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.

Calculategoal()

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

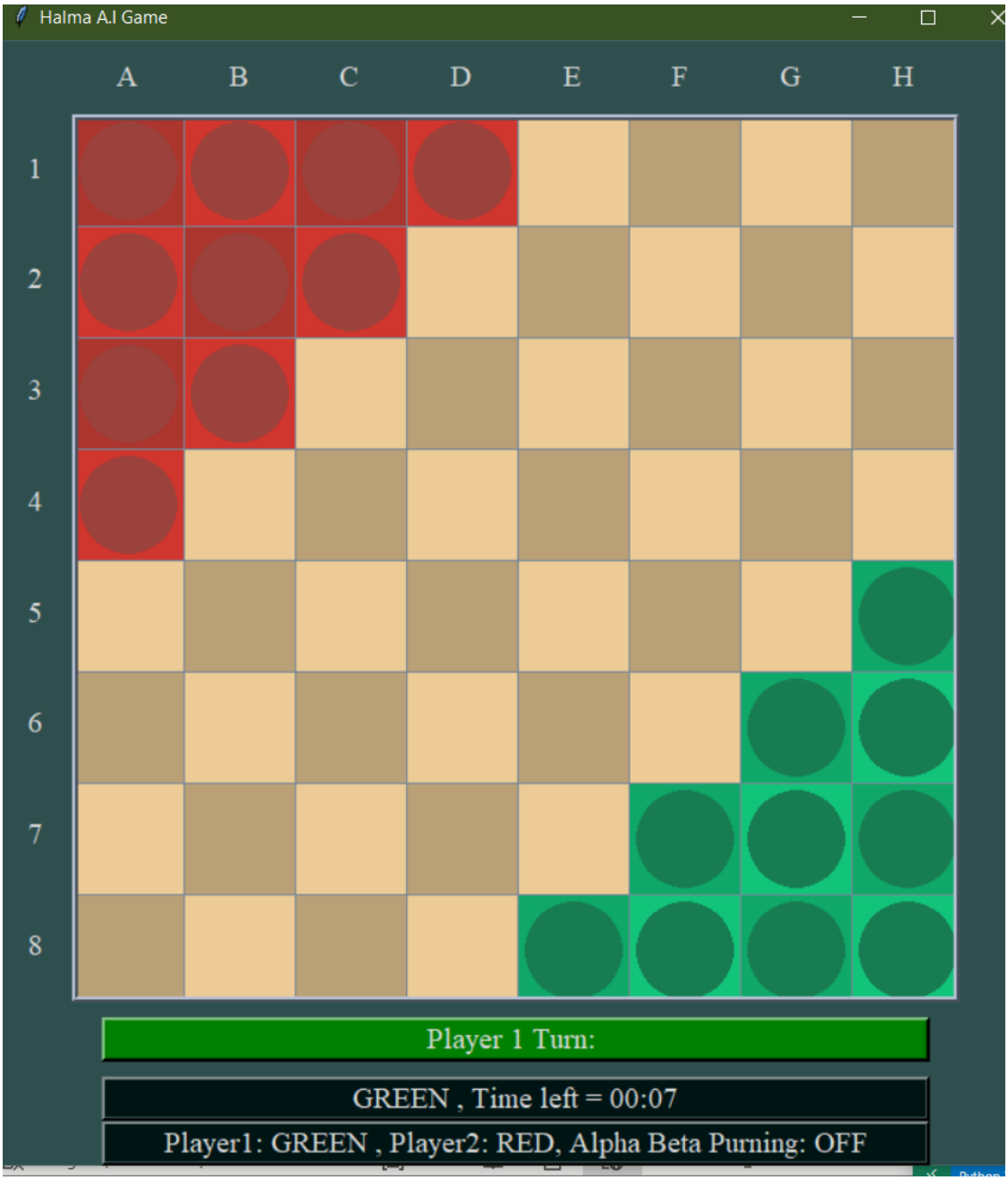
Gabriel Proctor:

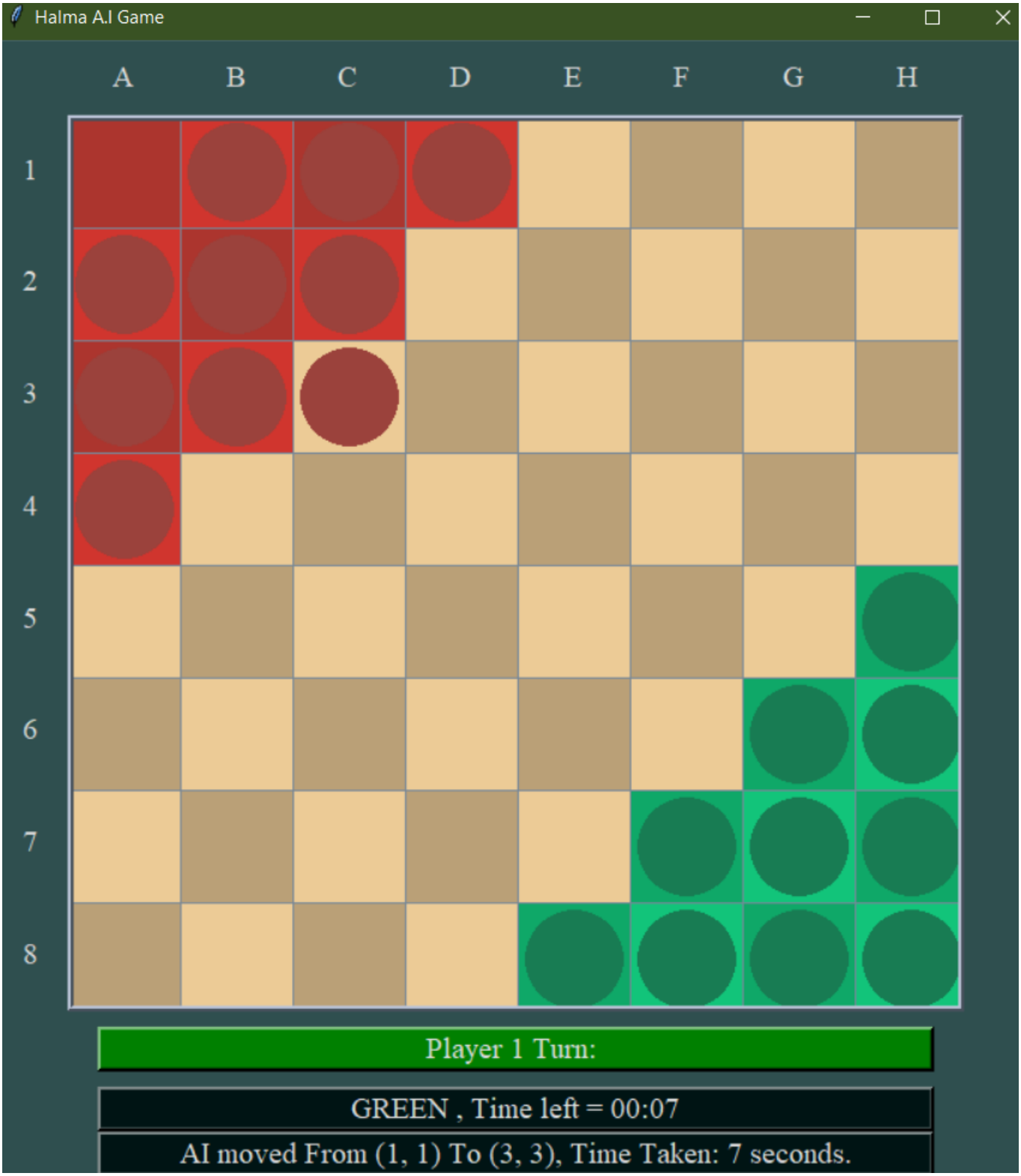
Worked on GUI class to get the board extra functionality that is needed for phase 2.

30% work done

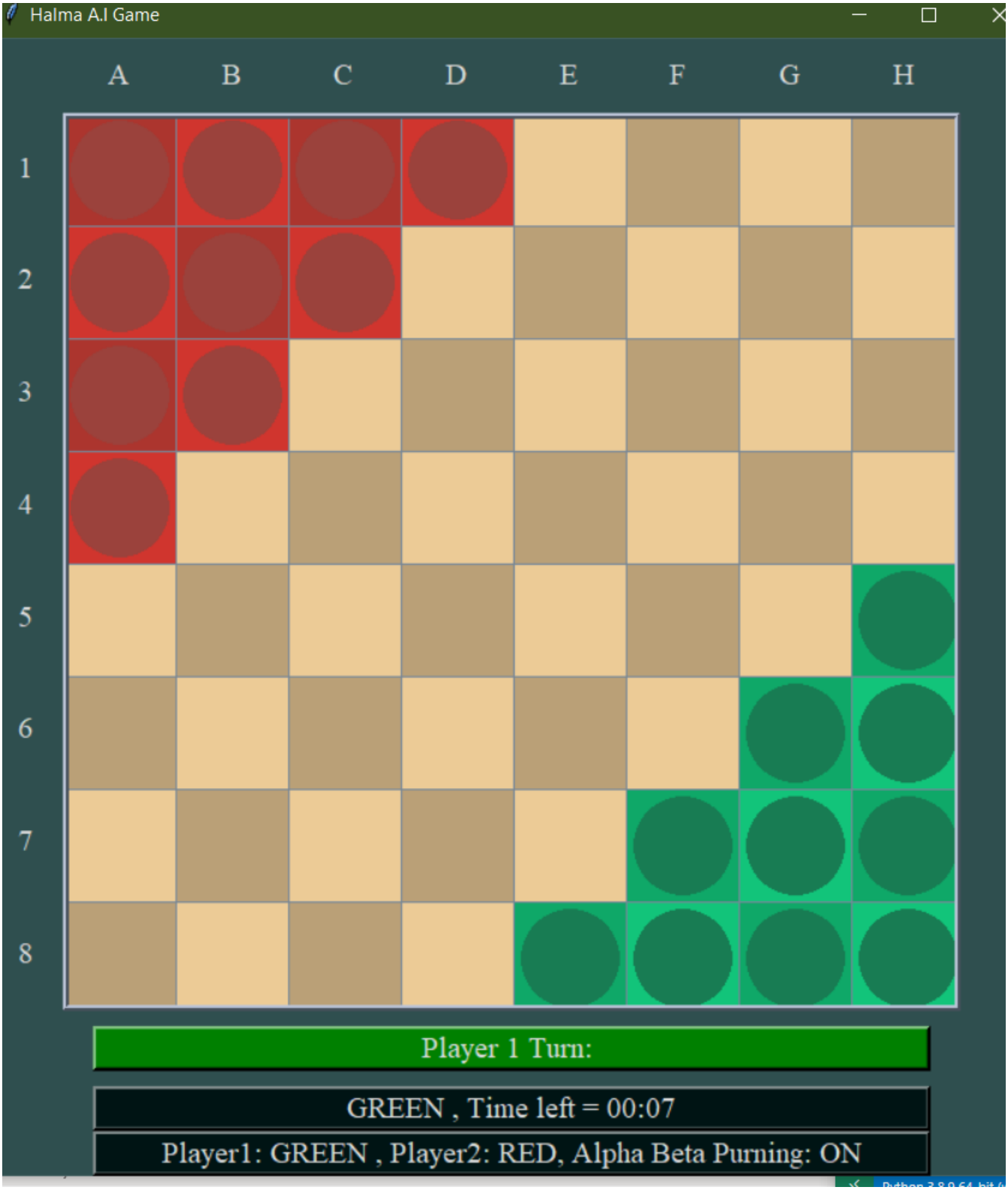
Functionality Checklist.

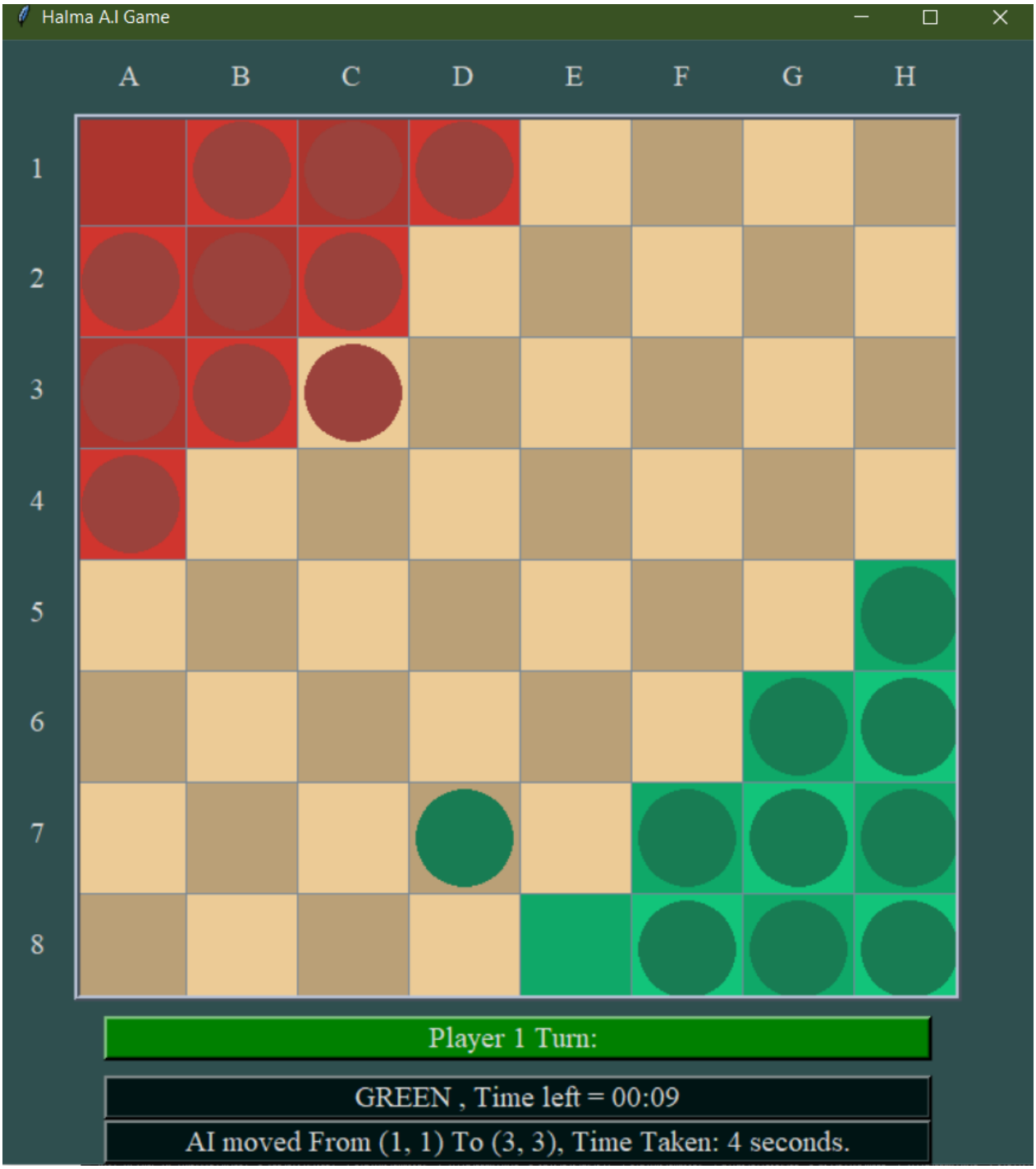
When Ab pruning is off: Red is AI and when an AI moves its turn it shows how much time is being taken on the grid as Time Taken.

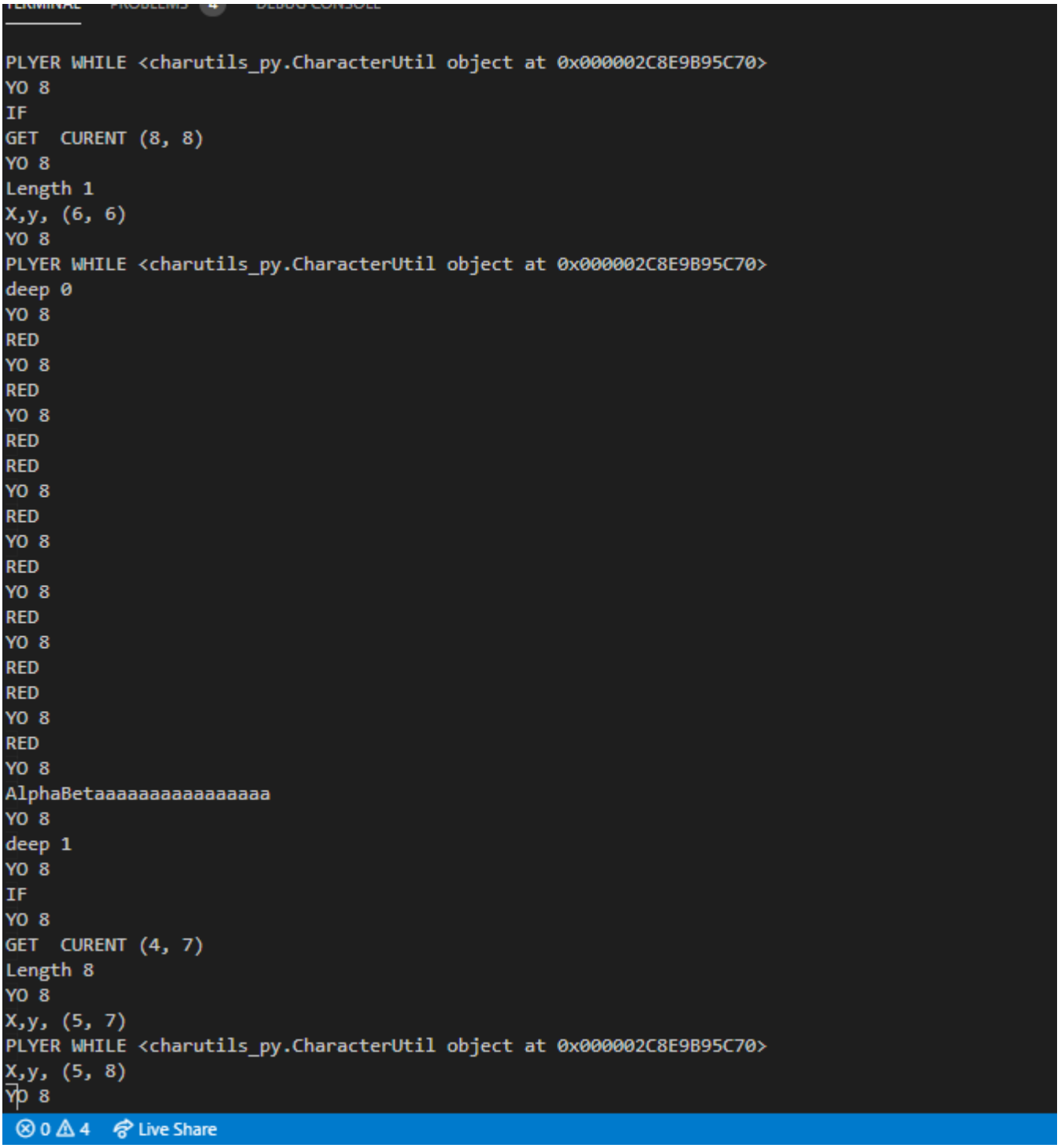



Halma A.I Game
A B C D E F G H
1 2 3 4 5 6 7 8
Player 1 Turn:
GREEN , Time left = 00:07
AI moved From (1, 1) To (3, 3), Time Taken: 7 seconds.

