                 temp_val += max(dist_to_goal) if len(dist_to_goal) else -50
268             # if ai is assigned the color red
269             elif (player.color == "RED" and cor.character == 2):
270                 print(player.color)
271                 # Simialrly using our distance funciton to calculate the distance between cord and goal
272                 dist_to_goal = [self.distance_calculation(cor.x, cor.y, x - 1, y - 1) for (x, y) in player.goal if
273                 self.coordinate[y - 1][x - 1].character != 2]

```

```

273         self.coordinate[y - 1][x - 1].character != 2]
274         # Similarly as above we calculate the distance to the goal and choose the maximum value
275         temp_val += max(dist_to_goal) if len(dist_to_goal) else -50
276         temp_val *= hold
277         return temp_val
278
279         # This function takes all the valid moves
280         # and return the coordinates of every valid moves
281     def CharTurnCoord(self, all_valids):
282         moves = []
283         for i in all_valids:
284             # looping through all the valid moves and returning their
285             # coordinates
286             val_turn = self.coordinate[i[1] - 1][i[0] - 1]
287             moves.append(val_turn)
288
289         return moves
290
291     # This is our main AI algorithm that will play the game
292     # it starts with the depth that we provide currently it is
293     # 2 so that it can explore the moves
294     def minimax(self, deep, playermin, playermax, MAX=True, alp=float("-inf"), bet=float("inf")):
295         # basis
296         print("deep", deep)
297         # if the depth is 0 we explored then we return the moves of max player
298         if deep == 0:
299             return self.calculategoal(playermax), None
300
301         # with the help of recursion we calculate the best move to the goal node
302         cur_best = None
303         # we start with getting all the valid moves so that we have a understanding where we can move
304         if MAX:
305             cur_best_val = float("-inf")
306             All_Valid_moves = self.Player_Char_moves(playermax)
307
308         else:
309             cur_best_val = float("inf")
310             All_Valid_moves = self.Player_Char_moves(playermin)
311
312         # Once we have all the valid moves that we can play
313         # we loop through all the moves
314         # and using a temporary movement we calculate all the moves
315
316         for turn in All_Valid_moves:
317             for grid_to in turn["grid_to"]:
318                 # so that in the end AI has a best move to play
319                 self.Char_Move_Temp((turn["grid_from"].y + 1, turn["grid_from"].x + 1), (grid_to.y + 1, grid_to.x + 1))
320             # AI use a Temporary character movement function
321             # which is just a copy of
322             # our char_movement function
323             # Using the recursion technique we explore the game again by subtracting the depth by
324             # -1
325             # that is depth=-1
326             temp_val, unk = self.minimax(deep - 1, playermin, playermax, not MAX, alp, bet,)
327             # Temporary moved our Tokens
328             self.Char_Move_Temp((grid_to.y + 1, grid_to.x + 1), (turn["grid_from"].y + 1, turn["grid_from"].x + 1))
329             # This is the main part

```

```

326 # This is the main part
327 # that is used to get the best move
328 if MAX and temp_val > cur_best_val:
329     #if our new best move is better then the old best then it is assined as new best move
330     cur_best_val = temp_val
331     # So that we always have a current best move out of all
332     cur_best = ((turn["grid_from"].y + 1, turn["grid_from"].x + 1), (grid_to.y + 1, grid_to.x + 1))
333
334     alp = max(alp, temp_val)
335     # we get the maximum of that so that we can use it in alpha beta prunning later
336     #if alpha beta is enabled
337
338 if not MAX and temp_val < cur_best_val:
339     # if we didnt find the best move in the Max depth
340     # then we explore more and get the best move as possinble
341     cur_best_val = temp_val
342     cur_best = ((turn["grid_from"].y + 1, turn["grid_from"].x + 1), (grid_to.y + 1, grid_to.x + 1))
343     # calculating the beta to later use in alpha beta prunning
344     bet = min(bet, temp_val)
345
346 # alpha beta prunning
347 # this is only a optimization for our minimax AI Algortithm
348 if self.alphabeta=="ON":
349     #user can turn it on and off when user stars the game
350     if alp >=bet:
351         print("AlphaBetaaaaaaaaaaaaaaaaa")
352         return cur_best_val, cur_best
353
354 #returning our current best move that we got using the minimax recursion
355 return cur_best_val, cur_best
356
357 # this funciton is used to get the current move and all the possible moves that a token can be
358 # moved to played
359 def Player_Char_moves(self, player):
360     moves = [] # All possible moves
361     # everytime thsis function is called it will get the current tile locationa nd
362     # calculate all the possible moves then minimax get the best move
363     for p in player.charss:
364         curr_tile = self.coordinate[p.y - 1][p.x - 1]
365         turn = {
366             "grid_from": curr_tile,
367             "grid_to": self.CharTurnCord(self.validatemove(p))
368         }
369         # Returning all the valid moves form the Current tile
370         moves.append(turn)
371     return moves
372
373 # This is our temporay token movement function
374 # which is a copy of our char movemnt function
375 def Char_Move_Temp(self, xpos, ypos):
376     # we use this fucntion to calculate the temp move while searchin the best move for
377     # our AI
378     grid_from = self.coordinate[xpos[0] - 1][xpos[1] - 1]
379     grid_to = self.coordinate[ypos[0] - 1][ypos[1] - 1]
380     # IT will work similardy by getting the coordinats of the tiles
381     # and whoever AI is assignesd it makes a temp move.
382     if grid_from.character == 1:
383         self.plcop.Char_Move_Temp((grid_from.x + 1, grid_from.y + 1), (grid_to.x + 1, grid_to.y + 1))

```



```

382         self.plcop.Char_Move_Temp((grid_from.x + 1, grid_from.y + 1), (grid_to.x + 1, grid_to.y + 1))
383     elif grid_from.character == 2:
384         self.p2cop.Char_Move_Temp((grid_from.x + 1, grid_from.y + 1), (grid_to.x + 1, grid_to.y + 1))
385     else:
386         print("INVALID")
387         return
388     grid_to.character = grid_from.character
389     grid_from.character = 0
390 # This functions Handels all the Fucntionality of the AI Player
391 #This function calls our minimax algo to start the search fr the best posiible move.
392 def AiPlay(self):
393     playermax = self.player2
394     playermin = self.player1
395     # Onces we get the best move in return our AI plays the move in the given time.
396     unk, turn = self.minimax(self.deep, playermin, playermax)
397     # if our AI is unable to find any move then no token is moved.
398     if turn == None:
399         print("No MOove")
400     else:
401         # else we get our cordinates of the tile we have to play our move.
402         (x1, y1) = turn[0]
403         (x2, y2) = turn[1]
404         # once we have our moves AI use the char movement function to make a move and the turn is
405         # assigned to other plyaer once AI completed his move.
406         self.char_movement((x1, y1), (x2, y2))
407         self.ai_move=[(y1,x1),(y2,x2)]
408         # self.winn.move = True
409         print(f"MOVED FROM {(y1, x1)} TO {(y2, x2)}")
410
411
412
413
414
415
416
417

```

:- Main.py

```
Run Terminal Help Main.py - halma_part_2 - Visual Studio Code
Main.py 4 Utilities.py
Main.py > HalmaGame > play
1 # Import Nessecary Libraries
2 from charutils_py import CharacterUtil
3 from game import Game
4 import tkinter as tk
5 import threading
6 import time
7
8 # initialized a global variable
9 ch = 0
10
11 # main Game class HalmaGame
12
13
14 class HalmaGame:
15     # Initialize Cosntrutor so that player can be assigned colors and their respective goal.
16     def __init__(self, boardsize, timelimit, player1, player2, AI, alphabeta):
17         # assignning player from our characterUtils Class which is on Other script
18         self.player1 = CharacterUtil(player1, boardsize)
19         # assignning player from our characterUtils Class which is on Other script
20         self.player2 = CharacterUtil(player2, boardsize)
21         self.human = False
22         self.alphabeta = alphabeta
23         if AI == "AI":
24             self.AI = "M"
25             AI = "M"
26         else:
27             self.AI = None
28             AI = None
29         # Initizlize Game object which is the main Class that Handels overall game moves and spawns.
30         self.game = Game(boardsize, timelimit, self.player1,
31                             self.player2, self.human, AI, alphabeta)
32         # Turn so that each player can play a move simulutnously
33         self.turn = 1
34         # Initilizing Time limit so that player has to play move within timelimit
35         self.timelimit = self.game.timelimit
36         # Initializing Our Game mode Which is GUI based
37         self.mode = "GUI"
38         self.play()
39         # play function runs a loop until the game is not finished .. each player gets
40         # one chance to play one move
41         # if a user forget to play his move
42         # within a time limit then he cannot make his move until his next turn
43
44     def play(self):
45         # initializing our GUI which is created using tkinter
46         self.winn = Halmaguimode(self.game)
47         # initizling first move so that a player can play his move
48         self.winn.move = True
49
50         self.turn = 2 if self.player1.color == "RED" else 1
51         self.winn.move = False if self.player1.color == "RED" else True
52         print(self.turn)
53         # setting the Status bar which shows Player turn.
54         # game alwways starts with Green Player as mentioned in the doc.
55         self.winn.status.config(
```