When Ab pruning is on:-

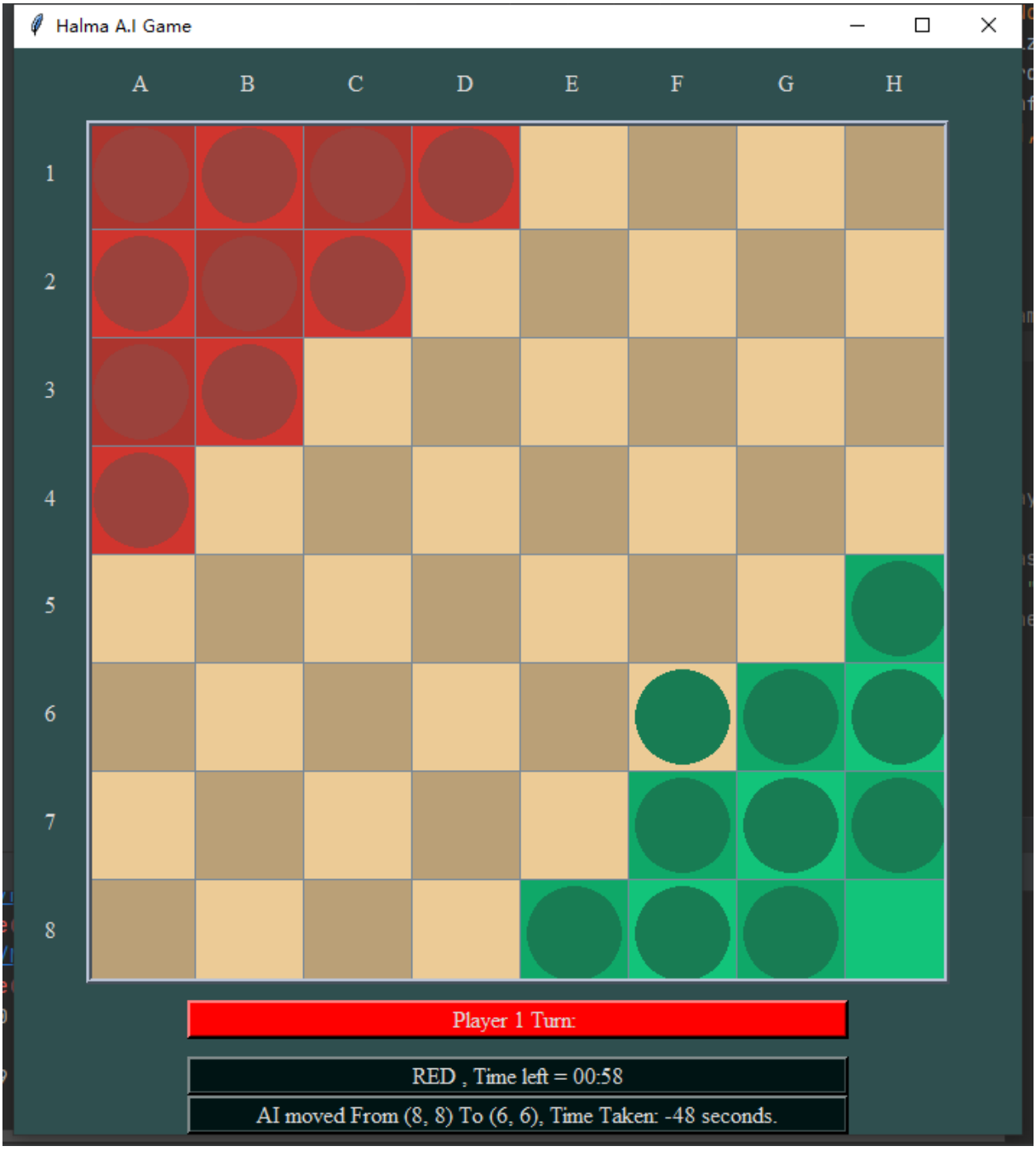


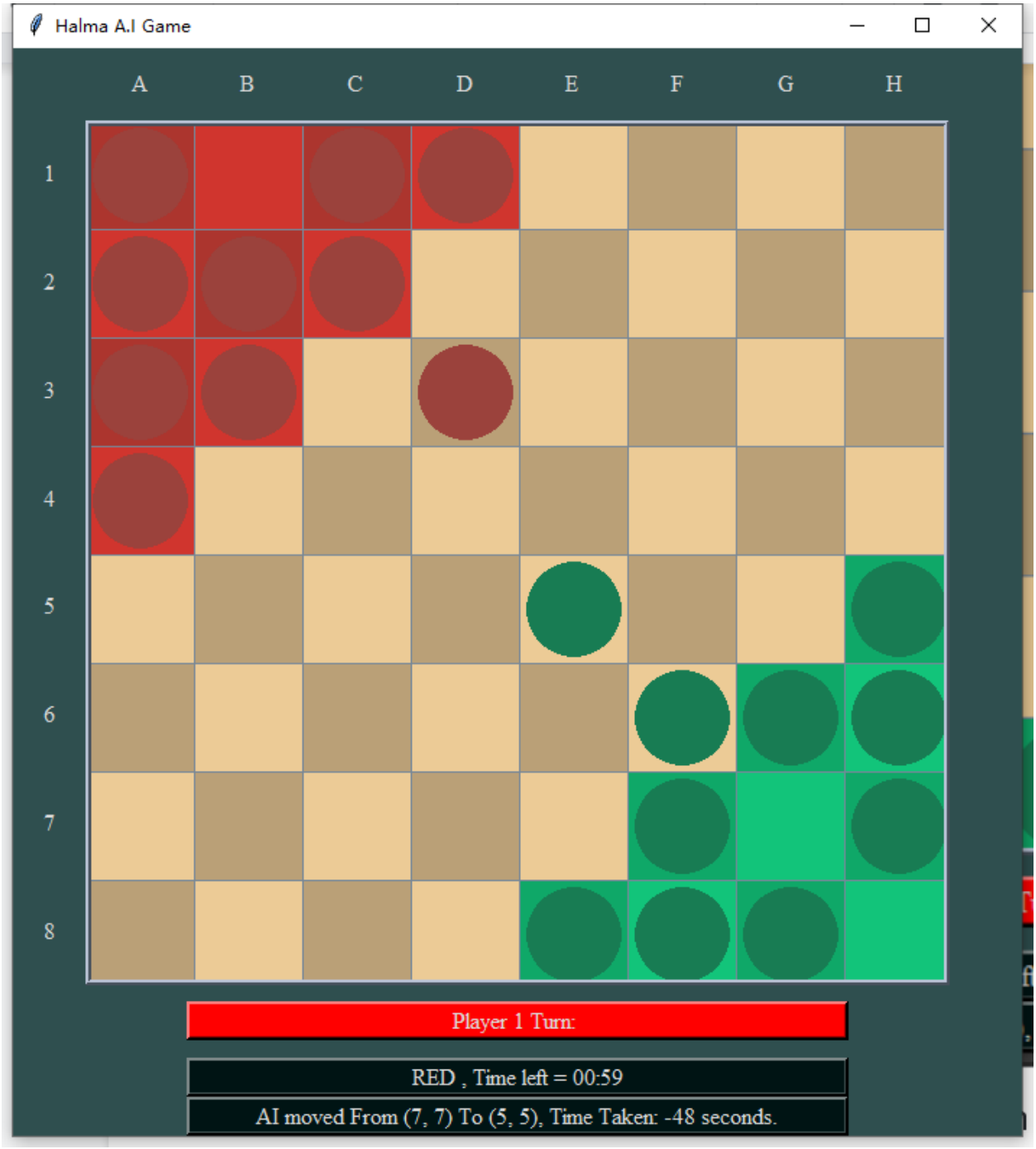

Halma A.I Game
A B C D E F G H
1 2 3 4 5 6 7 8
Player 1 Turn:
GREEN , Time left = 00:09
AI moved From (1, 1) To (3, 3), Time Taken: 4 seconds.

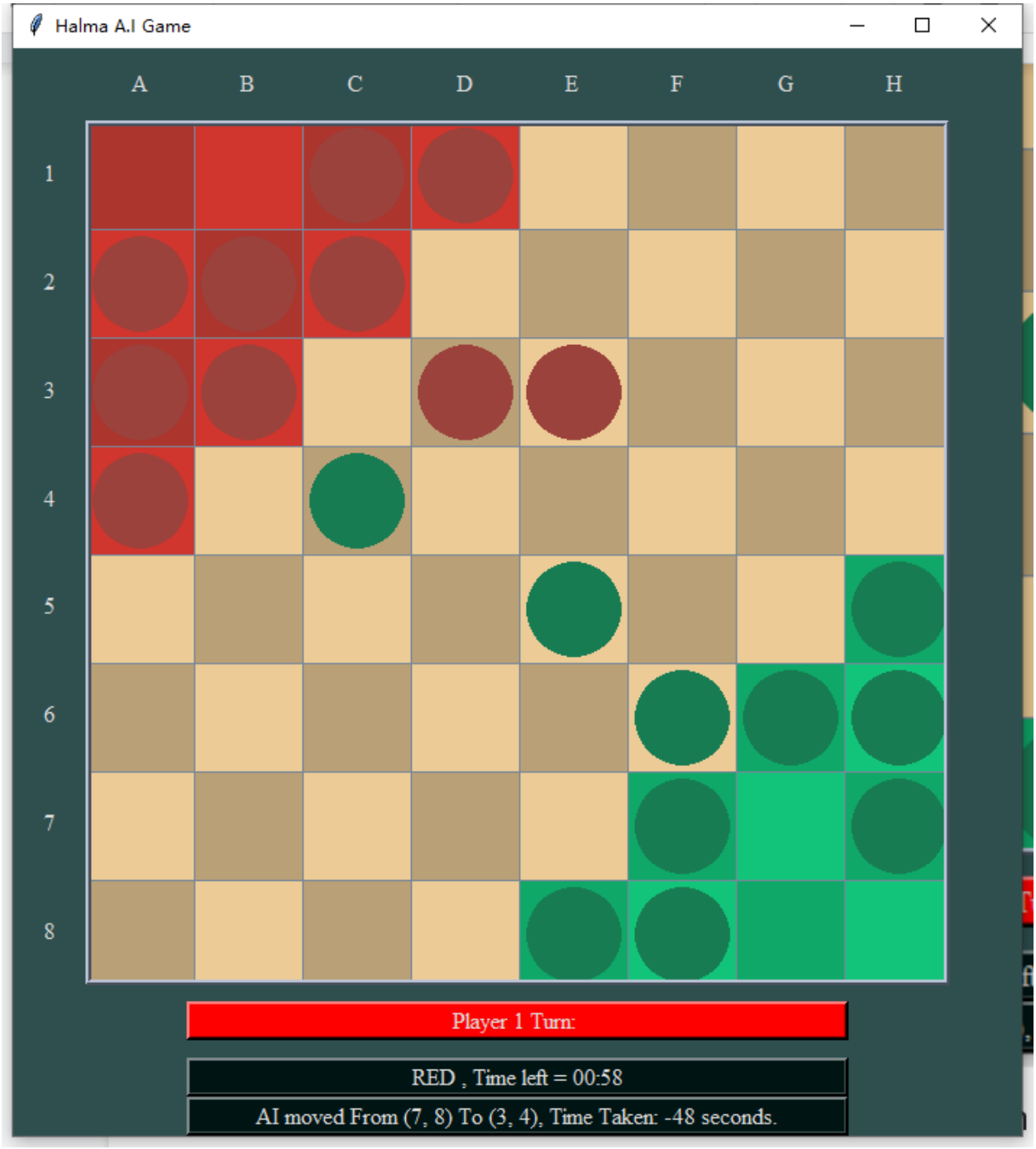

PLYER WHILE <charutils_py.CharacterUtil object at 0x000002C8E9B95C70>
YO 8
IF
GET CURENT (8, 8)
YO 8
Length 1
X,y, (6, 6)
YO 8
PLYER WHILE <charutils_py.CharacterUtil object at 0x000002C8E9B95C70>
deep 0
YO 8
RED
YO 8
RED
YO 8
RED
RED
YO 8
RED
YO 8
RED
YO 8
RED
YO 8
RED
RED
YO 8
RED
YO 8
AlphaBetaaaaaaaaaaaaaaa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

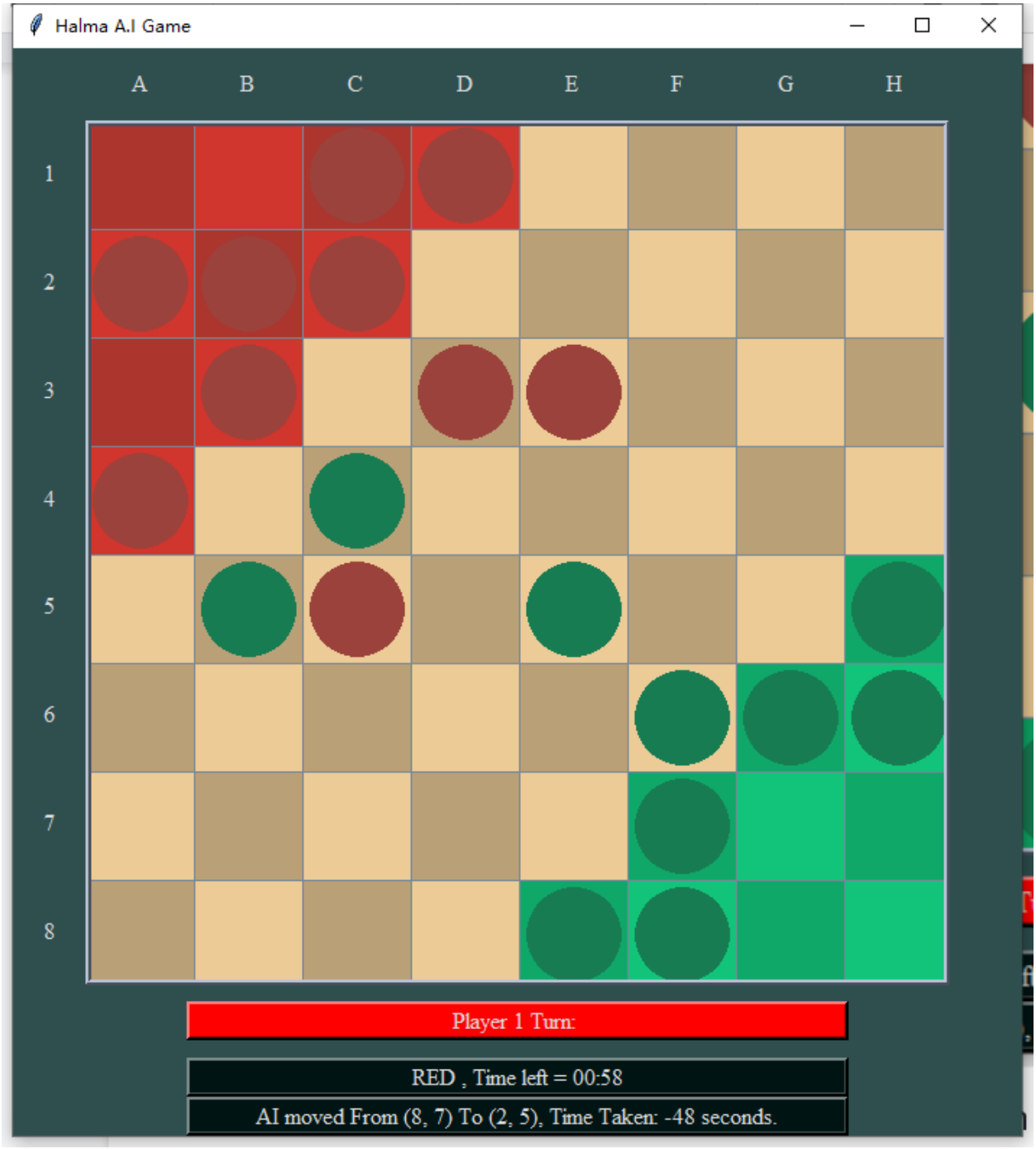
```
YO 8
PLYER WHILE <charutils_py.CharacterUtil object at 0x000002C8E9B95C70>
YO 8
X,y, (4, 8)
PLYER WHILE <charutils_py.CharacterUtil object at 0x000002C8E9B95C70>
X,y, (3, 8)
YO 8
PLYER WHILE <charutils_py.CharacterUtil object at 0x000002C8E9B95C70>
X,y, (3, 7)
PLYER WHILE <charutils_py.CharacterUtil 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
```