```

54 # game always starts with Green Player as mentioned in the doc.
55 self.winn.status.config(
56     text=f"Green Player Start Game...", bg="#187c53")
57 self.winn.info.config(
58     text=f"Player1: {self.player1.color} , Player2: {self.player2.color}, Alpha Beta Purning: {self.alphabeta}")
59 print("here")
60 self.winn.t = self.timelimit
61 self.winn.backup = self.timelimit
62 # creating a threaded object so that we can also monitor the time limit.
63 p1 = threading.Thread(target=self.winn.timeren)
64 p1.start()
65 checko = 0
66 # looping until a player wons the game
67 while (self.endgame() == True):
68     # endgame checks that a player has won or not.
69     # a player won if his all the spawns are in opponent home.
70     # which is goal in player case
71     if (self.turn == 2 and not (self.winn.move)):
72         print("PLAYER@@@@")
73         # playing Move and setting the status bar for player
74         if self.AI != None:
75             self.winn.timer3.config(text=f" AI is Thinking ")
76             self.winn.status.config(
77                 text="Player 2 Turn: ", bg=self.player2.color)
78
79             self.winn.makechar()
80 # WE modified our last code that
81 # is human vs human to Human vs AI in this section.
82 # if Human wants to play with AI
83 if self.AI != None:
84     self.winn.timer3.config(text=f" AI is Thinking ")
85     # we used a thread oboject so that AI can calucate its best move beside while
86     # we run a timer in other thread to keep track of the Timer of the AI
87     # if AI failed to move within the time range then AI turn will be skiped
88     game = threading.Thread(target=self.game.AiPlay)
89
90     self.winn.timer3.config(text=f" AI is Thinking ")
91     game.start()
92     # Once ai player his turn then turn will be assigned to other player
93     while self.player2.move == False:
94         times = self.winn.t
95         # self.winn.timer3.config(text=f" {self.player2.color} , Time left = {self.winn.t}")
96         # self.winn.timer3.config(text=f" {self.player2.color} , Time left = {self.winn.t}")
97         print("YO", self.winn.t)
98
99         self.winn.timer3.config(text=f" AI is Thinking ")
100
101         if p1.isAlive():
102             # print("ALVE")
103             self.winn.timer3.config(
104                 text=f" {self.player2.color} , Time left = {self.winn.t} ")
105
106             # print("yes")
107             # pass
108         else:
109             self.winn.t = 0

```

```

106         #                                     print("yes")
107         #                                     pass
108     else:
109         self.winn.t = 0
110         time.sleep(1)
111         self.winn.t = self.timelimit
112         p1.join()
113         self.game.selected_tuple = None
114         for i in range(self.winn.board_size):
115             for j in range(self.winn.board_size):
116                 self.winn.canvas.itemconfigure(
117                     self.winn.tiles[i, j], outline="black", width=0)
118         self.winn.move = True
119         self.turn = 1
120
121         p1 = threading.Thread(target=self.winn.timeren)
122         p1.start()
123         self.winn.info.config(
124             text=f"AI moved From {self.game.ai_move[0]} To {self.game.ai_move[1]}, Time Taken: {(10-self.winn.t)+2} seconds.")
125         self.winn.t = 0
126         time.sleep(1)
127         self.winn.t = self.timelimit
128         p1.join()
129         self.winn.move = True
130         self.turn = 1
131
132         p1 = threading.Thread(target=self.winn.timeren)
133         p1.start()
134         self.winn.move = True
135         self.player2.move = False
136         print("BACLLLLKKK")
137     # Activating the timer so that player knows how much
138     # time is remaining to play a move.
139     if p1.is_alive():
140         self.winn.timer3.config(
141             text=f" {self.player2.color} , Time left = {self.winn.curtime}")
142         #                                     print("yes")
143         #                                     pass
144     else:
145         # if a player unable to perform a move in given
146         # specific time then
147         # player lost his turn to play move. He/she needs to wait until next turn.
148         self.winn.t = 0
149         time.sleep(1)
150         self.winn.t = self.timelimit
151         p1.join()
152         self.game.selected_tuple = None
153         # clearing all the moves if player not played any.
154         for i in range(self.winn.board_size):
155             for j in range(self.winn.board_size):
156                 self.winn.gamewin.itemconfigure(
157                     self.winn.tiles[i, j], outline="black", width=0)
158         self.winn.move = True
159         # assigning turn to next player.
160         self.turn = 1

```

```

Main.py > HalmaGame > play
50 self.turn = 1
51 # activating timer for next player
52
53 p1 = threading.Thread(target=self.winn.timeren)
54 p1.start()
55
56 # self.timer()
57 # self.winn.makechar()
58
59 elif (self.turn == 1 and self.winn.move == True):
60     # setting the status for next player
61     self.winn.status.config(
62         text="Player 1 Turn: ", bg=self.player1.color)
63
64     self.winn.makechar()
65     # if player makes his move then turn will be assigned to other player
66     if p1.isAlive():
67         self.winn.timer3.config(
68             text=f" {self.player1.color} , Time left = {self.winn.curtime} ")
69         # print("yes")
70         #
71         #
72         # pass
73         # if player cannot make his move. his chance lost.
74     else:
75         self.winn.t = 0
76         time.sleep(1)
77         self.winn.t = self.timelimit
78         p1.join()
79         self.game.selected_tuple = None
80         # clearing all the highlighted board moves.
81         for i in range(self.winn.board_size):
82             for j in range(self.winn.board_size):
83                 self.winn.gamewin.itemconfigure(
84                     self.winn.tiles[i, j], outline="black", width=0)
85         self.winn.move = False
86
87         p1 = threading.Thread(target=self.winn.timeren)
88         p1.start()
89
90         # print("else2")
91         # t1 = threading.Thread(target=self.winn.timer())
92
93         # self.winn.move = False
94         self.turn = 2 if self.turn == 1 else 1
95         self.game.turn = 2 if self.game.turn == 1 else 1
96
97         # op=0
98         # while(True) :
99         #     op+=1
100         #     self.winn.status.config(text=f"Player {self.endgame()} Won The Game.")
101         #     time.sleep(1)
102         #     if op==5:
103         #         break
104         self.winn.t = 0
105         self.winn.makechar()
106         p1.join()

```

```

216     pls = self.endgame()
217     if pls == 2:
218         colr = self.player2.color
219     else:
220         colr = self.player1.color
221     self.winn.status.config(
222         text=f"Player {self.endgame()} Won The Game.", bg=colr)
223     self.winn.update()
224     time.sleep(5)
225     print(f"Player {self.endgame()} Won The Game.")
226 # Fucntion to Check that game has a winner or not..
227
228
229 def endgame(self):
230
231     # this function called everytime in loop to check that a player won or not
232     # after every single move made by player
233     if (self.player1.all_char_opp() and not (self.player2.all_char_opp())):
234         return 1
235     # if player reach its goal that is home for other player then he/she wins
236     elif (self.player2.all_char_opp() and not (self.player1.all_char_opp())):
237         return 2
238     # otherwise game continues until we get a winner
239     else:
240         return True # 0 for not terminate
241
242 # Gui Class which is made using tkinter
243 # this class handels all the functionality of GUI to show the moves
244 # Show players, status, timer and evrything else.
245
246
247 class HalmaguiMode(tk.Tk):
248     def __init__(self, board, *args, **kwargs):
249         # initialize the parent class of tkinter
250         tk.Tk.__init__(self, *args, **kwargs)
251         # setting it to true so that our window can be resized
252         self.resizable(True, True)
253
254         self.configure(bg='#2F4F4F') # setting Background color
255         # initializing Our game Objects which helps in calculating game.
256         self.game = board
257         self.time = 0
258         self.board_size = board.gamesiz()
259         self.t = 20
260         self.backup = 0
261
262         # Assigning X and Y values to our grid board so that we can
263         # calculate each move by its Coordinate
264         # that is X,Y in our case it is 1,A for Eg.
265         for i in range(self.board_size):
266             roww = tk.Label(self, text=i + 1, font='Times',
267                             bg='#2F4F4F', fg='DCDCDC')
268             roww.grid(row=i + 1, column=0)
269
270             coll = tk.Label(self, text=chr(i + 97).upper(),

```