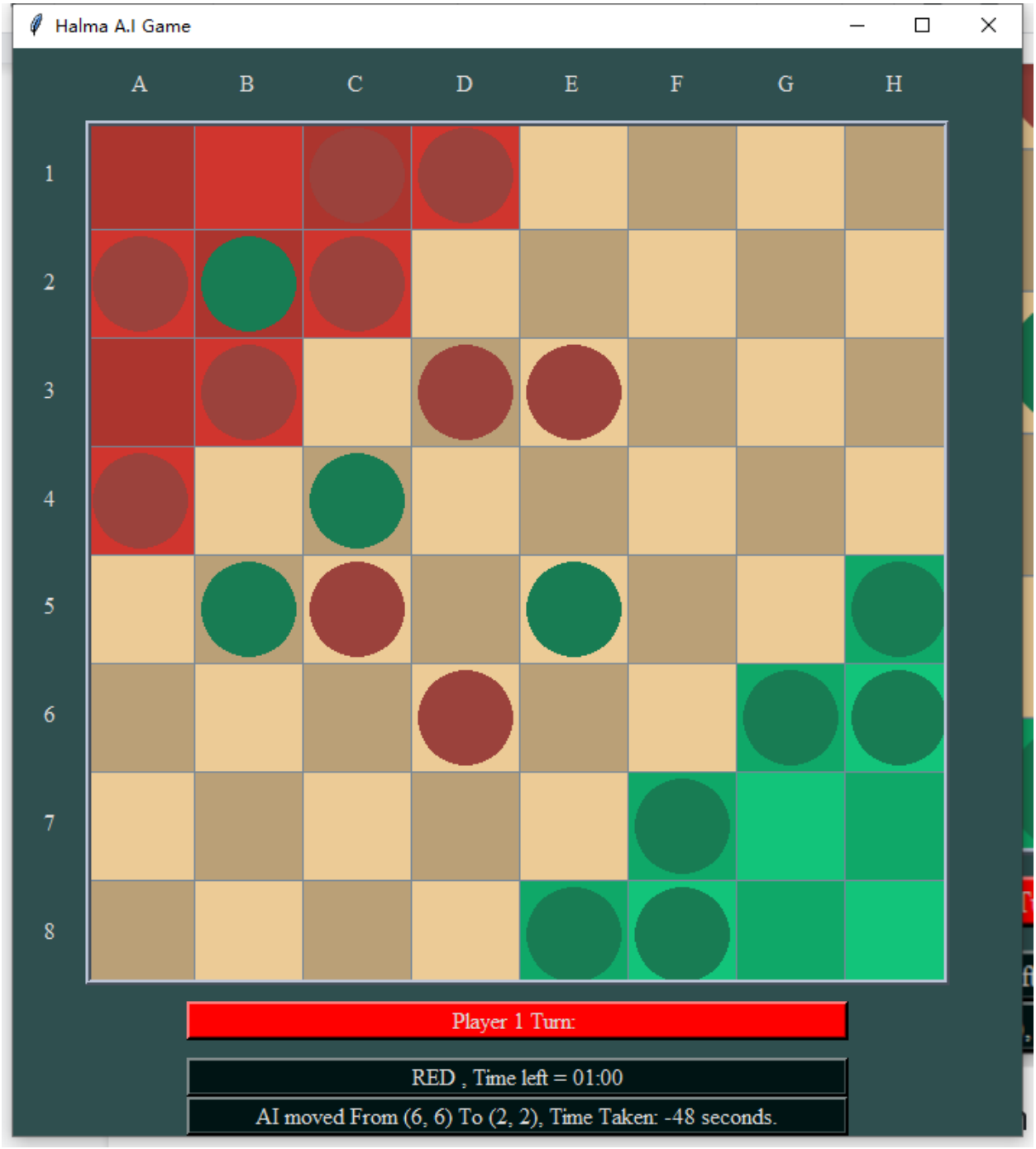
Demos:

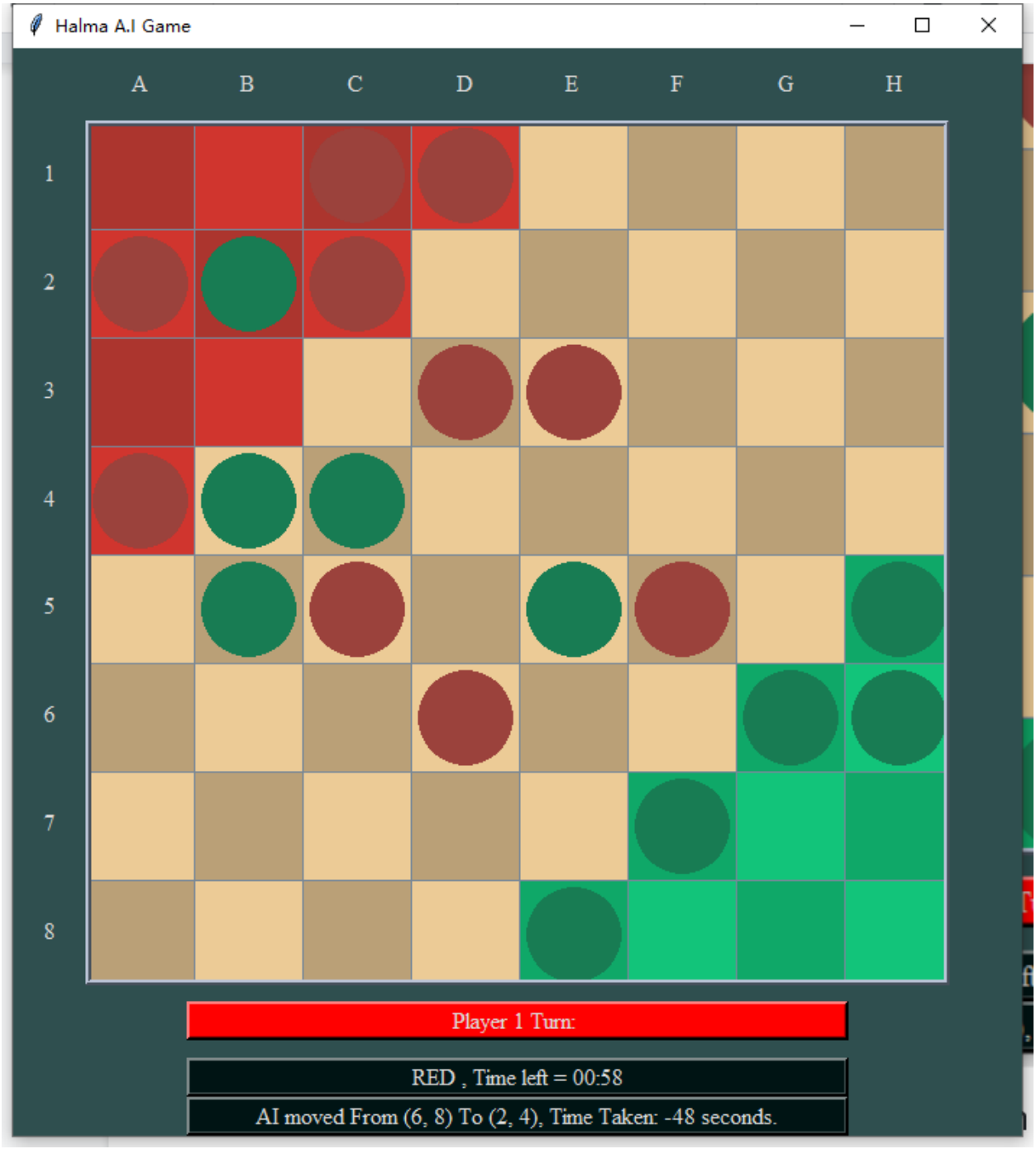

Halma A.I Game
A B C D E F G H
1 2 3 4 5 6 7 8
Player 1 Turn:
RED , Time left = 00:58
AI moved From (8, 8) To (6, 6), Time Taken: -48 seconds.

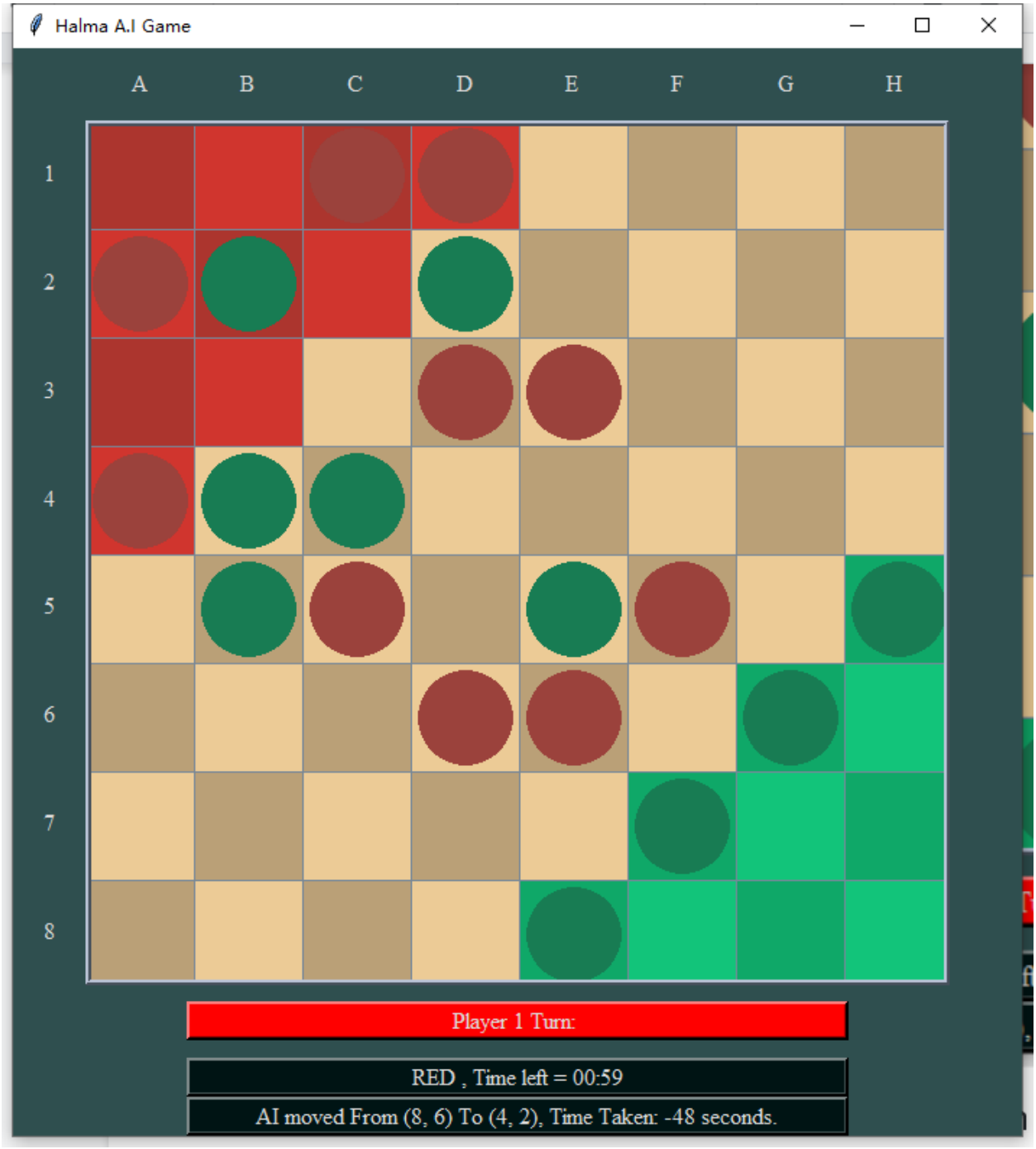

Halma A.I Game
A B C D E F G H
1 2 3 4 5 6 7 8
Player 1 Turn:
RED , Time left = 00:59
AI moved From (7, 7) To (5, 5), Time Taken: -48 seconds.

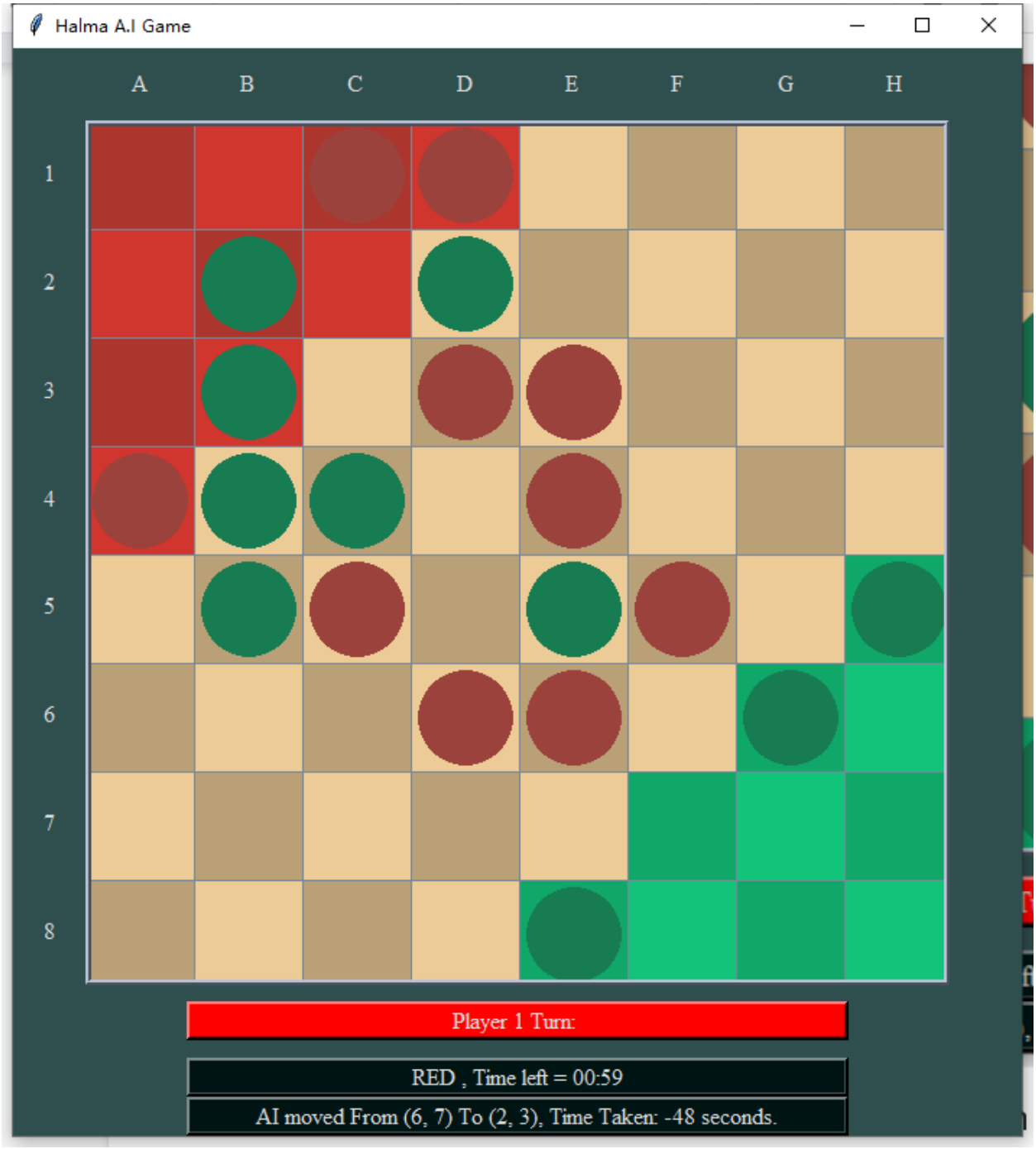

Halma A.I Game
A B C D E F G H
1 2 3 4 5 6 7 8
Player 1 Turn:
RED , Time left = 00:58
AI moved From (7, 8) To (3, 4), Time Taken: -48 seconds.