```

Main.py > HalmaGame > play
68         roww.grid(row=i + 1, column=0)
69
70         coll = tk.Label(self, text=chr(i + 97).upper(),
71                          font='Times', bg='#2F4F4F', fg='#DCDCDC')
72         coll.grid(row=0, column=i + 1)
73     # Making Status bar to print the current status of that that is happening
74     self.status = tk.Label(self, height=1, text="Welcome to Halma Game.....Green Player Start Game", width=50,
75                            relief="raised", font="Times", bd=3, bg="#001414",
76                            fg="#DCDCDC") # Customizing the status bar with some additional parameters
77     self.timer3 = tk.Label(self, height=1, text="TIMER...", width=50, relief="raised", font="Times", bd=3,
78                            bg="#001414",
79                            fg="#DCDCDC")
80     self.Info = tk.Label(self, height=1, text="TIMER...", width=50, relief="raised", font="Times", bd=3,
81                          bg="#001414",
82                          fg="#DCDCDC")
83     # Making our Game timer so that each player has same time to play a move
84
85     #         status.grid(row=0,column=i+1)
86     # initializing our Tiles for the Game.
87
88     self.tiles = {}
89
90     # Making a Blank Canvas of size 590*590
91     # with some additional Style Parameters
92     self.gamewin = tk.Canvas(self, width=590, height=590, bd=4,
93                              relief="ridge", bg="#778899", highlightthickness=0)
94     # using Grid insted of pack because we cannot use pack with grid.
95     self.gamewin.grid(
96         row=1, column=1, columnspan=self.board_size, rowspan=self.board_size)
97     # similarly setting status bar using grid method
98     self.status.grid(columnspan=self.board_size +
99                     2, rowspan=self.board_size)
100    # similarly setting Timer bar using grid method
101    self.timer3.grid(columnspan=self.board_size +
102                    2, rowspan=self.board_size)
103    self.Info.grid(columnspan=self.board_size + 2, rowspan=self.board_size)
104    self.title('Halma A.I Game') # main title of the window
105    # some additional configuration
106    self.columnconfigure(0, minsize=50)
107    self.rowconfigure(0, minsize=50)
108    self.columnconfigure(self.board_size + 1, minsize=50)
109    self.rowconfigure(self.board_size + 1, minsize=50)
110    self.gamewin.bind("<Configure>", self.makegrid)
111    self.game.selected_tuple = None
112    # our Timer which is called in each player move. so that each player
113    # gets equal amount of time to make a move.
114
115    def timeren(self):
116        while self.t != 0:
117            print("TIMEEEEEEEEE", self.t)
118            # using divmod to ge the time in min and secs
119            mins, secs = divmod(self.t, 60)
120            self.curtime = '{:02d}:{:02d}'.format(mins, secs)
121            # subtracting our decremented by 1 after a sec..
122            time.sleep(1)
123            #         print(self.t)

```

```

Main.py > HalmaGame > play
320     self.curtime = '{:02d}:{:02d}'.format(mins, secs)
321     # subtracting our decremented by 1 after a sec..
322     time.sleep(1)
323     #         print(self.ti)
324     if self.t == 0:
325         break
326     self.t -= 1
327     print("HEER BREAKDE")
328
329     #         print(self.curtime)
330     #         if self.t==0:
331     #             self.t=60
332     #         self.status.config(text="Timeleft")
333     # Function used to make grid in our window canvas
334
335     def makegrid(self, event=None):
336         self.gamewin.delete("checks")
337         heightcanva = 600
338         marginnsiz = 1
339         # Calculating each box size of our game as our game has 3 boards 8,10,16
340         boxx = int(heightcanva / self.board_size)
341         # looping thoug our board row and cos so that we can create a interface
342         # that is grided so that we can easily make moves.
343         for col in range(self.board_size):
344             for row in range(self.board_size):
345                 # calculcating all the cordiates
346                 x1 = col * boxx + marginnsiz / 2
347                 y1 = row * boxx + marginnsiz / 2
348                 x2 = (col + 1) * boxx - marginnsiz / 2
349                 y2 = (row + 1) * boxx - marginnsiz / 2
350
351                 if (self.board_size == 8):
352                     player1 = 4
353                     player2 = 10
354                 elif (self.board_size == 10):
355                     player1 = 5
356                     player2 = 13
357                 else:
358                     player1 = 6
359                     player2 = 24
360                 # assiging colors to each player Home with their respective colors
361                 if ((row + col) < player1):
362                     if ((row + col) % 2 == 0):
363                         color = '#AC352E'
364                     else:
365                         color = '#D0352E'
366
367                 # Red color always on the top left of the game screen
368                 elif ((row + col) > player2):
369                     if ((row + col) % 2 == 0):
370                         color = '#12C47A'
371                 # green color always on the bottom left of the game screen
372                 else:
373                     color = '#0FA868'
374                 # all other tiles are maked as playzone of the agame with a tan collor
375                 else:

```



```

Main.py > HalmaGame > play
370         color = '#12C47A'
371         # green color always on the bottom left of the game screen
372         else:
373             color = '#0FA868'
374         # all other tiles are made as playzone of the agame with a tan collor
375         else:
376             if ((row + col) % 2 == 0):
377                 color = '#ECCB96'
378             else:
379                 color = '#BAA077'
380         # creating rectangle Checked Grids.
381         checks = self.gamewin.create_rectangle(
382             x1, y1, x2, y2, tags="checks", width=0, fill=color)
383         self.tiles[col, row] = checks
384         # binding our checkers with our onpress function which is activated whenever we click to make move.
385         self.gamewin.tag_bind(
386             checks, "<1>", lambda event, row=row, col=col: self.onpress(row + 1, col + 1))
387
388     self.makechar()
389     # Fucntion used to make Charcter players that is red player and Green player
390     # we used Oval shape to assign a shape to a player of theri respective color
391
392     def makechar(self):
393
394         canvas_width = 600
395         heightcanva = 600
396         marginnsiz = 10
397         # similarly calucated box size as above
398         boxx = int(heightcanva / self.board_size)
399         # Initializing Our Tokens of the game. for both Players
400         self.p1char = self.game.player1.charss
401         self.p2char = self.game.player2.charss
402         # delete previous tokens canvas
403         self.gamewin.delete('character')
404         c = 0
405
406         for i in [range(len(self.p1char)), range(len(self.p2char))]:
407             for i in i:
408                 if c == 0:
409                     col = self.p1char[i].x - 1
410                     row = self.p1char[i].y - 1
411
412                 else:
413                     col = self.p2char[i].x - 1
414                     row = self.p2char[i].y - 1
415                 # calculation all the cordinates so that we can create
416                 # a oval shape for each player with their respective colors.
417
418                 x1 = col * boxx + marginnsiz / 2
419                 y1 = row * boxx + marginnsiz / 2
420                 # Using Simple Mathematics
421                 x2 = (col + 1) * boxx - marginnsiz / 2
422                 y2 = (row + 1) * boxx - marginnsiz / 2
423                 # Creating Shapes for charcted with their respecitive colors
424                 if c == 0:
425                     if (self.game.player1.colour == "GREEN"):

```

[illegible]

```

Main.py > HalmaGame > play
# whenever he select one of his token
473 if (self.game.player1.charlocCheck(column, row)):
474     character = self.game.player1.getChar(column, row)
475 elif (self.game.player2.charlocCheck(column, row)):
476     character = self.game.player2.getChar(column, row)
477     # it shows all the validated moves which a playr can make
478     validMoves = self.game.validatemove(character)
479
480     for i in range(len(validMoves)):
481         (x, y) = validMoves[i]
482         checks = self.tiles[x - 1, y - 1]
483         movesboxes.append(checks)
484         # highlighting all the possible moves with black boundary for better vision.
485     for i in range(len(movesboxes)):
486         self.gamewin.itemconfigure(
487             movesboxes[i], outline="black", width=2)
488
489     self.game.selected_tuple = (column, row)
490     # if a player wants to make a move then he must need to click on otehr tile.
491     elif (self.game.selected_tuple != None and (column, row) in self.game.validatemove(
492         self.game.player1.getChar(self.game.selected_tuple[0], self.game.selected_tuple[1]))):
493         # print("IF2")
494         self.status.config(
495             text="Player 1 Moved,Player 2 Turn", bg=f"{self.game.player2.color}")
496         # getting the coordinate of other tile
497         (x, y) = self.game.selected_tuple
498         # movind the token from one tile to other using movement function
499
500         self.game.char_movement((y, x), (row, column))
501         # removing all the highlighhed block after making a move
502         for i in range(self.board_size):
503             for j in range(self.board_size):
504                 self.gamewin.itemconfigure(
505                     self.tiles[i, j], outline="black", width=0)
506         self.game.selected_tuple = None
507         self.t = 0
508         # Assiging turn to other player if a player made a move.
509         time.sleep(1)
510         self.t = self.backup
511         p1.join()
512         p1 = threading.Thread(target=self.timeren)
513         thread.sleep(1)
514         # self.t=20
515         p1.start()
516         self.move = False
517
518     else:
519         # print("Else2")
520         self.game.selected_tuple = None
521         for i in range(self.board_size):
522             for j in range(self.board_size):
523                 self.gamewin.itemconfigure(
524                     self.tiles[i, j], outline="black", width=0)
525     else:
526         # similarly if other player want to do same thins
527         # all the conditions works similarly.

```

```

527 # all the conditions works similarly.
528 # print("ELSEEEEEEEEEEEEEEEEEEEEEEEEEEEEEe")
529 if (self.game.selected_tuple == None and self.game.player2.charlocCheck(column, row)):
530     self.status.config(
531         text=f".....Player 2 is Thinking.....", bg=f"{self.game.player2.color}")
532     # print("ELSEIF")
533     if (self.game.player1.charlocCheck(column, row)):
534         # print("ELSEIF 2 player if")
535         character = self.game.player1.getChar(column, row)
536     elif (self.game.player2.charlocCheck(column, row)):
537         character = self.game.player2.getChar(column, row)
538     # print("ELSEIF 2 player else")
539
540     validMoves = self.game.validatemove(character)
541
542     for i in range(len(validMoves)):
543         (x, y) = validMoves[i]
544         checks = self.tiles[x - 1, y - 1]
545         movesboxes.append(checks)
546
547     for i in range(len(movesboxes)):
548         self.gamewin.itemconfigure(
549             movesboxes[i], outline="black", width=2)
550
551     self.game.selected_tuple = (column, row)
552
553 elif (self.game.selected_tuple != None and (column, row) in self.game.validatemove(
554     self.game.player2.getChar(self.game.selected_tuple[0], self.game.selected_tuple[1]))):
555     # print("ELIF")
556     self.status.config(
557         text="Player 2 Moved,Player 1 Turn", bg=f"{self.game.player1.color}")
558     (x, y) = self.game.selected_tuple
559     self.game.char_movement((y, x), (row, column))
560
561     for i in range(self.board_size):
562         for j in range(self.board_size):
563             self.gamewin.itemconfigure(
564                 self.tiles[i, j], outline="black", width=0)
565     self.game.selected_tuple = None
566     # p1.stop()
567     # self.t=0
568     self.t = 0
569     time.sleep(1)
570     self.t = self.backup
571     p1.join()
572     del p1
573     p1 = threading.Thread(target=self.timeren)
574     # self.t=20
575     # time.sleep(1)
576     p1.start()
577     self.move = True
578 else:
579     # print("Else")
580     self.game.selected_tuple = None
581     for i in range(self.board_size):

```

```

580         self.game.selected_tuple = None
581     for i in range(self.board_size):
582         for j in range(self.board_size):
583             self.gamewin.itemconfigure(
584                 self.tiles[i, j], outline="black", width=0)
585
586     self.update()
587
588
589 if __name__ == "__main__":
590     # initializing our main class to play game
591     # Parameters:
592     # Board size:8,10,16
593     # Timelimit : in seconds
594     #player1="RED"or "GREEN"
595     #player2="RED or GREEN"
596     # AI="AI" if you want Bot to play as player 2 else None for Human
597     #AlphaBeta= "ON" or "OFF"
598     # Always the second player is assigned as AI
599     game = HalmaGame(8, 10, "GREEN", "RED", "AI", "ON") # For Human vs AI
600     # game= HalmaGame(8,10,"RED","GREEN",None,"ON") # for human vs Human
601

```

Phase 1 Source code:-

charutils.py

```
dominators-part1 > charutils.py ...
1  import copy
2
3  # Charcated Util Class which is used to create
4  #Objects of players when the game begins.
5
6  class CharacterUtil:
7      def __init__(self, color, boardSize):
8          # Setting each player with its resepective colors
9          self.color = color
10         self.starposchar(boardSize)
11         # Home and Goal Cordinates
12         # home cordinate is Goal cordinate for the other player
13         #and vice versa
14         self.homeCoord = []
15         self.goalCoord = []
16     # Function to get the starting postion of the charcters
17     def starposchar(self, boardSize):
18         # Creating empty list varibale to track the home
19         # Goal and Tokens
20         self.charss = []
21         self.home = []
22         self.goal = []
23         # ASsigning Iteration based on the board size.
24         if boardSize == 8:
25             lenmax = 5
26         elif boardSize == 10:
27             lenmax = 6
28         else: # default is 16 x 16
29             lenmax = 7
30         # Iterating though our borad to set the home GOal
31
32         for i in range(1, lenmax):
33             for j in range(1, lenmax):
34                 # and the positon of the player. so that we can keep a track later.
35                 if (i + j <= lenmax and i < lenmax and j < lenmax):
36                     self.charss.append(Utils(boardSize - j + 1, boardSize - i + 1))
37                     self.home.append((boardSize - j + 1, boardSize - i + 1))
38                     self.goal.append((j, i))
39         # For debugging PUrpse
40         print(self.charss)
41         print(self.home)
42         print(self.goal)
43
44
45         if self.color == 'RED':
46             # The other Player also have similar Cordinates the
47             # only change is that
48             temp = copy.deepcopy(self.home)
49             self.home = copy.deepcopy(self.goal)
50             # The goal cordinate for player 1 is Home Corindate of player 2
51
52             self.goal = temp
53             # and the Home Cordinate of player 1 is the Goal cordinate of player 2.
54             for i in range(len(self.charss)):
55                 self.charss[i].x = self.home[i][0]
56                 self.charss[i].y = self.home[i][1]
```

```

57
58
59
60 # FUnction to Check that the game is FINised Or not
61 # if all the players reached oppositon Home
62 # then the player wins the game.
63 def all_char_opp(self):
64     status = True
65     for p in self.charss:
66         if (p.x, p.y) in self.goal:
67             status = True
68         else:
69             status = False
70             break
71     return status
72
73 # Fucntion to Chekc the location of the chratcer
74 def charlocCheck(self,x,y):
75     # if the charcater is in X and Y coordiante it returns True
76     for i in range(len(self.charss)):
77         if(self.charss[i].x == x and self.charss[i].y== y ):
78             return True
79             break
80         else:
81             # Else It return false
82             pass
83     return False
84 # FUnction To check the Home Cordinate of the Chracter.
85 def charHomeCheck(self,x,y):
86     koor =(x, y)
87     # If Givenn Cordiante is in our H0me list that we alredy Trakced then it return true
88     if koor in self.home:
89         return True
90     else:
91         return False
92
93 #Function to Check the goal cordiante
94 def charGoalCheck(self,x,y):
95     koor =(x, y)
96     # If given Cord is Goal cord then true else false
97     if koor in self.goal:
98         return True
99     else:
100         return False
101 # FUnction to get the Charatcer
102 #Token from row and Column
103 def getChar(self, row, column):
104     i = 0
105     found = False
106     # If a chracter token is found withing our row and Columns
107
108     while i < len(self.charss) and not(found):
109         # it returns the chracters if it is avilible

```