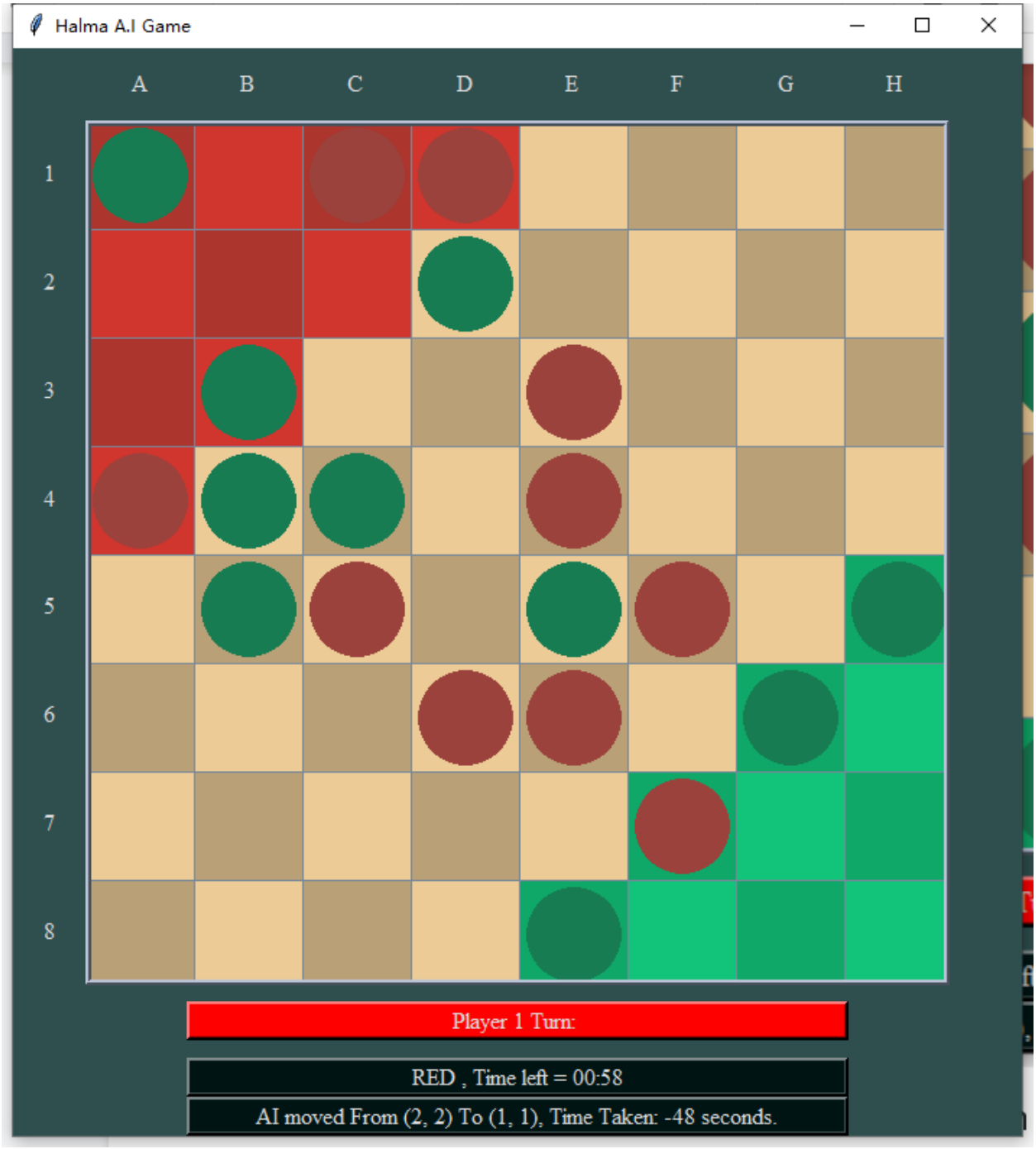

Halma A.I Game
A B C D E F G H
1 2 3 4 5 6 7 8
Player 1 Turn:
RED , Time left = 00:58
AI moved From (8, 7) To (2, 5), Time Taken: -48 seconds.

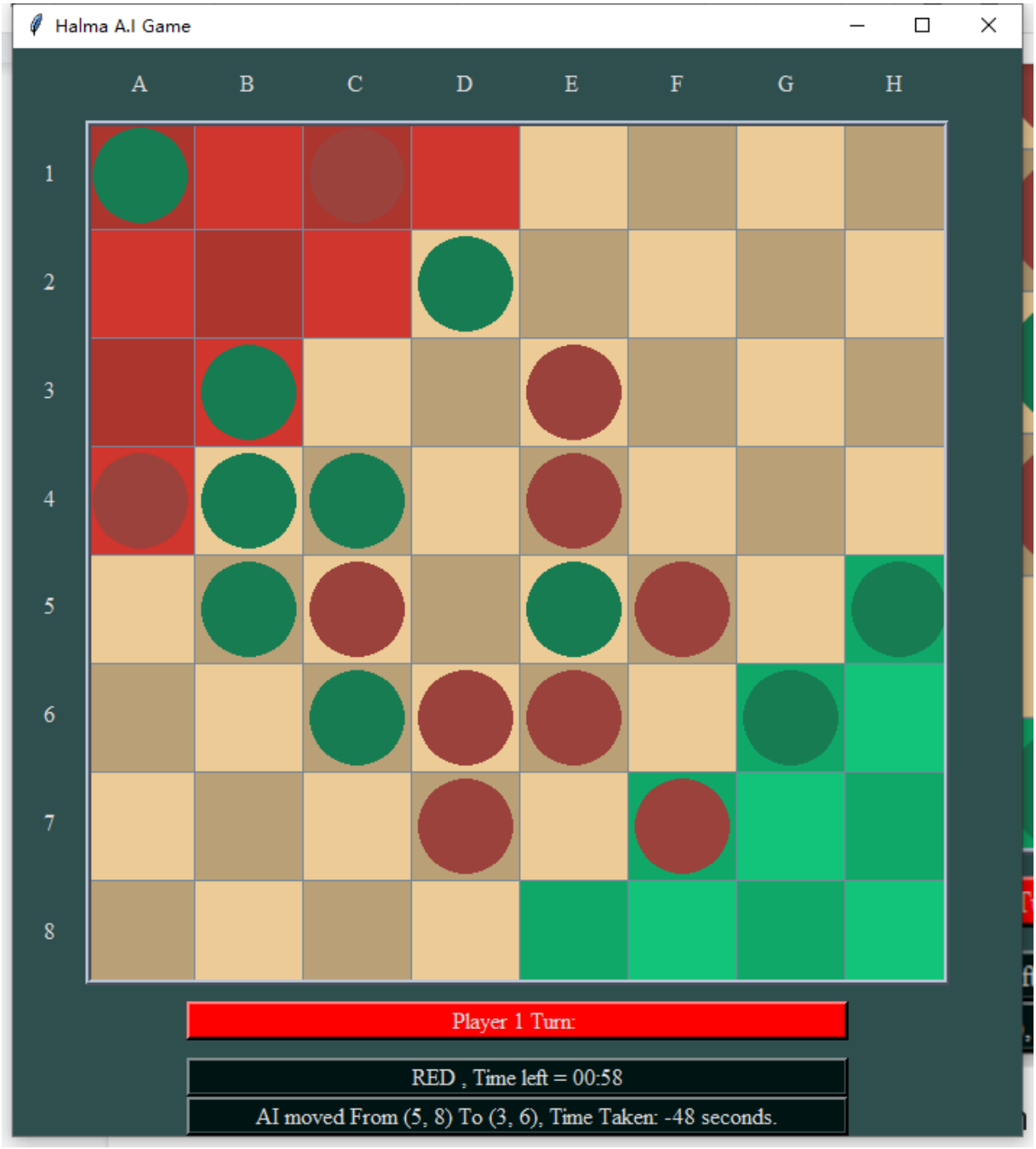

Halma A.I Game
A B C D E F G H
1 2 3 4 5 6 7 8
Player 1 Turn:
RED , Time left = 01:00
AI moved From (6, 6) To (2, 2), Time Taken: -48 seconds.

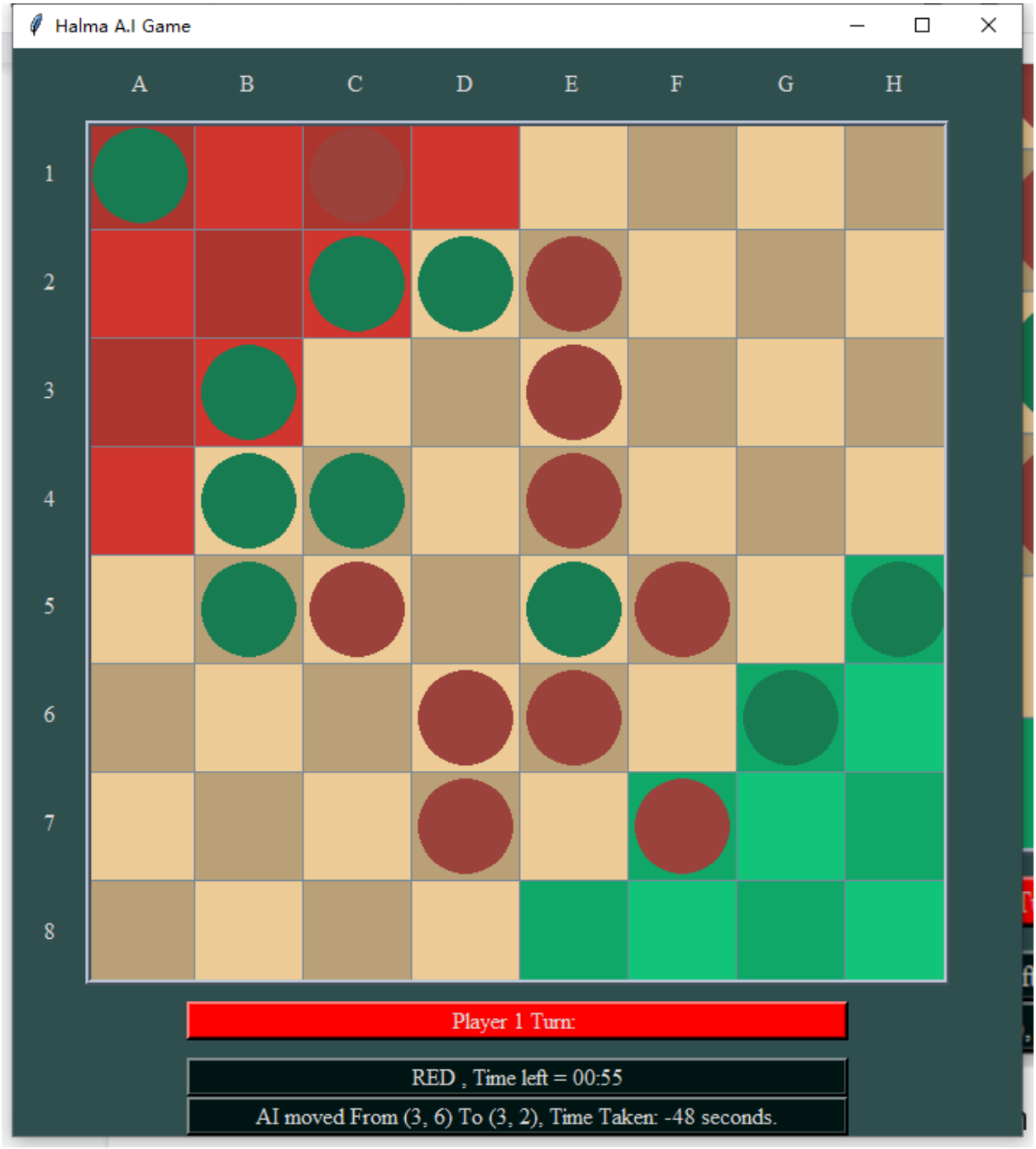

Halma A.I Game
A B C D E F G H
1 2 3 4 5 6 7 8
Player 1 Turn:
RED , Time left = 00:58
AI moved From (6, 8) To (2, 4), Time Taken: -48 seconds.

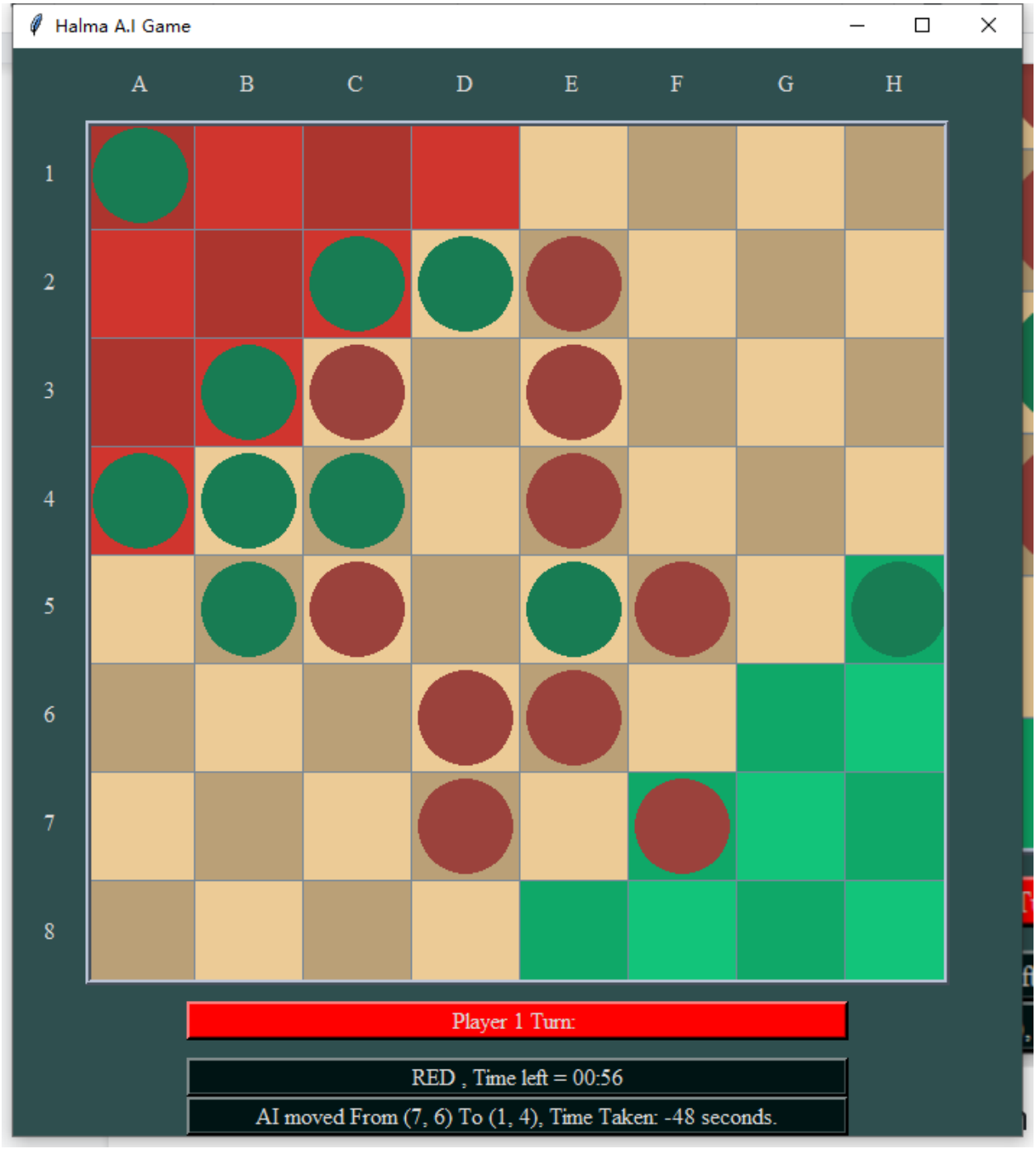

Halma A.I Game
A B C D E F G H
1 2 3 4 5 6 7 8
Player 1 Turn:
RED , Time left = 00:59
AI moved From (8, 6) To (4, 2), Time Taken: -48 seconds.

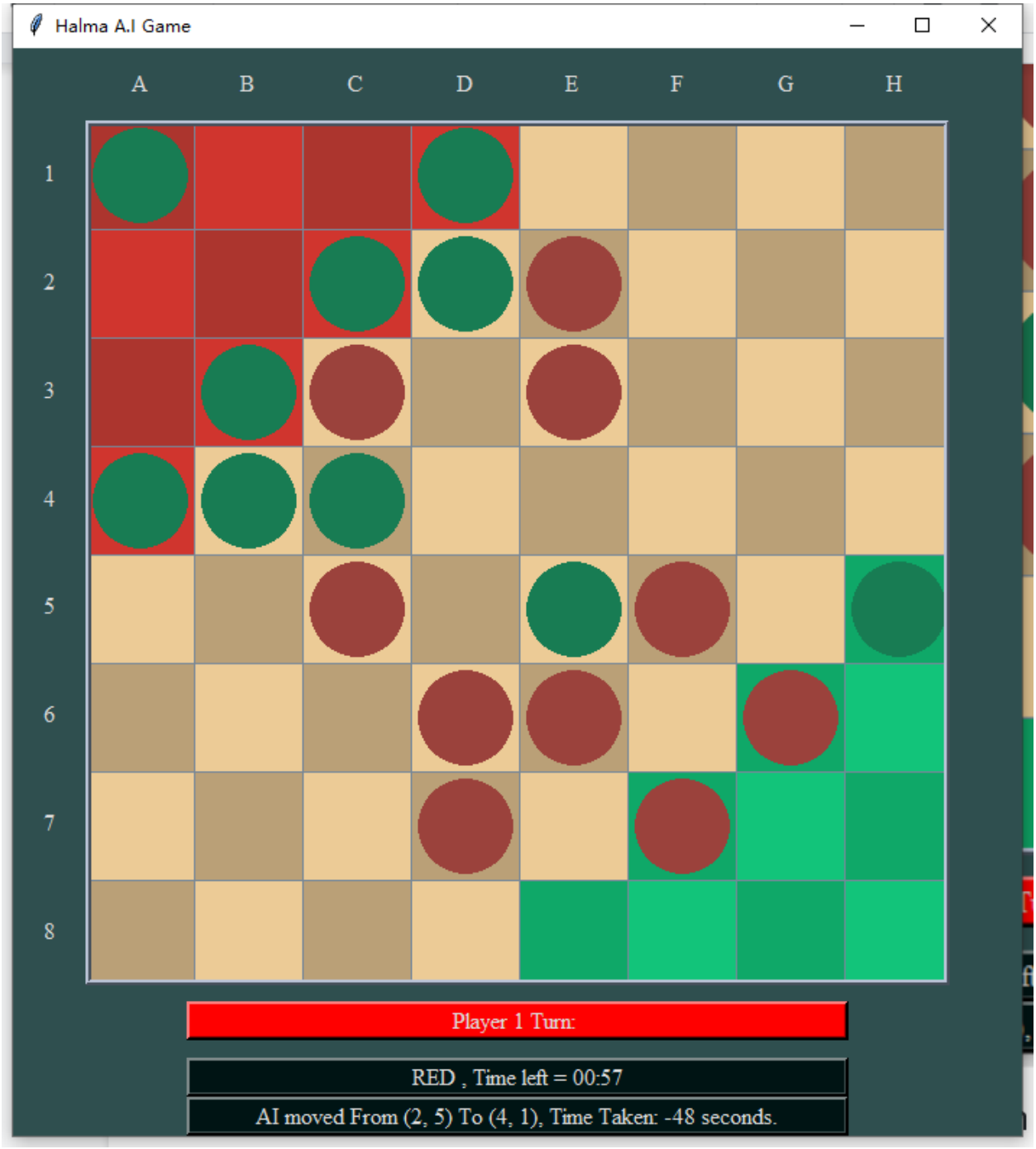

Halma A.I Game
A B C D E F G H
1 2 3 4 5 6 7 8
Player 1 Turn:
RED , Time left = 00:59
AI moved From (6, 7) To (2, 3), Time Taken: -48 seconds.

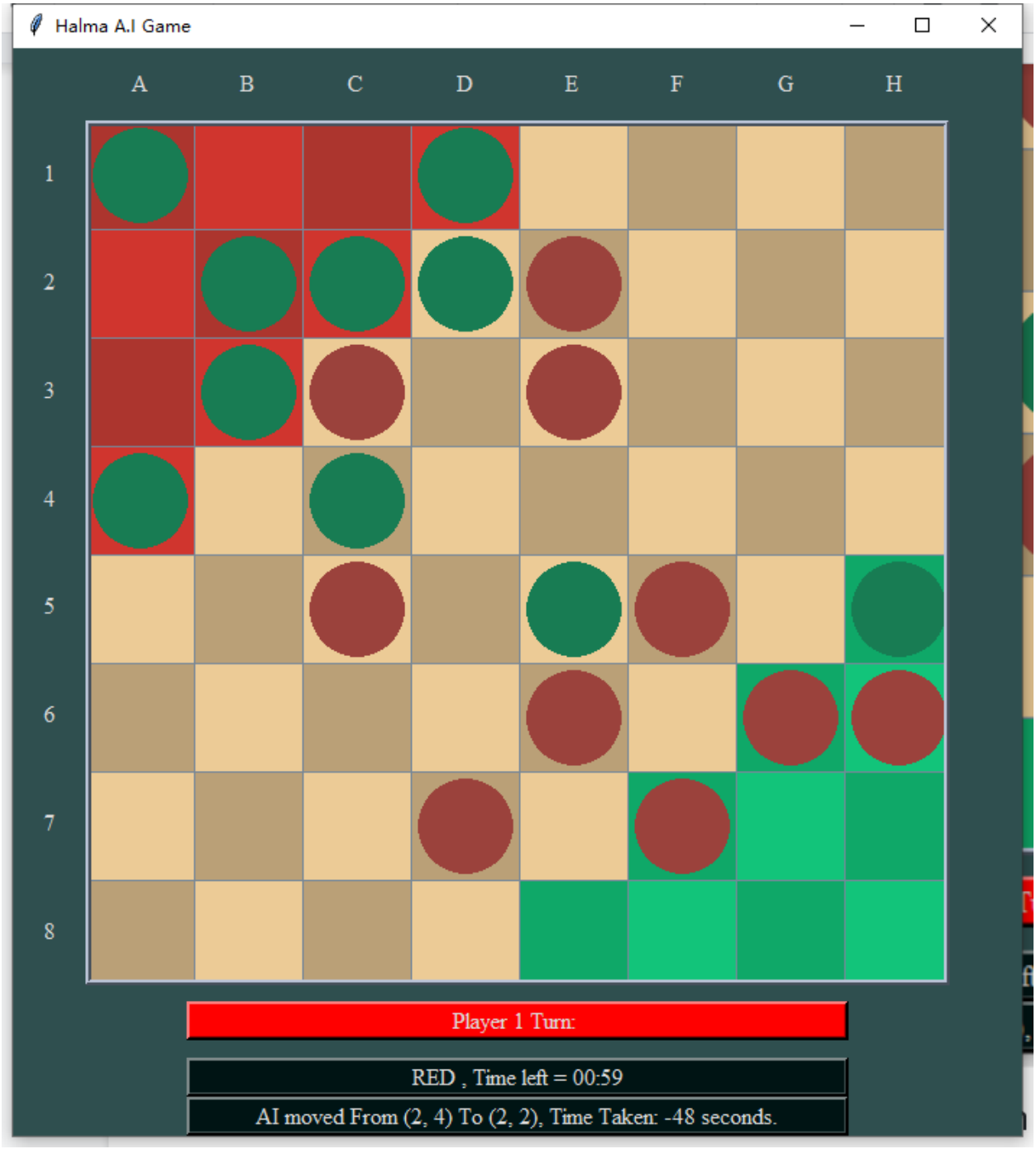

Halma A.I Game
A B C D E F G H
1 2 3 4 5 6 7 8
Player 1 Turn:
RED , Time left = 00:58
AI moved From (2, 2) To (1, 1), Time Taken: -48 seconds.

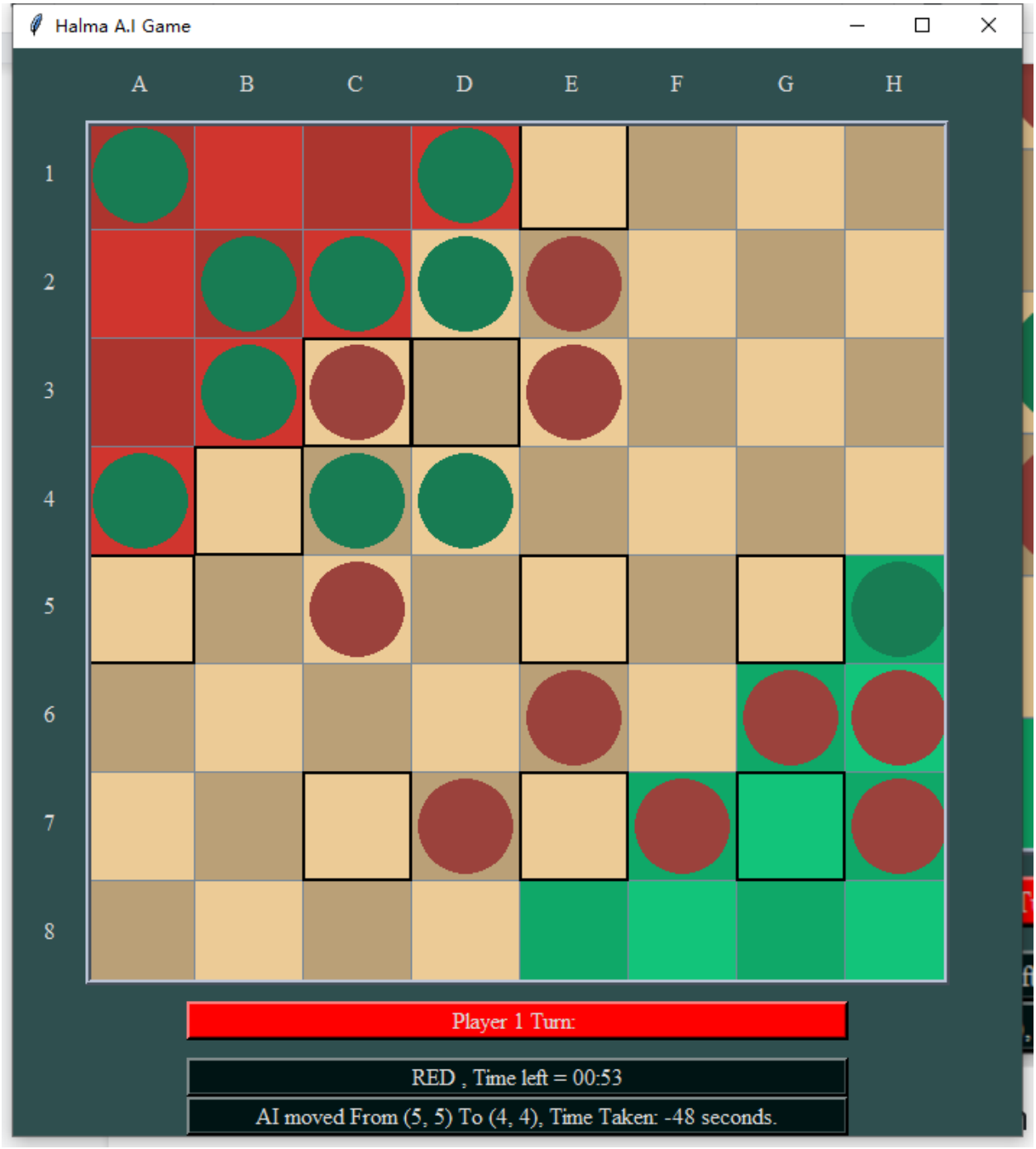

Halma A.I Game
A B C D E F G H
1 2 3 4 5 6 7 8
Player 1 Turn:
RED , Time left = 00:58
AI moved From (5, 8) To (3, 6), Time Taken: -48 seconds.

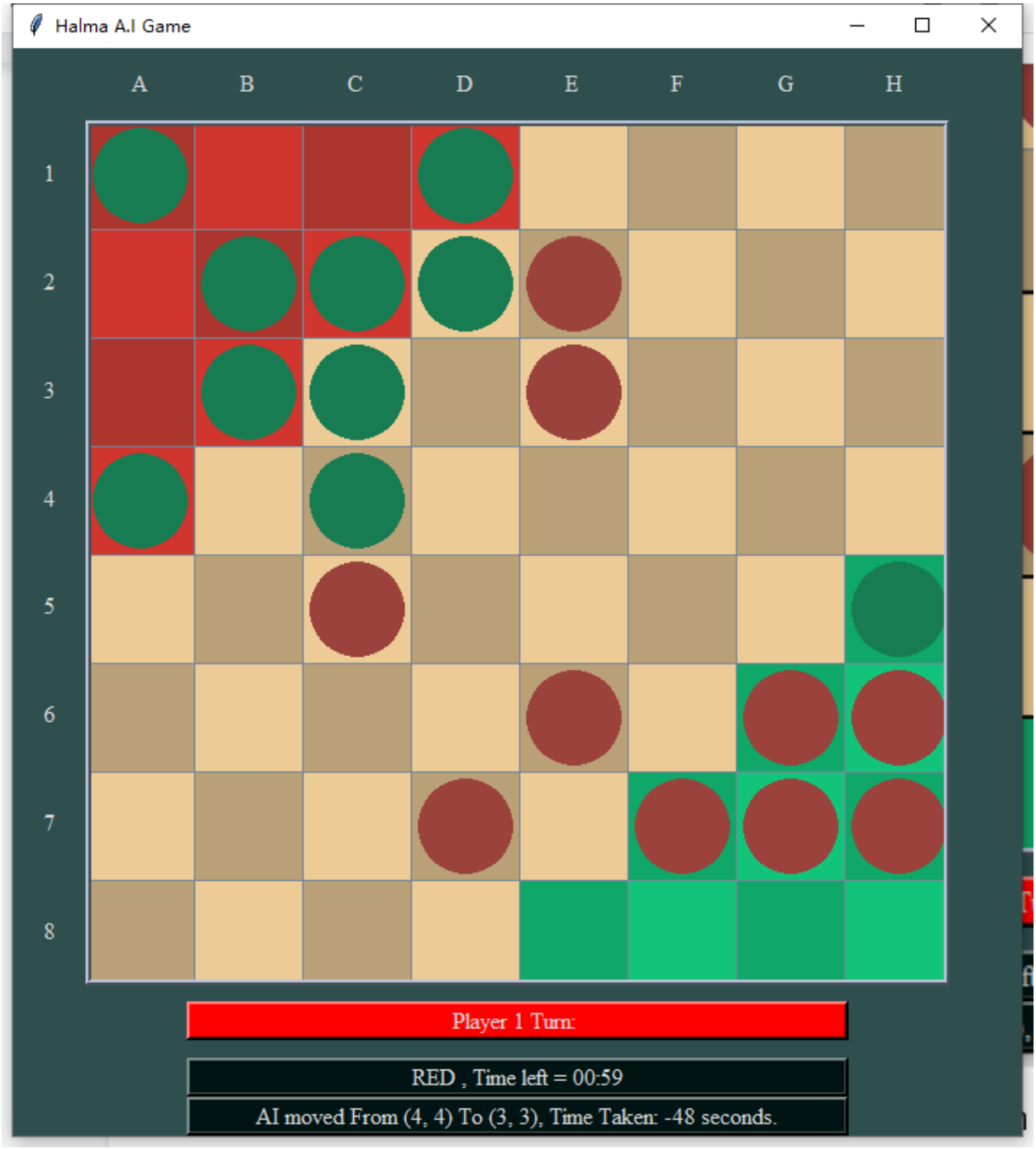

Halma A.I Game
A B C D E F G H
1 2 3 4 5 6 7 8
Player 1 Turn:
RED , Time left = 00:55
AI moved From (3, 6) To (3, 2), Time Taken: -48 seconds.

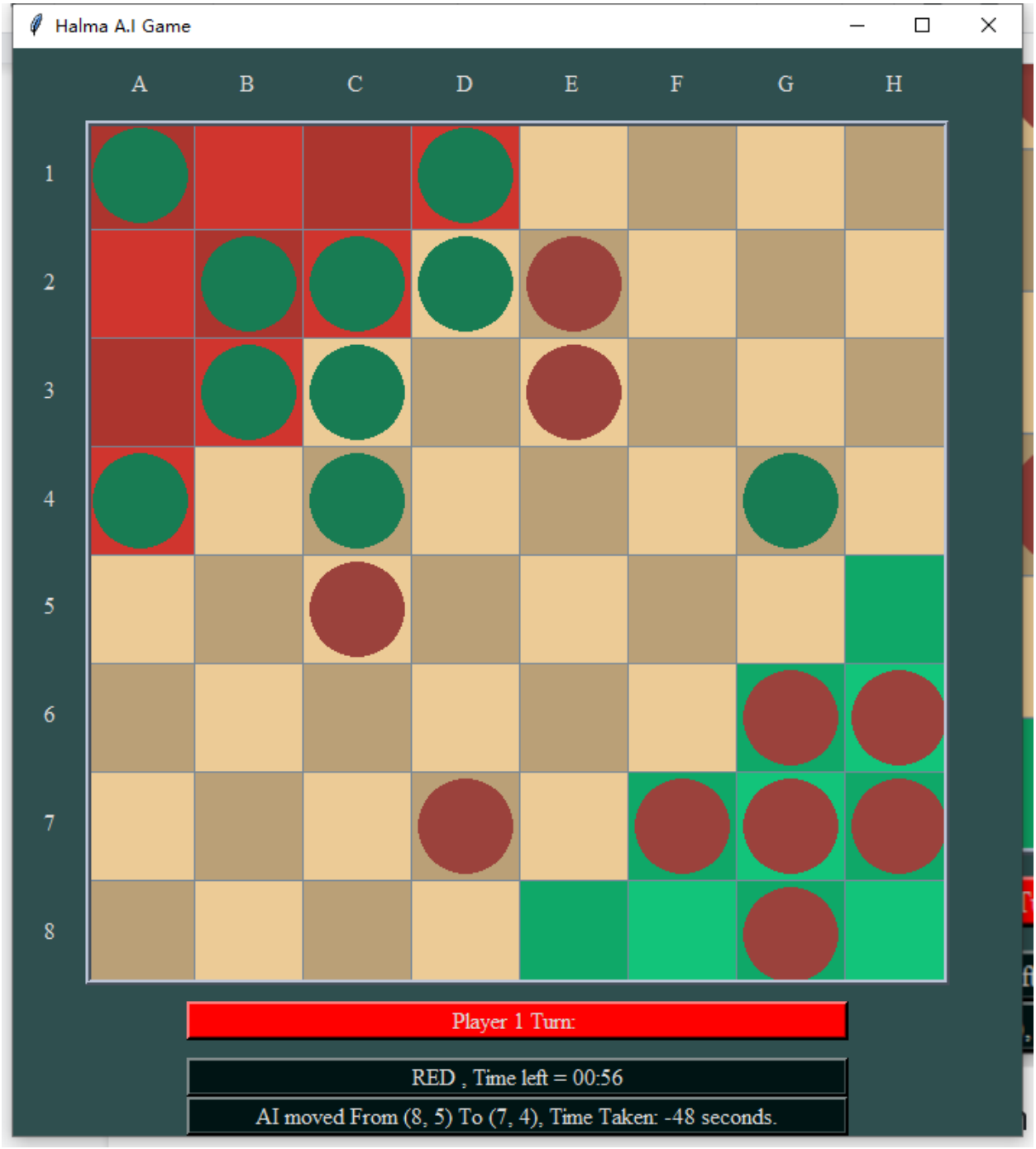

Halma A.I Game
A B C D E F G H
1 2 3 4 5 6 7 8
Player 1 Turn:
RED , Time left = 00:56
AI moved From (7, 6) To (1, 4), Time Taken: -48 seconds.