```

109         # it returns the chracters if it is avaiible
110         if (self.charss[i].x == row and self.charss[i].y == column):
111             character = self.charss[i]
112             return character
113             found = True
114         else:
115             i +=1
116
117
118     # Fucntion to move the chracter from one Grid Locatio
119     # TO the other grid location
120     def char_movement(self, gridfrom, gridto):
121         # get the current Location
122         (x, y) = gridfrom
123         (x2, y2) = gridto# get the destination
124         # Looping though our Tokens
125         for p in self.charss:
126             # if its x coordinates matches with the current coordinate
127             if p.x == x and p.y == y:
128                 # and if it is in Home Cord then we can make a movement
129                 if (x, y) in self.home and (x2, y2) not in self.home:
130                     p.char_gone = True
131                     # Similarly if it is goal then we cannot make a move
132                     elif (x, y) not in self.goal and (x2, y2) in self.goal:
133                         p.char_arr = True
134                     # Assigning GOal Cordinate
135                     p.x = x2
136                     p.y = y2
137                     self.charss = sorted(self.charss, key=lambda p: (p.x, p.y))
138                     break
139
140     # Utils Class for More debugging Purpose to get each and ANY single variable.
141     class Utils:
142         def __init__(self, x, y):
143             #setting the Cordinates
144             self.setCoordinate(x, y)
145             #Print("CHAR INISITALIZE")
146             self.setChar_Gone(False)
147             self.setChar_arr(False)
148             # TO set the cordinates again
149             def setCoordinate(self,x, y):
150                 self.x = x
151                 self.y = y
152             # to set Char Dest
153             def setChar_Gone(self, char_gone):
154                 self.char_gone = char_gone
155             # To set Char Currnet
156             def setChar_arr(self, char_arr):
157                 self.char_arr = char_arr
158             # To get the X cordinates of Player
159             def getCoordinateX(self):
160                 return (self.x)
161             # TO get Y cordinates
162             def getCoordinateY(self):
163                 return (self.y)

```



```
163
164     def getCoordinate(self):
165         return (self.x, self.y)
```

Utilities.py

```
dominators_part1 > Utilities.py > ...
1   #Utilities Class That helps in various stage to get the data
2   #related to our aGame
3   class Utilities:
4       # Initializing Constructor so that we can easily get the Color and Cordianat
5       # of our Player
6       def __init__(self, x, y, color="BLACK", character=0):
7           self.x = x
8           self.y = y
9           self.color = color
10          self.character = character
11          # setting the color of a player that he choosed.
12          def ColorSetup(self, color):
13              self.color = color
14          #setting the chracter tokens
15          def CharTokenSetup(self, character):
16              self.character = character
17          #Getting the cordinates of the player.
18          def printCoordinate(self):
19              print(str(self.x) + str(self.y) + self.color + str(self.character))
```

Game.py

dominators_part1 > game.py > ...

```
1  #importing other package
2  from Utilities import Utilities
3  #Our Game Class which handels and Genrate moves in the back
4  #It is also responsible to generate Jumps while making moves.
5  class Game:
6      #initializing our construter of the game.
7      def __init__(self, boardSize, timeLimit, p1, p2, human):
8          #assingig all the parametres essential to play the game
9          self.boardSize = boardSize
10         self.timeLimit = timeLimit
11         self.player1 = p1
12         self.player2 = p2
13         #assigning therir respective color to the players
14         self.p1cop = p1 if p1.color == "GREEN" else p2
15         self.p2cop = p2 if p2.color == "RED" else p1
16         #assigning turn to a player
17         self.turn = 1
18         #getting the cordianes of our board
19         self.coordinate = [[Utilities(i, j) for i in range(self.boardSize)] for j in range(self.boardSize)]
20         self.depth = 2
21         #humans as both players are playing the game.
22         self.human = human
23
24         #Assigning maximum iterations according to our board size.
25         if self.boardSize == 8:
26             lenmax = 4
27         elif self.boardSize == 10:
28             lenmax = 5
29         else:
30             lenmax = 6
31         #looping through all the iterations.
32         for i in range(lenmax):
33             for j in range(lenmax):
34                 if (i + j < lenmax and i < 6 and j < 6):
35                     #Making red player home coordinate red and
36                     self.coordinate[i][j].color = "RED"
37                     self.coordinate[i][j].character = 2
38                     #making green player home coordinates green
39                     #Appending all the home and goal coordinates of
40                     #player 1 and Player 2
41
42                     self.p2cop.homeCoord.append(self.coordinate[i][j])
43                     self.p1cop.goalCoord.append(self.coordinate[i][j])
44                     #similarly for Player 2 Assigning Color and Home goal coordinates
45                     self.coordinate[self.boardSize - 1 - i][self.boardSize - 1 - j].color = "GREEN"
46                     self.coordinate[self.boardSize - 1 - i][self.boardSize - 1 - j].character = 1
47                     self.p1cop.goalCoord.append(self.coordinate[self.boardSize - 1 - i][self.boardSize - 1 - j])
48                     self.p2cop.homeCoord.append(self.coordinate[self.boardSize - 1 - i][self.boardSize - 1 - j])
49
50
51         #function to track Board Size
52         def gamesiz(self):
53             return self.boardSize
54         #Function to Chehck that the coordinate is empty or not to make a move
55         def checkboxemp(self,x,y):
```

```

54 #function to check that the coordinate is empty or not to make a move
55 def checkboxemp(self,x,y):
56     #if a box is empty then player can make a move there if eligible
57     #if empty it return True
58     if(self.player1.charlocCheck(x,y) or self.player2.charlocCheck(x,y)):
59         return False
60     #else It return false
61     else:
62         return True
63 # Fucntion to Check the H0me Coordinates of the player
64 def checkhomecord(self, player, x, y):
65     return(player.charHomeCheck(x,y))
66 #Function to check the goal coordinates of the player
67 # so that we can check that a player reached goal or not
68 def checkgoalcord(self, player, x, y):
69     return(player.charGoalCheck(x,y))
70 # Getting all the list of avaibalve positons to make a move.
71 def allemppositions(self, position, alone):
72     # mengecek semua yang berdelta 1 itu kosong, dan gak melebihi size board
73     x, y = position
74     list_of_positions = []# to retrive all the locaitions from the bord.
75     # if player is in the center and no ther token of otehr player is in negibour
76     # then it can move one step to any direction
77     if (alone == 1):
78         list_of_positions.append((x+1, y))
79         list_of_positions.append((x+1, y+1))
80         list_of_positions.append((x, y+1))
81         list_of_positions.append((x-1, y+1))
82         list_of_positions.append((x-1, y))
83         list_of_positions.append((x-1, y-1))
84         list_of_positions.append((x, y-1))
85         list_of_positions.append((x+1, y-1))
86     #if other player lies in the neghibouruing tile then we need to
87     #calculate jumps all over the boards
88     #appending all the avaiable positions to our positions list
89     else:
90         if (not(self.checkboxemp(x+1,y)) and (self.checkboxemp(x+2, y))):
91             list_of_positions.append((x+2, y))
92         if (not(self.checkboxemp(x-1,y)) and (self.checkboxemp(x-2, y))):
93             list_of_positions.append((x-2, y))
94         if (not(self.checkboxemp(x,y+1)) and (self.checkboxemp(x, y+2))):
95             list_of_positions.append((x, y+2))
96         if (not(self.checkboxemp(x,y-1)) and (self.checkboxemp(x, y-2))):
97             list_of_positions.append((x, y-2))
98         if (not(self.checkboxemp(x+1,y+1)) and (self.checkboxemp(x+2, y+2))):
99             list_of_positions.append((x+2, y+2))
100         if (not(self.checkboxemp(x+1,y-1)) and (self.checkboxemp(x+2, y-2))):
101             list_of_positions.append((x+2, y-2))
102         if (not(self.checkboxemp(x-1,y+1)) and (self.checkboxemp(x-2, y+2))):
103             list_of_positions.append((x-2, y+2))
104         if (not(self.checkboxemp(x-1,y-1)) and (self.checkboxemp(x-2, y-2))):
105             list_of_positions.append((x-2, y-2))
106
107     length = len(list_of_positions)
108     i = 0

```