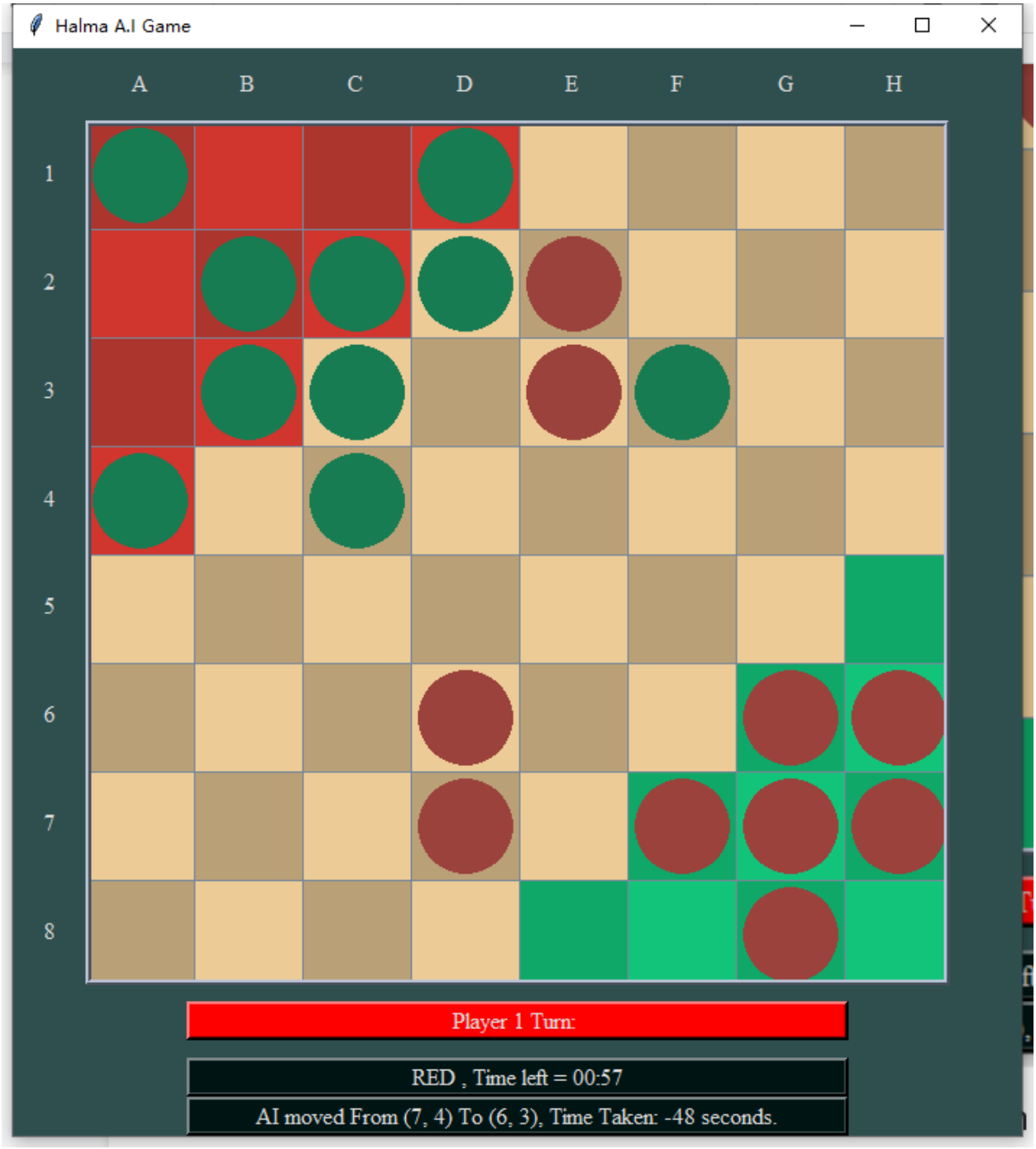

Halma A.I Game
A B C D E F G H
1 2 3 4 5 6 7 8
Player 1 Turn:
RED , Time left = 00:57
AI moved From (2, 5) To (4, 1), Time Taken: -48 seconds.

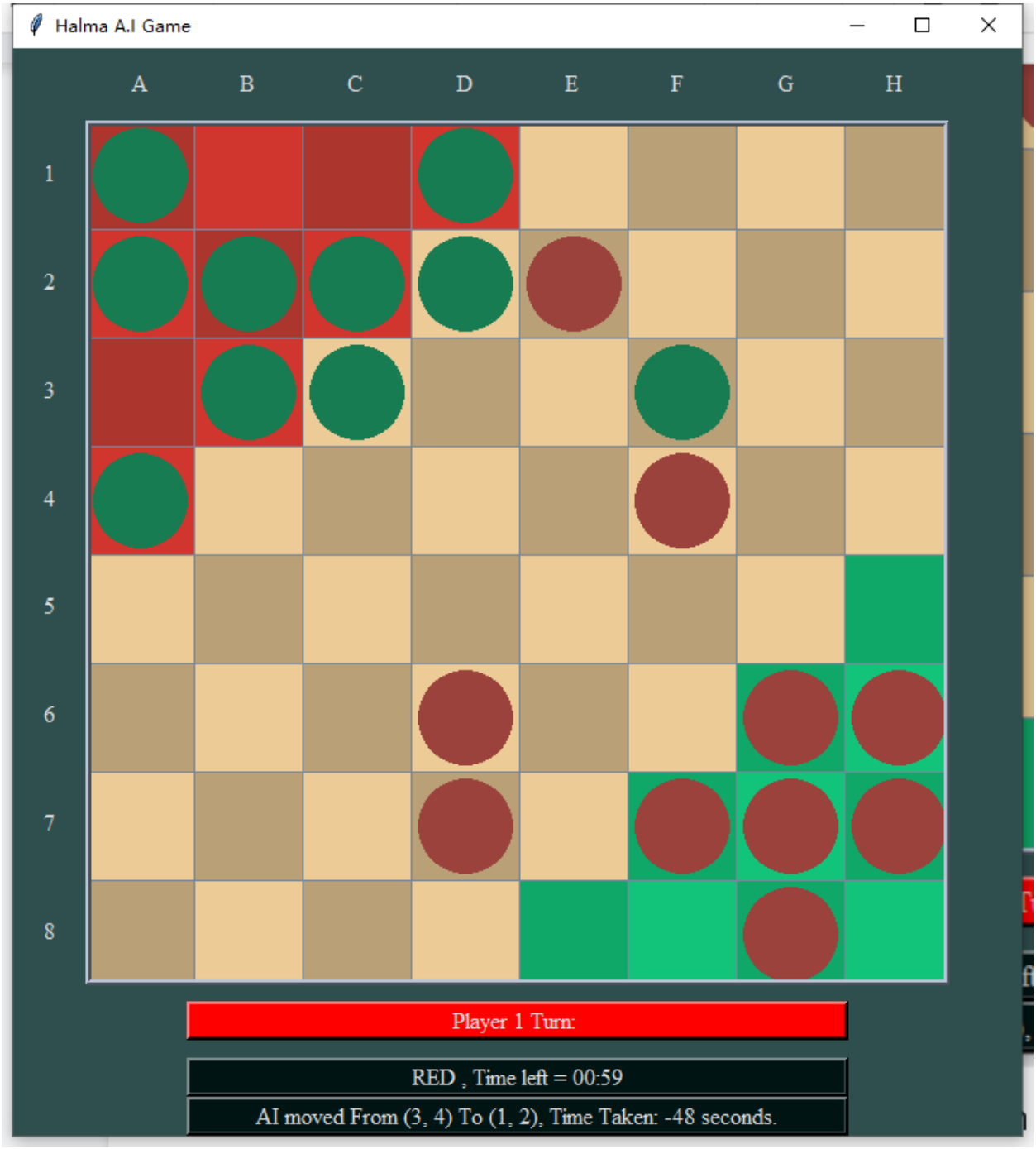

Halma A.I Game
A B C D E F G H
1 2 3 4 5 6 7 8
Player 1 Turn:
RED , Time left = 00:59
AI moved From (2, 4) To (2, 2), Time Taken: -48 seconds.

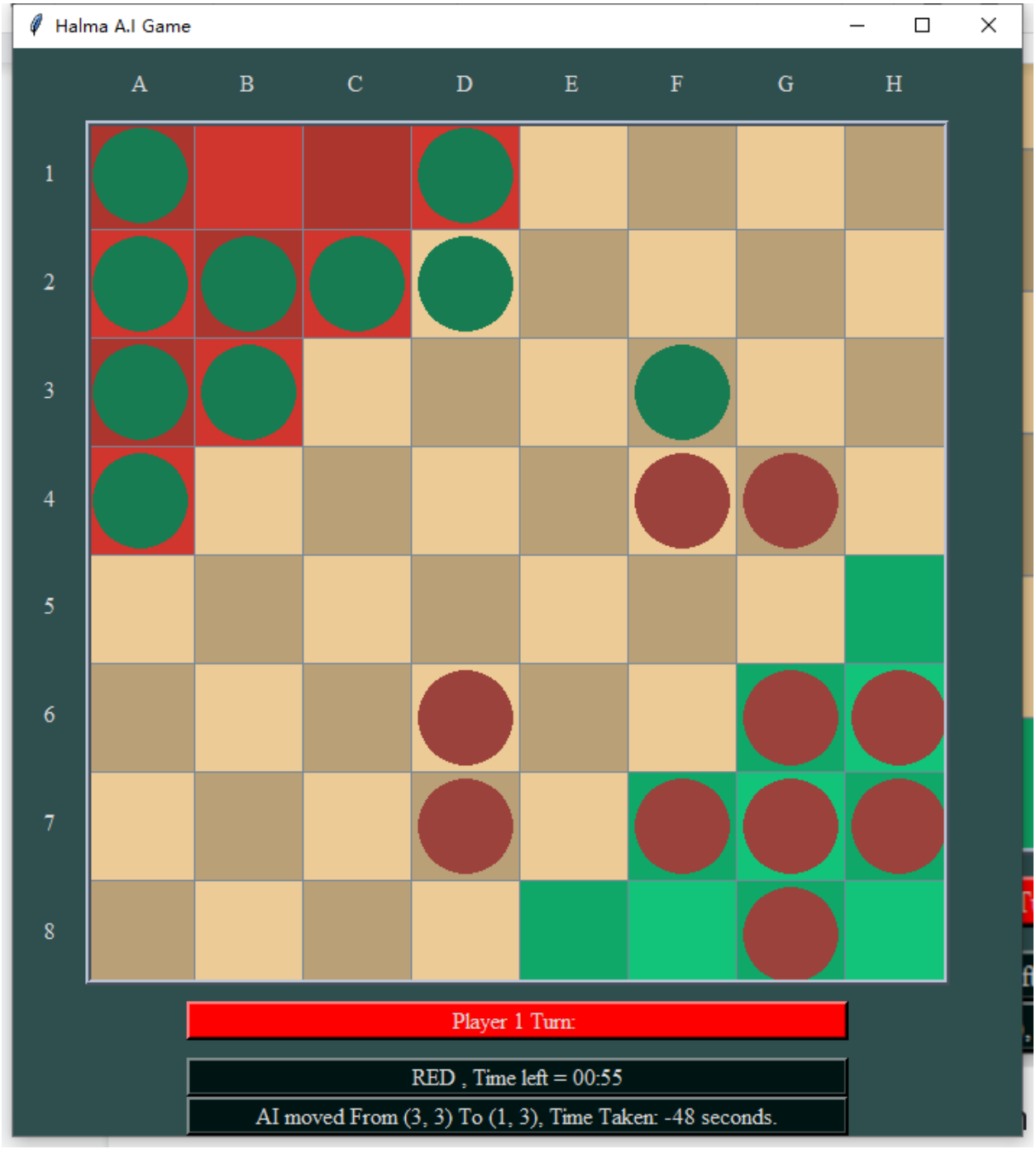

Halma A.I Game
A B C D E F G H
1 2 3 4 5 6 7 8
Player 1 Turn:
RED , Time left = 00:53
AI moved From (5, 5) To (4, 4), Time Taken: -48 seconds.

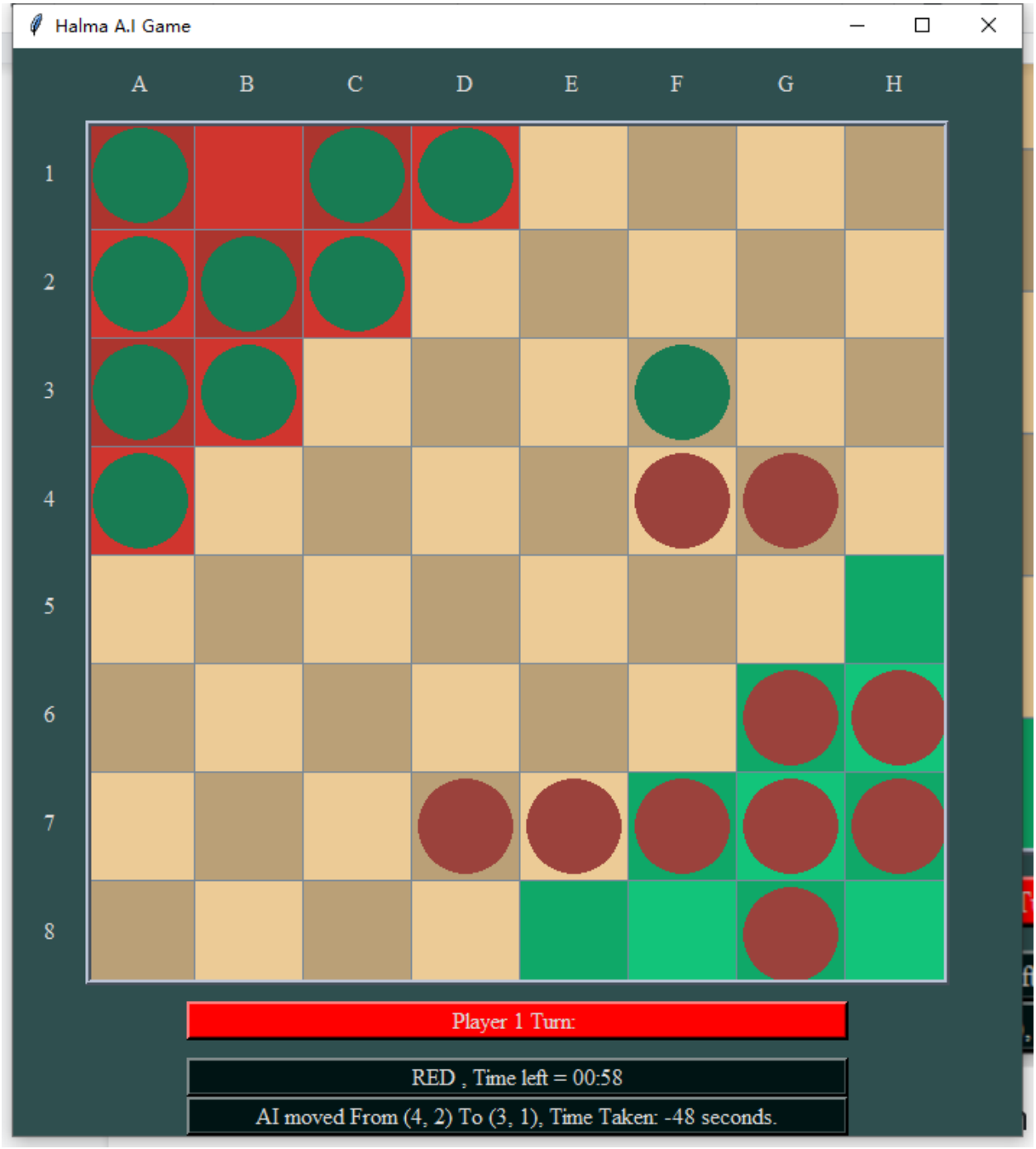

Halma A.I Game
A B C D E F G H
1 2 3 4 5 6 7 8
Player 1 Turn:
RED , Time left = 00:59
AI moved From (4, 4) To (3, 3), Time Taken: -48 seconds.

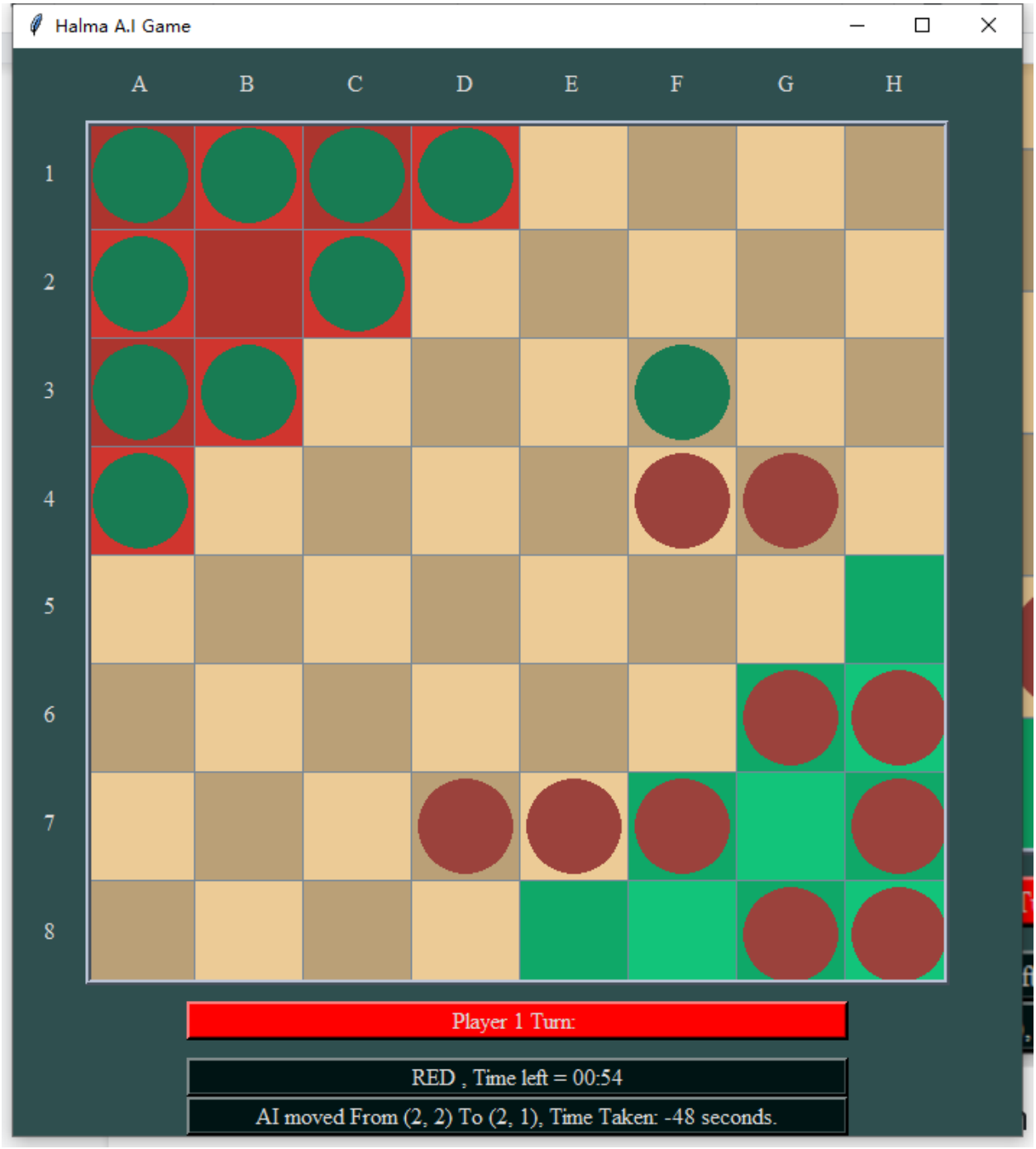

Halma A.I Game
A B C D E F G H
1 2 3 4 5 6 7 8
Player 1 Turn:
RED , Time left = 00:56
AI moved From (8, 5) To (7, 4), Time Taken: -48 seconds.