```

109     #once we get all the position to make a move its time to get Valid Positons.
110     while (i < length):
111         (x, y) = list_of_positions[i]
112
113         if(x<1 or y<1 or x>self.boardSize or y>self.boardSize):
114             list_of_positions.remove(list_of_positions[i])
115             length -= 1
116
117         elif (alone == 1 and not(self.checkboxemp(x, y))):
118             list_of_positions.remove(list_of_positions[i])
119             length -= 1
120         else:
121             i += 1
122     #all valid positons are returned to the player to make a move
123
124
125
126     return list_of_positions
127 #Function to Check JUMP on the board based
128
129 def checkjump(self, position, jumps, last_position): # On the current Positon of a player
130     #jumps can be made is a plyer is in neghibouring tile and the next tile
131     # to the other player is Empty
132     lis_available_jump = self.allemppositions(position, 2)
133
134     try:
135         lis_available_jump.remove(last_position)
136     except:
137         pass
138
139     if (len(lis_available_jump) == 0):
140         return jumps
141     else:
142         # Getting all the valid tiles to make a jump
143         #from one plce to another
144
145         for i in range (len(lis_available_jump)):
146             # if a tile is not avaiable to jump it is removed.
147             if lis_available_jump[i] not in jumps:
148                 jumps.append(lis_available_jump[i])
149                 #with the help of Recursion we check eachjumps for tiles.
150                 self.checkjump(lis_available_jump[i], jumps, position)
151 # if a player is making a move then it is validated that player is making a correct move or
152 def validatemove(self, character):
153     if (self.player1.charlocCheck(character.x, character.y)):
154         player = self.player1
155         print("IF")
156     else:
157
158         player = self.player2
159         print(" ELSE")
160
161     ##getting the current position of the Player
162     current_position = (character.x, character.y)
163     print("GET CURRENT",current_position)

```

```

164
165
166     # Once we have all the available positions for the move. we
167     # can validate each move. and jumps
168     list_of_positions = self.all_positions(current_position, 1)
169     all_jumps = self.all_positions(current_position, 2)
170     # looping until we check all the availables
171     if (len(all_jumps) > 0):
172         #if a jump is available to move then it is appended to our list
173         for i in range (len(all_jumps)):
174             if (all_jumps[i] not in list_of_positions):
175                 list_of_positions.append(all_jumps[i])
176             jumps = []
177             # once all the jumps are retrieved we can validate to the available jumps to make a move
178             self.check_jump(all_jumps[i], jumps, current_position)
179             #looping until we get all the validated moves
180             if (len(jumps) > 0):
181                 for i in range (len(jumps)):
182                     if (jumps[i] not in list_of_positions):
183                         list_of_positions.append(jumps[i])
184             # same concept as above using List of Moves
185             length = len(list_of_positions)
186             print("Length",length)
187             i = 0
188
189             while (i < length):
190                 (x, y) = list_of_positions[i]
191                 print("X,y," ,list_of_positions[i])
192                 print("PLAYER WHILE",player)
193                 #checking all the home and goal coordinates of a player if it is not then we removed that
194                 # the list of positions
195                 if (character.char_arr and not(self.check_goal_coord(player, x, y))) or (character.char_goal):
196                     list_of_positions.remove(list_of_positions[i])
197                     length -= 1
198                 else:
199                     i += 1
200             # once we have a list we sort it
201             list_of_positions = sorted(list_of_positions, key=lambda tup: (tup[0], tup[1]))
202             return list_of_positions
203
204
205     # function in backend to move character from one player to another.
206     def char_movement(self, xpos, ypos):
207         #getting from coordinate
208
209         gridfrom = self.coordinate[xpos[0]-1][xpos[1]-1]
210         #getting the coordinate where we have to move the player
211         gridto = self.coordinate[ypos[0]-1][ypos[1]-1]
212         # Checking that the move is valid or not
213
214         if gridfrom.character == 0 or gridto.character != 0:
215             print("Invaliddddddddddd")
216             return
217         # only a valid move can be played
218         # once a move is validated

```

```
218     # once a move is validated
219     if gridfrom.character == 1:
220         self.p1cop.char_movement((gridfrom.x+1, gridfrom.y+1), (gridto.x+1, gridto.y+1))
221         # the move is played by our backend board which is then displayed on our scree
222         #using the interface that we created.
223     elif gridfrom.character == 2:
224         self.p2cop.char_movement((gridfrom.x+1, gridfrom.y+1), (gridto.x+1, gridto.y+1))
225     else:
226         print("Invalidddddddddddddd")
227         return
228     gridto.character = gridfrom.character
229     gridfrom.character = 0
```

Main.py

```
1  # Import Necessary Libraries
2  from charutils import CharacterUtil
3  from game import Game
4  import tkinter as tk
5  import threading
6  import time
7
8  # initialized a global variable
9  ch = 0
10
11 # main Game class HalmaGame
12
13
14 class HalmaGame:
15     # Initialize Constructor so that player can be assigned colors and their respective goal.
16     def __init__(self, boardsize, timelimit, player1, player2):
17         # assigning player from our characterUtils Class which is on Other script
18         self.player1 = CharacterUtil(player1, boardsize)
19         # assigning player from our characterUtils Class which is on Other script
20         self.player2 = CharacterUtil(player2, boardsize)
21         self.human = False
22         # Initialize Game object which is the main Class that Handles overall game moves and spawns.
23         self.game = Game(boardsize, timelimit, self.player1,
24                           self.player2, self.human)
25         # Turn so that each player can play a move simultaneously
26         self.turn = 1
27         # Initializing Time limit so that player has to play move within timelimit
28         self.timelimit = self.game.timelimit
29         # Initializing Our Game mode Which is GUI based
30         self.mode = "GUI"
31         self.play()
32
33     # play function runs a loop until the game is not finished .. each player gets
34     # one chance to play one move
35     # if a user forget to play his move
36     # within a time limit then he cannot make his move until his next turn
37
38     def play(self):
39         # initializing our GUI which is created using tkinter
40         self.winn = Halmaguimode(self.game)
41         # initializing first move so that a player can play his move
42         self.winn.move = True
43
44         self.turn = 2 if self.player1.color == "RED" else 1
45         self.winn.move = False if self.player1.color == "RED" else True
46         print(self.turn)
47         # setting the Status bar which shows Player turn.
48         # game always starts with Green Player as mentioned in the doc.
49         self.winn.status.config(
50             text=f"Green Player Start Game...", bg="#187c53")
51         print("here")
52         self.winn.t = self.timelimit
53         self.winn.backup = self.timelimit
54         # creating a threaded object so that we can also monitor the time limit.
55         p1 = threading.Thread(target=self.winn.timeren)
56         p1.start()
57         check = 0
```

```

55     p1.start()
56     checko = 0
57     # looping until a player wins the game
58     while (self.endgame() == True):
59         # endgame checks that a player has won or not.
60         # a player won if his all the spawns are in opponent home.
61         # which is goal in player case
62         if (self.turn == 2 and not (self.winn.move)):
63             print("PLAYER00000")
64             # playing Move and setting the status bar for player
65             self.winn.status.config(
66                 text="Player 2 Turn: ", bg=self.player2.color)
67
68             self.winn.makechar()
69             # Activating the timer so that player knows how much
70             # time is remaning to play a move.
71             if p1.isAlive():
72                 self.winn.timer3.config(
73                     text=f" {self.player2.color} , Time left = {self.winn.curtime}")
74                 # print("yes")
75                 # pass
76             else:
77                 # if a player unable to perform a move in given
78                 # specific time then
79                 # player lost his turn to play move. He she needs to wait until next turn.
80                 self.winn.t = 0
81                 time.sleep(1)
82                 self.winn.t = self.timelimit
83                 p1.join()
84                 self.game.selected_tuple = None
85                 # clearing all the moves if player not played any.
86                 for i in range(self.winn.board_size):
87                     for j in range(self.winn.board_size):
88                         self.winn.gamewin.itemconfigure(
89                             self.winn.tiles[i, j], outline="black", width=0)
90                 self.winn.move = True
91                 # assigning turn to next player.
92                 self.turn = 1
93                 # activaiting timer for next player
94
95                 p1 = threading.Thread(target=self.winn.timeren)
96                 p1.start()
97
98                 # self.timer()
99                 # self.winn.makechar()
100
101             elif (self.turn == 1 and self.winn.move == True):
102                 # setting the staus for next player
103                 self.winn.status.config(
104                     text="Player 1 Turn: ", bg=self.player1.color)
105
106                 self.winn.makechar()
107                 # if player makes his move then turn will be assigned to other player
108                 if p1.isAlive():
109                     self.winn.timer3.config(
110                         text=f" {self.player1.color} , Time left = {self.winn.curtime}")

```