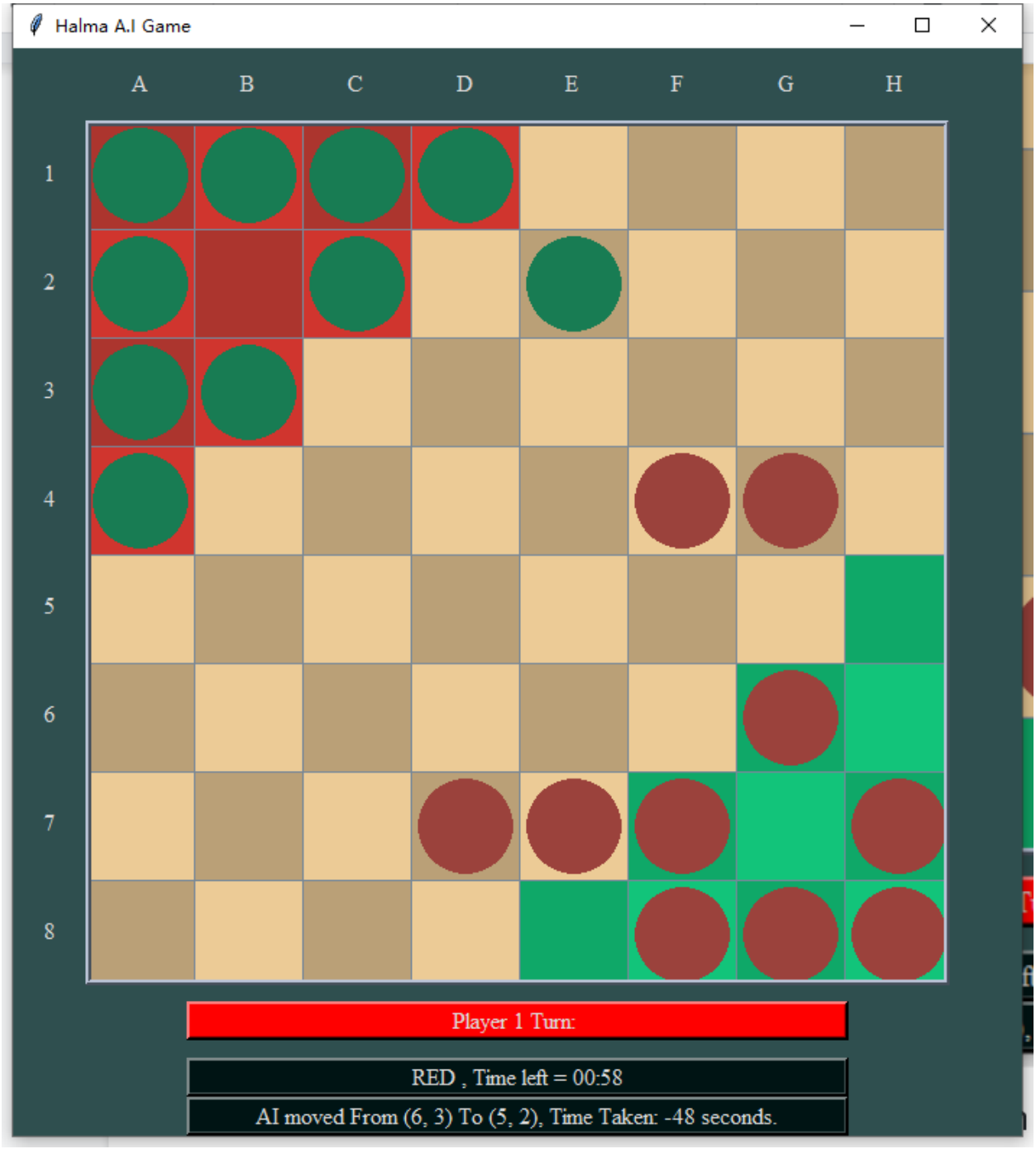

Halma A.I Game
A B C D E F G H
1 2 3 4 5 6 7 8
Player 1 Turn:
RED , Time left = 00:57
AI moved From (7, 4) To (6, 3), Time Taken: -48 seconds.

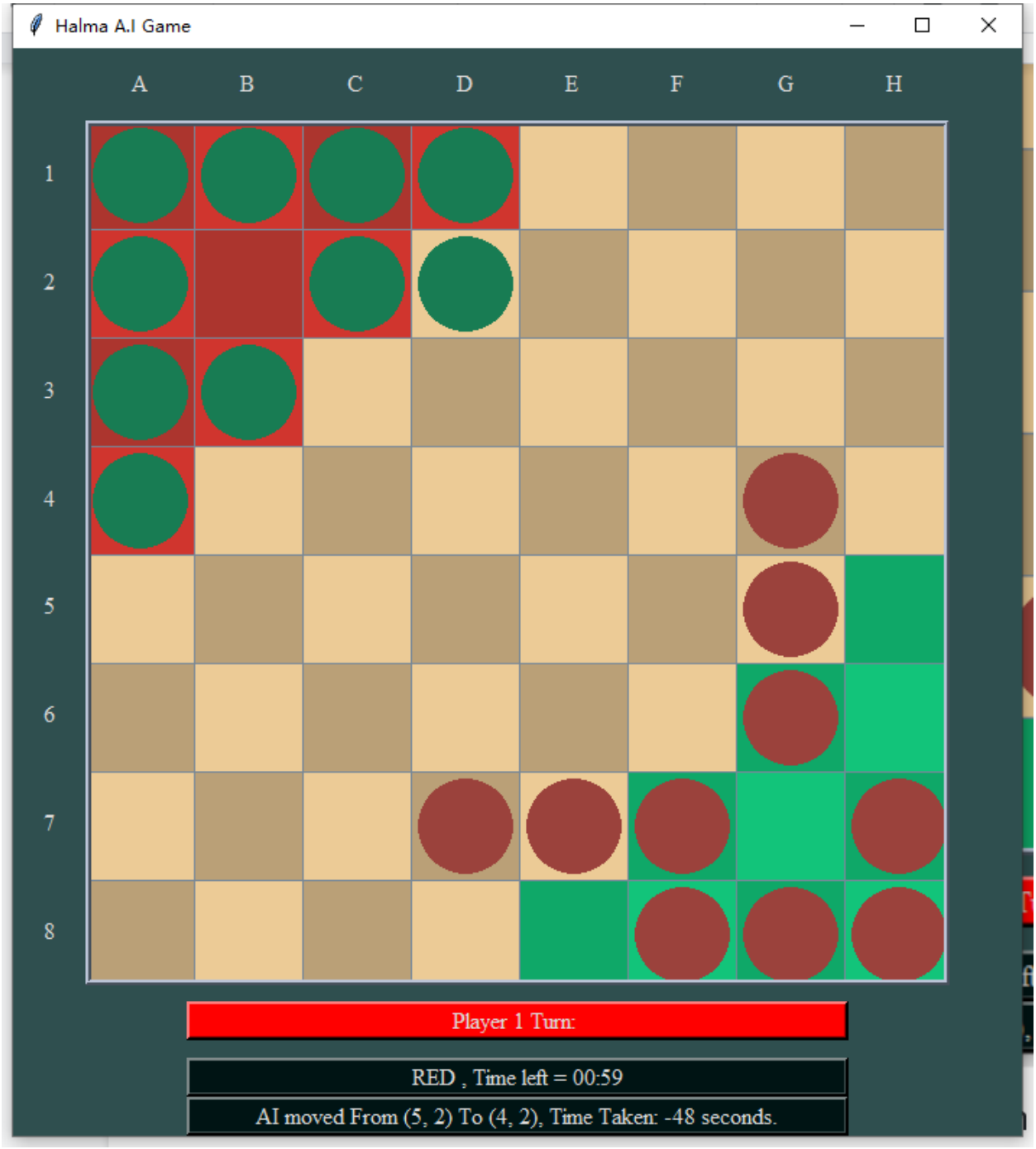

Halma A.I Game
A B C D E F G H
1 2 3 4 5 6 7 8
Player 1 Turn:
RED , Time left = 00:59
AI moved From (3, 4) To (1, 2), Time Taken: -48 seconds.

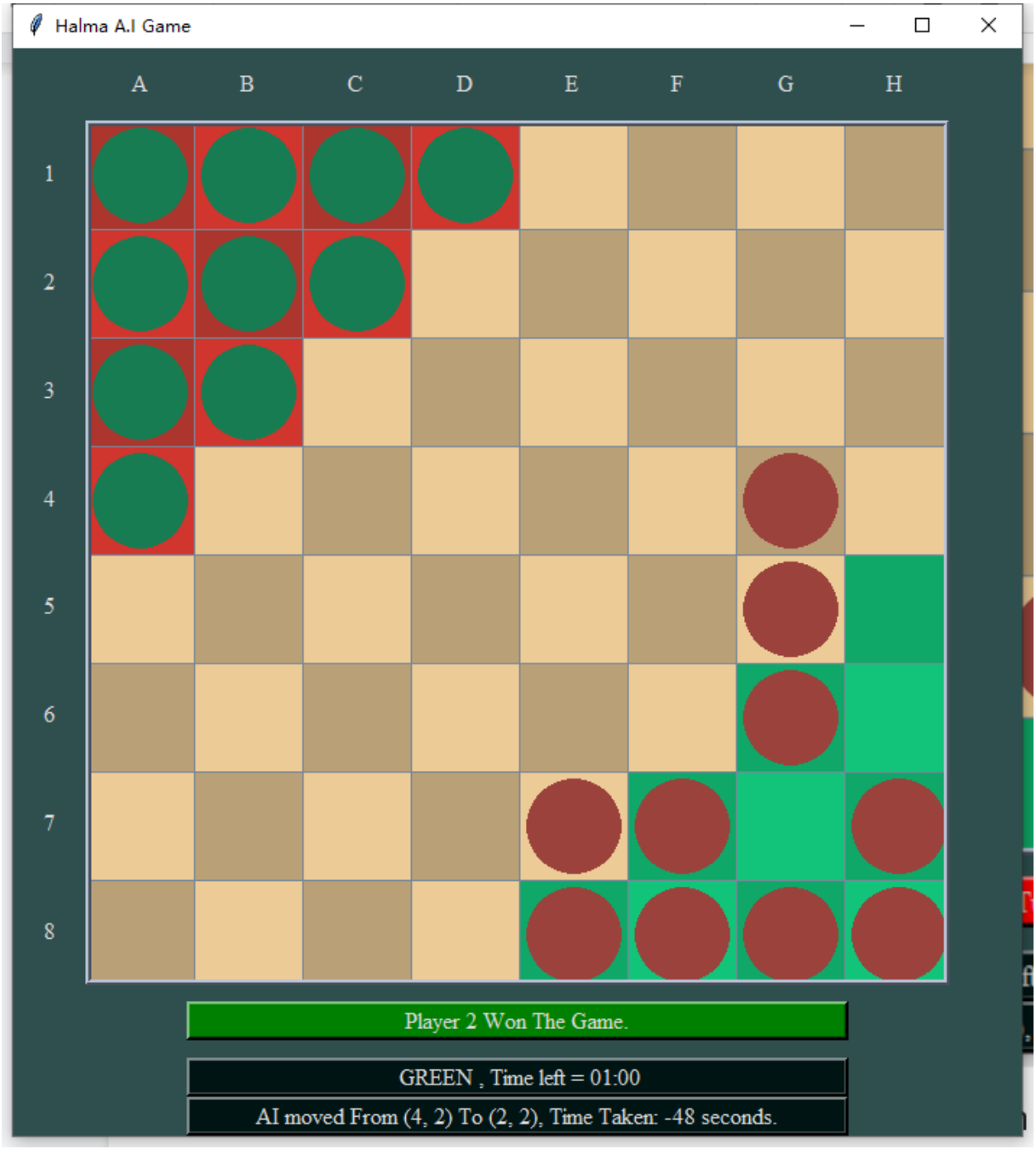

Halma A.I Game
A B C D E F G H
1 2 3 4 5 6 7 8
Player 1 Turn:
RED , Time left = 00:55
AI moved From (3, 3) To (1, 3), Time Taken: -48 seconds.


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 = 00:58
AI moved From (4, 2) To (3, 1), 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 = 00:54
AI moved From (2, 2) To (2, 1), 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 = 00:58
AI moved From (6, 3) To (5, 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 = 00:59
AI moved From (5, 2) To (4, 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 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

```python
import copy

# Charcated Util Class which is used to create
#Objects of players when the game begins.

class CharacterUtil:
  def __init__(self, color, boardSize):
    # Setting each player with its resepective colors
    self.color = color
    self.starposchar(boardSize)
    # Home and Goal Cordinates
    # home cordinate is Goal cordinate for the other player
    #and vice versa
    self.homeCoord = []
    self.goalCoord = []
    self.move = False
  # Functon to get the starting postion of the charcters
  def starposchar(self, boardSize):
    # Creating empty list varibale to track the home
    # Goal and Tokens
    self.charss = []
    self.home = []
    self.goal = []
    # ASsigning Iteration based on the board size.
    if boardSize == 8:
      lenmax = 5
    elif boardSize == 10:
      lenmax = 6
    else: # default is 16 x 16
      lenmax = 7
    # Iterating though our borad to set the home GOal

    for i in range(1, lenmax):
      for j in range(1, lenmax):
        # and the positon of the player. so that we can keep a track later.
        if (i + j <= lenmax and i < lenmax and j < lenmax):
          self.charss.append(Utils(boardSize - j + 1, boardSize - i + 1))
          self.home.append((boardSize - j + 1, boardSize - i + 1))
          self.goal.append((j, i))
    # For debugging PUrpose
    print(self.charss)
    print(self.home)
    print(self.goal)


    if self.color == 'RED':
      # The other Player also have similar Cordinates the
      # only change is that
      temp = copy.deepcopy(self.home)
      self.home = copy.deepcopy(self.goal)
      # The goal cordinate for player 1 is Home Corindate of player 2

      self.goal = temp
      # and the Home Cordinate of player  1 is the Goal cordinate of player 2.
      for i in range(len(self.charss)):
        self.charss[i].x = self.home[i][0]
        self.charss[i].y = self.home[i][1]
```

```
                self.charss[i].y = self.home[i][1]


    # FUction to Check that the game is FInised Or not
    # if all the players reached oppositon Home
    # then the player wins the game.
    def all_char_opp(self):
        status = True
        for p in self.charss:
            if (p.x, p.y) in self.goal:
                status = True
            else:
                status = False
                break
        return status

    # Fucntion to Chekc the location of the chratcer
    def charlocCheck(self,x,y):
    # if the charcater is in X and Y cordinate it returns True
        for i in range(len(self.charss)):
            if(self.charss[i].x == x and self.charss[i].y== y ):
                return True
                break
            else:
    # Else It return false
                pass
        return False
    # FUcntion To check the Home Cordinate of the Chracter.
    def charHomeCheck(self,x,y):
        koor =(x, y)
        # If Givenn Cordinate is in our HOme list that we alredy Trakced then it return true
        if koor in self.home:
            return True
        else:
            return False

    #Function to Check the goal cordinate
    def charGoalCheck(self,x,y):
        koor =(x, y)
        # If given Cord is Goal cord then true else false
        if koor in self.goal:
            return True
        else:
            return False
    # FUnction to get the Charatcer
    #Token from row and Column
    def getChar(self, row, column):
        i = 0
        found = False
        # If a chracter token is found withing our row and Columns

        while i < len(self.charss) and not(found):
            # it returns the chratcers if it is avalible
```

```
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 cordinate of tile to the goal tile and Move the token
124         # temporarirly
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     # Fucntion to move the chracter from one Grid Locatio
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 though our Tokens
139       for p in self.charss:
140         # if its x cordinates matches with the current cordinate
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 Cordinate
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 Cordinates
160       self.setCoordinate(x, y)
161       #Print("CHAR INISITALIZE")
162       self.setChar_Gone(False)
163       self.setChar_arr(False)
164     # TO set the cordinates 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 cordinates of Player
175     def getCoordinateX(self):
176         return (self.x)
177     # TO get Y cordinates
178     def getCoordinateY(self):
179         return (self.y)
180 
181     def getCoordinate(self):
182         return (self.x, self.y)
```