```

113         #pass
114         # if player cannot make his move. his chance lost.
115         else:
116             self.winn.t = 0
117             time.sleep(1)
118             self.winn.t = self.timelimit
119             p1.join()
120             self.game.selected_tuple = None
121             # clearing all the highlighted board moves.
122             for i in range(self.winn.board_size):
123                 for j in range(self.winn.board_size):
124                     self.winn.gamewin.itemconfigure(
125                         self.winn.tiles[i, j], outline="black", width=0)
126             self.winn.move = False
127
128             p1 = threading.Thread(target=self.winn.timeren)
129             p1.start()
130
131             # print("else2")
132             t1 = threading.Thread(target=self.winn.timer())
133
134             # self.winn.move = False
135             self.turn = 2 if self.turn == 1 else 1
136             self.game.turn = 2 if self.game.turn == 1 else 1
137
138         # op=0
139         # while(True) :
140         #     op+=1
141         #     self.winn.status.config(text=f"Player {self.endgame()} Won The Game.")
142         #     time.sleep(1)
143         #     if op==5:
144         #         break
145         self.winn.t = 0
146         self.winn.makechar()
147         p1.join()
148         pls = self.endgame()
149         if pls == 2:
150
151             colr = self.player2.color
152         else:
153             colr = self.player1.color
154         self.winn.status.config(
155             text=f"Player {self.endgame()} Won The Game.", bg=colr)
156         self.winn.update()
157         time.sleep(5)
158         print(f"Player {self.endgame()} Won The Game.")
159         # Fucnction to Check that game has a winner or not..
160
161         def endgame(self):
162
163             # this function called everytime in loop to check that a player won or not

```

```

163         # this function called everytime in loop to check that a player won or not
164         # after every single move made by player
165         if (self.player1.all_char_opp() and not (self.player2.all_char_opp())):
166             return 1
167         # if player reach its goal that is home for other player then he/she wins
168         elif (self.player2.all_char_opp() and not (self.player1.all_char_opp())):
169             return 2
170         # otherwise game continues until we get a winner
171         else:
172             return True # 0 for not terminate
173
174     # Gui Class which is made using tkinter
175     # this class handels all the functionality of GUI to show the moves
176     # Show players, status, timer and evrything else.
177
178
179     class Halmaguimode(tk.Tk):
180         def __init__(self, board, *args, **kwargs):
181             # initialize the parent class of tkinter
182             tk.Tk.__init__(self, *args, **kwargs)
183             # setting it to true so that our window can be resized
184             self.resizable(True, True)
185
186             self.configure(bg='#2F4F4F') # setting Background color
187             # initializing Our game Objects which helps in calculating game.
188             self.game = board
189             self.time = 0
190             self.board_size = board.gamesiz()
191             self.t = 20
192             self.backup = 0
193
194             # Assigning X and Y values to our grid bord so that we can
195             # calculate each move by its Cordinate
196             # that is X,Y in our case it is 1,A for Eg.
197             for i in range(self.board_size):
198                 roww = tk.Label(self, text=i + 1, font='Times',
199                                bg='#2F4F4F', fg='#DCDCDC')
200                 roww.grid(row=i + 1, column=0)
201
202                 coll = tk.Label(self, text=chr(i + 97).upper(),
203                                font='Times', bg='#2F4F4F', fg='#DCDCDC')
204                 coll.grid(row=0, column=i + 1)
205             # Making Staus bar to print the current status of that that is happening
206             self.status = tk.Label(self, height=3, text="Welcome to Halma Game.....Green Player Start Game", width=50,
207                                   relief="raised", font="Times", bd=3, bg="#001414",
208                                   fg="#DCDCDC") # Customizing the status bar with some additional prameters
209             self.timer3 = tk.Label(self, height=3, text="TIMER...", width=50, relief="raised", font="Times", bd=3,
210                                   bg="#001414",
211                                   fg="#DCDCDC")
212             # Making our Game timer so that each player has same time to play a move
213
214             # status.grid(row=0,column=i+1)
215             # initializing our Tiles for the Game.
216
217             self.tiles = {}
218

```

```

218
219     # Making a Blank Canvas of size 590*590
220     # with some additional Style Parameters
221     self.gamewin = tk.Canvas(self, width=590, height=590, bd=4,
222                               relief="ridge", bg="#778899", highlightthickness=0)
223     # using Grid instead of pack because we cannot use pack with grid.
224     self.gamewin.grid(
225         row=1, column=1, columnspan=self.board_size, rowspan=self.board_size)
226     # similarly setting status bar using grid method
227     self.status.grid(columnspan=self.board_size +
228                      2, rowspan=self.board_size)
229     # similarly setting Timer bar using grid method
230     self.timer3.grid(columnspan=self.board_size +
231                      2, rowspan=self.board_size)
232     self.title('Halma Player vs Player') # main title of the window
233     # some additional configuration
234     self.columnconfigure(0, minsize=50)
235     self.rowconfigure(0, minsize=50)
236     self.columnconfigure(self.board_size + 1, minsize=50)
237     self.rowconfigure(self.board_size + 1, minsize=50)
238     self.gamewin.bind("<Configure>", self.makegrid)
239     self.game.selected_tuple = None
240
241     # our Timer which is called in each player move. so that each player
242     # gets equal amount of time to make a move.
243
244     def timeren(self):
245         while self.t != 0:
246             print("TIMEEEEEEEEE", self.t)
247             # using divmod to get the time in min and secs
248             mins, secs = divmod(self.t, 60)
249             self.curtime = '{:02d}:{:02d}'.format(mins, secs)
250             # subtracting our decremented by 1 after a sec..
251             time.sleep(1)
252             # print(self.ti)
253             if self.t == 0:
254                 break
255             self.t -= 1
256         print("HEER BREAKDE")
257
258     # print(self.curtime)
259     # if self.t==0:
260     #     self.t=60
261     #     self.status.config(text="Timeleft")
262
263     # Function used to make grid in our window canvas
264
265     def makegrid(self, event=None):
266         self.gamewin.delete("checks")
267         heightcanva = 600
268         marginnsiz = 1
269         # Calculating each box size of our game as our game has 3 boards 8,10,16
270         boxx = int(heightcanva / self.board_size)
271         # looping through our board row and col so that we can create an interface
272         # that is grided so that we can easily make moves.
273         for col in range(self.board_size):
274             for row in range(self.board_size):

```

```

269 # looping thoug our board row and col so that we can create a interface
270 # that is grided so that we can easily make moves.
271 for col in range(self.board_size):
272     for row in range(self.board_size):
273         # calculcating all the cordiates
274         x1 = col * boxx + marginnsiz / 2
275         y1 = row * boxx + marginnsiz / 2
276         x2 = (col + 1) * boxx - marginnsiz / 2
277         y2 = (row + 1) * boxx - marginnsiz / 2
278
279         if (self.board_size == 8):
280             player1 = 4
281             player2 = 10
282         elif (self.board_size == 10):
283             player1 = 5
284             player2 = 13
285         else:
286             player1 = 6
287             player2 = 24
288         # assiging colors to each player Home with their respective colors
289         if ((row + col) < player1):
290             if ((row + col) % 2 == 0):
291                 color = '#AC352E'
292             else:
293                 color = '#D0352E'
294
295         # Red color always on the top left of the game screen
296         elif ((row + col) > player2):
297             if ((row + col) % 2 == 0):
298                 color = '#12C47A'
299         # green color always on the bottom left of the game screen
300         else:
301             color = '#0FA868'
302         # all other tiles are maked as playzone of the agame with a tan collor
303         else:
304             if ((row + col) % 2 == 0):
305                 color = '#ECCB96'
306             else:
307                 color = '#BAA077'
308         # creating rectangle Checked Grids.
309         checks = self.gamewin.create_rectangle(
310             x1, y1, x2, y2, tags="checks", width=0, fill=color)
311         self.tiles[col, row] = checks
312         # binding our checkers with our onpress function which is activated whenever we click to make move.
313         self.gamewin.tag_bind(
314             checks, "<1>", lambda event, row=row, col=col: self.onpress(row + 1, col + 1))
315
316     self.makechar()
317 # Fucntion used to make Charcter players that is red player and Green player
318 # we used Oval shape to assign a shape to a player of theri respective color
319
320 def makechar(self):
321
322     canvas_width = 600
323     heightcanva = 600
324     marginnsiz = 10

```

```

324 marginnsiz = 10
325 # similarly calucated box size as above
326 boxx = int(heightcanva / self.board_size)
327 # Initializing Our Tokens of the game. for both Players
328 self.p1char = self.game.player1.charss
329 self.p2char = self.game.player2.charss
330 # delete previous tokens canvas
331 self.gamewin.delete('character')
332 c = 0
333
334 for i in [range(len(self.p1char)), range(len(self.p2char))]:
335     for i in i:
336         if c == 0:
337             col = self.p1char[i].x - 1
338             row = self.p1char[i].y - 1
339
340         else:
341             col = self.p2char[i].x - 1
342             row = self.p2char[i].y - 1
343             # calculation all the cordinales so that we can create
344             # a oval shape for each player with their respective colors.
345
346             x1 = col * boxx + marginnsiz / 2
347             y1 = row * boxx + marginnsiz / 2
348             # Using SImple Mathematics
349             x2 = (col + 1) * boxx - marginnsiz / 2
350             y2 = (row + 1) * boxx - marginnsiz / 2
351             # Creating Shapes for charcted with their respecitve colors
352             if c == 0:
353                 if (self.game.player1.color == "GREEN"):
354                     character = self.gamewin.create_oval(
355                         x1, y1, x2, y2, tags="character", width=0, fill="#187c53")
356
357                 else:
358                     character = self.gamewin.create_oval(
359                         x1, y1, x2, y2, tags="character", width=0, fill="#9b423c")
360
361                 self.gamewin.tag_bind(
362                     character, "<1>", lambda event, row=row, col=col: self.onpress(row + 1, col + 1))
363
364             # And Binding each of our tokens of game with our onpress fucntion
365             # so that whenever a player click his token.
366             # he can look upon all the
367             # possible Moves
368             else:
369                 if (self.game.player1.color == "GREEN"):
370
371                     character = self.gamewin.create_oval(
372                         x1, y1, x2, y2, tags="character", width=0, fill="#9b423c")
373                 else:
374                     character = self.gamewin.create_oval(
375                         x1, y1, x2, y2, tags="character", width=0, fill="#187c53")
376                 self.gamewin.tag_bind(
377                     character, "<1>", lambda event, row=row, col=col: self.onpress(row + 1, col + 1))
378             c += 1
379 # updating our game interface

```

```

377         c += 1
378     # updating our game interface.
379     self.update()
380     # function works on clicking a token of a game.
381     # if a player press one of his pice of token this funciton
382     # shows all the possible moves in the interface with a boundary Highlighted
383     # and if a player presses another tile to move.
384     # this function is responsible to make a move using interface.
385
386
387 def onpress(self, row, column):
388     global ch
389     checks = self.tiles[column - 1, row - 1]
390     # all possible moves.
391     movesboxes = []
392     movesboxes.append(checks)
393     # print("CHHH",ch)
394     if self.move:
395         # if Player one thinking to make a move.
396         # print("00000000000000000000000000000000")
397         if (self.game.selected_tuple == None and self.game.player1.charlocCheck(column, row)):
398             self.status.config(
399                 text=f".....Player 1 is Thinking.....", bg=f"{self.game.player1.color}")
400             # whenever he select one of his token
401             if (self.game.player1.charlocCheck(column, row)):
402                 character = self.game.player1.getChar(column, row)
403             elif (self.game.player2.charlocCheck(column, row)):
404                 character = self.game.player2.getChar(column, row)
405             # it shows all the validated moves which a playr can make
406             validMoves = self.game.validatemove(character)
407
408             for i in range(len(validMoves)):
409                 (x, y) = validMoves[i]
410                 checks = self.tiles[x - 1, y - 1]
411                 movesboxes.append(checks)
412                 # highlighting alll the possible moves with black boundary for better vision.
413             for i in range(len(movesboxes)):
414                 self.gamewin.itemconfigure(
415                     movesboxes[i], outline="black", width=2)
416
417             self.game.selected_tuple = (column, row)
418             # if a player wants to make a move then he must need to click on otehr tile.
419             elif (self.game.selected_tuple != None and (column, row) in self.game.validatemove(
420                 self.game.player1.getChar(self.game.selected_tuple[0], self.game.selected_tuple[1]))):
421                 # print("IF2")
422                 self.status.config(
423                     text="Player 1 Moved,Player 2 Turn", bg=f"{self.game.player2.color}")
424                 # getting the cordinate of other tile
425                 (x, y) = self.game.selected_tuple
426                 # movind the token from one tile to other using movement function
427
428                 self.game.char_movement((y, x), (row, column))
429                 # removing all the highlighhed block after making a move
430                 for i in range(self.board_size):
431                     for j in range(self.board_size):

```

```

431         for j in range(self.board_size):
432             self.gamewin.itemconfigure(
433                 self.tiles[i, j], outline="black", width=0)
434         self.game.selected_tuple = None
435         self.t = 0
436         # Assigning turn to other player if a player made a move.
437         time.sleep(1)
438         self.t = self.backup
439         p1.join()
440         p1 = threading.Thread(target=self.timeren)
441         thread.sleep(1)
442         # self.t=20
443         p1.start()
444         self.move = False
445
446     else:
447         # print("Else2")
448         self.game.selected_tuple = None
449         for i in range(self.board_size):
450             for j in range(self.board_size):
451                 self.gamewin.itemconfigure(
452                     self.tiles[i, j], outline="black", width=0)
453     else:
454         # similarly if other player want to do same thins
455         # all the conditions works similarly.
456         # print("ELSEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEe")
457         if (self.game.selected_tuple == None and self.game.player2.charlocCheck(column, row)):
458             self.status.config(
459                 text=f".....Player 2 is Thinking.....", bg=f"{self.game.player2.color}")
460             # print("ELSEIF")
461             if (self.game.player1.charlocCheck(column, row)):
462                 # print("ELSEIF 2 player if")
463                 character = self.game.player1.getChar(column, row)
464             elif (self.game.player2.charlocCheck(column, row)):
465                 character = self.game.player2.getChar(column, row)
466             # print("ELSEIF 2 player else")
467
468             validMoves = self.game.validatemove(character)
469
470             for i in range(len(validMoves)):
471                 (x, y) = validMoves[i]
472                 checks = self.tiles[x - 1, y - 1]
473                 movesboxes.append(checks)
474
475             for i in range(len(movesboxes)):
476                 self.gamewin.itemconfigure(
477                     movesboxes[i], outline="black", width=2)
478
479             self.game.selected_tuple = (column, row)
480
481         elif (self.game.selected_tuple != None and (column, row) in self.game.validatemove(
482             self.game.player2.getChar(self.game.selected_tuple[0], self.game.selected_tuple[1]))):
483             # print("ELIF")
484             self.status.config(
485                 text="Player 2 Moved,Player 1 Turn", bg=f"{self.game.player1.color}")

```

```

484         self.status.config(
485             text="Player 2 Moved,Player 1 Turn", bg=f"{self.game.player1.color}")
486         (x, y) = self.game.selected_tuple
487         self.game.char_movement((y, x), (row, column))
488
489         for i in range(self.board_size):
490             for j in range(self.board_size):
491                 self.gamewin.itemconfigure(
492                     self.tiles[i, j], outline="black", width=0)
493         self.game.selected_tuple = None
494         # p1.stop()
495         # self.t=0
496         self.t = 0
497         time.sleep(1)
498         self.t = self.backup
499         p1.join()
500         del p1
501         p1 = threading.Thread(target=self.timeren)
502         # self.t=20
503         # time.sleep(1)
504         p1.start()
505         self.move = True
506     else:
507         # print("Else")
508         self.game.selected_tuple = None
509         for i in range(self.board_size):
510             for j in range(self.board_size):
511                 self.gamewin.itemconfigure(
512                     self.tiles[i, j], outline="black", width=0)
513
514         self.update()
515
516
517 if __name__ == "__main__":
518     # initializing our main class to play game
519     # Parameters:
520     # Board size:8,10,16
521     # Timelimit : in seconds
522     #player1="RED"or "GREEN"
523     #player2="RED or GREEN"
524     game = HalmaGame(12, 20, "RED", "GREEN")
525

```