Utilities.py

```
1   #Utilities Class That helps in various stage to get the data
2   #related to our aGame
3   class Utilities:
4       # Initializing Constructer 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 cordinates of the player.
18      def printCoordinate(self):
19          print(str(self.x) + str(self.y) + self.color + str(self.character))
```

Game.py

game.py > ...

```python
#importing other package
from Utilities import Utilities
import math
import time
#Our Game Class which handels and Genrate moves in the back
#It is also responsible to generate Jumps while making moves.
class Game:
  #initializing our construter of the game.
  def __init__(self, boardSize, timeLimit, p1, p2, human,bot,alphabeta):
    #assiging all the parametres essential to play the game
    self.boardSize = boardSize
    self.timelimit = timeLimit
    self.player1 = p1
    self.player2 = p2
    self.alphabeta=alphabeta
    #assiginng therir respective color to the players
    self.p1cop = p1 if p1.color == "GREEN" else p2
    self.p2cop = p2 if p2.color == "RED" else p1
    #assigning turn to a player
    self.turn = 1
    self.bot=bot
    #getting the cordianes of our board
    self.coordinate = [[Utilities(i, j) for i in range(self.boardSize)] for j in range(self.boardSize)]
    self.deep = 2
    #humans as both players are playing the game.
    self.human = human
    self.ai_move=None

    #Assigning maximum iterations according to our board size.
    if self.boardSize == 8:
      lenmax = 4
    elif self.boardSize == 10:
      lenmax = 5
    else:
      lenmax = 6
#looping through all the iterations.
    for i in range(lenmax):
      for j in range(lenmax):
        if (i + j < lenmax and i < 6 and j < 6):
          #Making red player home  cordinate red and
          self.coordinate[i][j].color = "RED"
          self.coordinate[i][j].character = 2
          #making green player home cordinates green
          #Appending all the home and goal cordinates of
          #player 1 and Player 2

          self.p2cop.homeCoord.append(self.coordinate[i][j])
          self.p1cop.goalCoord.append(self.coordinate[i][j])
          #similarly for Player 2 Assigning COlor and Home goal cordinates
          self.coordinate[self.boardSize - 1 - i][self.boardSize - 1 - j].color = "GREEN"
          self.coordinate[self.boardSize - 1 - i][self.boardSize - 1 - j].character = 1
          self.p1cop.goalCoord.append(self.coordinate[self.boardSize - 1 - i][self.boardSize - 1 - j])
          self.p2cop.homeCoord.append(self.coordinate[self.boardSize - 1 - i][self.boardSize - 1 - j])


```

```
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 Cordinates of the player
69      def checkhomecord(self, player, x, y):
70          return(player.charHomeCheck(x,y))
71      #Function to check the goal cordinates 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 avaiable 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))
00              if (not(self.checkboxemp(x,y-1)) and (self.checkboxemp(x, y-2))):
01                  list_of_positions.append((x, y-2))
02              if (not(self.checkboxemp(x+1,y+1)) and (self.checkboxemp(x+2, y+2))):
03                  list_of_positions.append((x+2, y+2))
04              if (not(self.checkboxemp(x+1,y-1)) and (self.checkboxemp(x+2, y-2))):
05                  list_of_positions.append((x+2, y-2))
06              if (not(self.checkboxemp(x-1,y+1)) and (self.checkboxemp(x-2, y+2))):
07                  list_of_positions.append((x-2, y+2))
08              if (not(self.checkboxemp(x-1,y-1)) and (self.checkboxemp(x-2, y-2))):
09                  list_of_positions.append((x-2, y-2))
10
```

```
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 plyer 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 alll 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         # same concept a above using List of Moves
189         length = len(list_of_positions)
190         print("Length",length)
191         i = 0
192
193         while (i < length):
194           (x, y) = list_of_positions[i]
195           print("X,y," ,list_of_positions[i])
196           print("PLYER WHILE",player)
197           #checking all the home and goal cordinates of a player if it is not then we removed that positon from
198           # the list of positos
199           if (character.char_arr and not(self.checkgoalcord(player, x, y))) or (character.char_gone and (self.checkhomecord(player, x ,y))):
200             list_of_positions.remove(list_of_positions[i])
201             length -= 1
202           else:
203             i += 1
204         # once we have alist we sorted it
205         list_of_positions = sorted(list_of_positions, key=lambda tup: (tup[0], tup[1]))
206         return list_of_positions
207
208
209       # fucntion in backend to move chracter from one player to another.
210       def char_movement(self, xpos, ypos):
211         #getting from cordinate
212
213         gridfrom = self.coordinate[xpos[0]-1][xpos[1]-1]
214         #getting the cordinate where we have to move the player
215         gridto = self.coordinate[ypos[0]-1][ypos[1]-1]
216         # Chekcing that the move is valid or not
217
218         if gridfrom.character == 0 or gridto.character != 0:
```

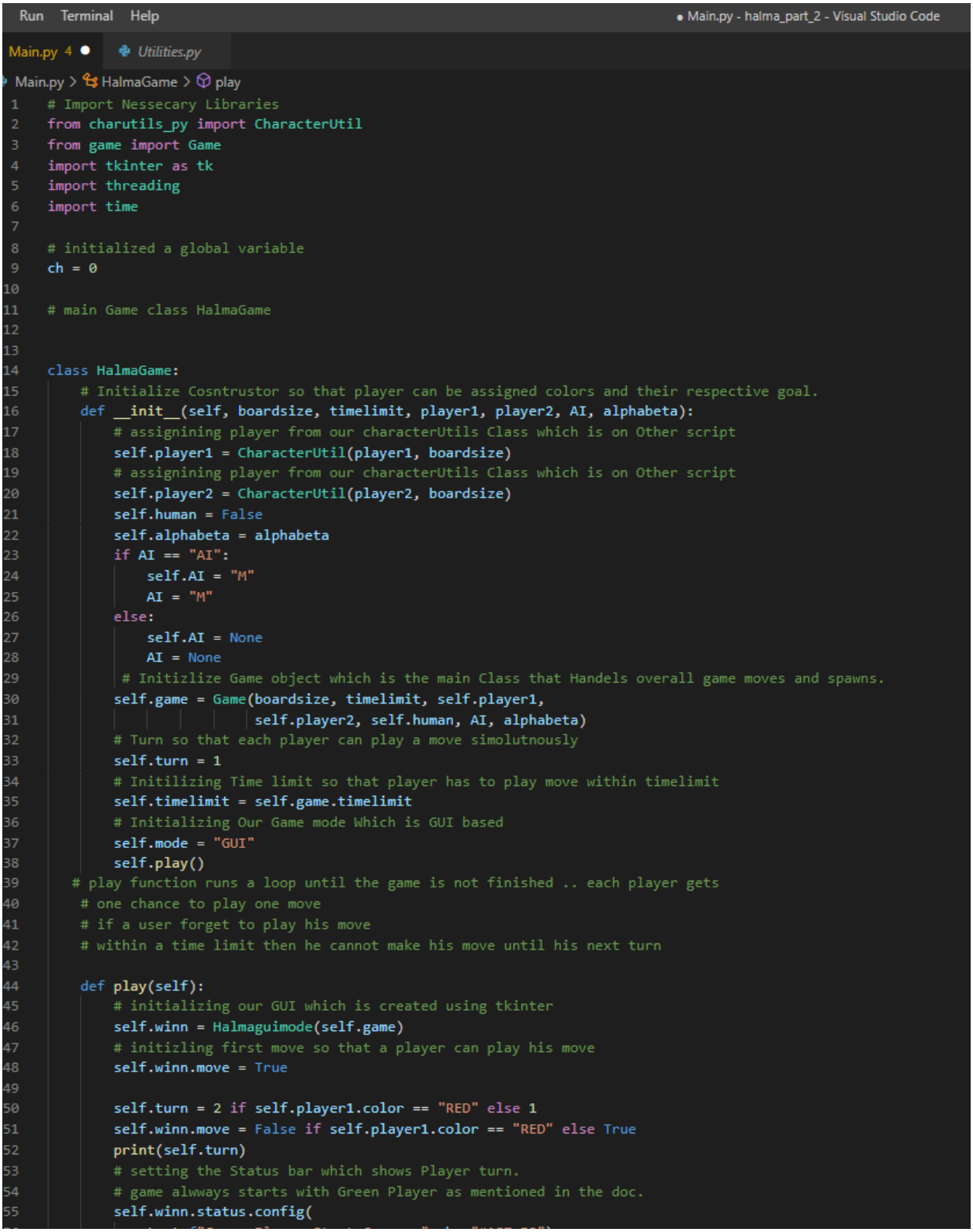
```
218         if gridfrom.character == 0 or gridto.character != 0:
219             print("Invalidddddddddddddddd")
220             return
221         # only a valid move can be played
222         # onces a move is validated
223         if gridfrom.character == 1:
224             self.p1cop.char_movement((gridfrom.x+1, 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 calulate the distance between current and the goal cordinates
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 calulate 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 cordinates 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 calcualte 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]
```

```
                                self.coordinate[y - 1][x - 1].character != 2]
            # Similarly as above we calculate the distiance to the goal and choose the maxmium vale
            temp_val += max(dist_to_goal) if len(dist_to_goal) else -50
        temp_val *= hold
        return temp_val

        # This funciton takes all the valid moves
        # and return the cordinates of every valid moves
    def CharTurnCord(self, all_valids):
        moves = []
        for i in all_valids:
            # looping through all the valid moves and returning their
            # cordinates
            val_turn = self.coordinate[i[1] - 1][i[0] - 1]
            moves.append(val_turn)

        return moves
#This is our main AI algorithm that will play the game
    # it starts with the depth that we provide currently it is
    # 2 so that ti can explore the moves
    def minimax(self, deep,  playermin, playermax, MAX=True, alp=float("-inf"), bet=float("inf")):
        # basis
        print("deep", deep)
        # if the depth is 0  we explored then we return the moves of max player
        if deep == 0 :
            return self.calculategoal(playermax), None
# with the help of recursion we calclate the best move to the goal node
        cur_best = None
        # we start with geting all the valid moves so that we have a udnerstanding where we can move
        if MAX:
            cur_best_val = float("-inf")
            All_Valid_moves = self.Player_Char_moves(playermax)

        else:
            cur_best_val = float("inf")
            All_Valid_moves = self.Player_Char_moves(playermin)
    # Once we have all the valid moves that we can play
    #    we loop thorugh all the moves
    #    and using a tempmovement we calcualte all the moves

        for turn in All_Valid_moves:
            for grid_to in turn["grid_to"]:
                # so that in the end Ai has a best move to play
                self.Char_Move_Temp((turn["grid_from"].y + 1, turn["grid_from"].x + 1), (grid_to.y + 1, grid_to.x + 1))
        # Ai use a Temporary character movement function
                # which is just a copy of
                # our char_movement function
            # USing the recursion technique we explore the game again by subtracting the depth  by
                #-1
                #that is depth=-1
                temp_val, unk = self.minimax(deep - 1,playermin,playermax,not MAX, alp, bet,)
                # Temporary moved our Tokenns
                self.Char_Move_Temp((grid_to.y + 1, grid_to.x + 1), (turn["grid_from"].y + 1, turn["grid_from"].x + 1))
                # This is the main part
```

```
            # This is the main part
            # that is used to get the best move
            if MAX and temp_val > cur_best_val:
              #if our new best move is better then the old best then it is assined as new best move
              cur_best_val = temp_val
              # So that we always have a current best move out of all
              cur_best = ((turn["grid_from"].y + 1, turn["grid_from"].x + 1), (grid_to.y + 1, grid_to.x + 1))

              alp = max(alp, temp_val)
              # we get the maximum of that so that we can use it in alpha beta purning later
                #if alpha beta is enabled

            if not MAX and temp_val < cur_best_val:
              # if we didnt find the best move in the Max depth
              # then we explore more and get the best move as possinble
              cur_best_val = temp_val
              cur_best = ((turn["grid_from"].y + 1, turn["grid_from"].x + 1), (grid_to.y + 1, grid_to.x + 1))
              # calculating the beta to later use in alpha beta purning
              bet = min(bet, temp_val)
            # alpha beta pruning
            # this is only a optimization for our minimax AI Algortithm
            if self.alphabeta=="ON":
              #user can turn it on and off when user stars the game
              if  alp >=bet:
                print("AlphaBetaaaaaaaaaaaaaaa")
                return cur_best_val, cur_best
        #returning our current best move that we got using the minimax recusrion
        return cur_best_val, cur_best

    # this funciton is used to get the current move and all the possible moves that a token can be
      # moved to played
      def Player_Char_moves(self, player):
        moves = []  # All possible moves
        # everytime thsis function is called it will get the current tile locationa nd
        # calculate all the possible moves then minimax get the best move
        for p in player.charss:
          curr_tile = self.coordinate[p.y - 1][p.x - 1]
          turn = {
            "grid_from": curr_tile,
            "grid_to": self.CharTurnCord(self.validatemove(p))
          }
          # Returning all the valid moves form the Current tile
          moves.append(turn)
        return moves


    # This is our temporay token movement function
      # which is a copy of our char movemnt function
      def Char_Move_Temp(self, xpos, ypos):
        # we use this fucntion  to calculate the temp move while searchin the best move for
        # our AI
        grid_from = self.coordinate[xpos[0] - 1][xpos[1] - 1]
        grid_to = self.coordinate[ypos[0] - 1][ypos[1] - 1]
        # IT will work similardy by getting the cordinates of the tiles
        # and whoever AI is assignesd it makes a temp move.
        if grid_from.character == 1:
          self.p1cop.Char_Move_Temp((grid_from.x + 1, grid_from.y + 1), (grid_to.x + 1, grid_to.y + 1))
```

```
382             self.p1cop.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 Cosntrustor so that player can be assigned colors and their respective goal.
16 def __init__(self, boardsize, timelimit, player1, player2, AI, alphabeta):
17 # assignining player from our characterUtils Class which is on Other script
18 self.player1 = CharacterUtil(player1, boardsize)
19 # assignining 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 simolutnously
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(

```
        # game alwways starts with Green Player as mentioned in the doc.
        self.winn.status.config(
            text=f"Green Player Start Game...", bg="#187c53")
        self.winn.Info.config(
            text=f"Player1: {self.player1.color} , Player2: {self.player2.color}, Alpha Beta Purning: {self.alphabeta}")
        print("here")
        self.winn.t = self.timelimit
        self.winn.backup = self.timelimit
        # creating a threded object so that we can also monitor the time limit.
        p1 = threading.Thread(target=self.winn.timeren)
        p1.start()
        checko = 0
        # looping until a player wons the game
        while (self.endgame() == True):
            # endgame checks that a player has won or not.
            # a player won if his all the spawns are in opponent home.
            # which is goal in player case
            if (self.turn == 2 and not (self.winn.move)):
                print("PLAYER@@@@@")
                # playing Move and setting the status bar for player
                if self.AI != None:
                    self.winn.timer3.config(text=f" AI is Thinking ")
                self.winn.status.config(
                    text="Player 2 Turn: ", bg=self.player2.color)

                self.winn.makechar()
    # WE modified our last code that
                # is human vs human to Human vs AI in this section.
                # if Human wants to play with AI
                if self.AI != None:
                    self.winn.timer3.config(text=f" AI is Thinking ")
                # we used a thread obeject so that AI can calucate its best move beside while
                    # we run a timer in other thread to keep track of the Timer of the AI
                    # if AI failed to move within the time range then AI turn will be skiped
                    game = threading.Thread(target=self.game.AiPlay)

                    self.winn.timer3.config(text=f" AI is Thinking ")
                    game.start()
                    # Once ai player his turn then turn will be assigned to other player
                    while self.player2.move == False:
                        times = self.winn.t
                        # self.winn.timer3.config(text=f" {self.player2.color} , Time left = {self.winn.t}")
                        # self.winn.timer3.config(text=f" {self.player2.color} , Time left = {self.winn.t}")
                        print("YO", self.winn.t)

                        self.winn.timer3.config(text=f" AI is Thinking ")

                        if p1.isAlive():
                            #                                       print("ALVE")
                            self.winn.timer3.config(
                                text=f" {self.player2.color} , Time left = {self.winn.t} ")

                        #                                   print("yes")
                        #                                   pass
                        else:
                            self.winn.t = 0
```

```
                #                                   print("yes")
                #                                   pass
                else:
                    self.winn.t = 0
                    time.sleep(1)
                    self.winn.t = self.timelimit
                    p1.join()
                    self.game.selected_tuple = None
                    for i in range(self.winn.board_size):
                        for j in range(self.winn.board_size):
                            self.winn.canvas.itemconfigure(
                                self.winn.tiles[i, j], outline="black", width=0)
                    self.winn.move = True
                    self.turn = 1

                    p1 = threading.Thread(target=self.winn.timeren)
                    p1.start()
                self.winn.Info.config(
                    text=f"AI moved From {self.game.ai_move[0]} To {self.game.ai_move[1]}, Time Taken: {(10-self.winn.t)+2} seconds.")
                self.winn.t = 0
                time.sleep(1)
                self.winn.t = self.timelimit
                p1.join()
                self.winn.move = True
                self.turn = 1

                p1 = threading.Thread(target=self.winn.timeren)
                p1.start()
                self.winn.move = True
                self.player2.move = False
                print("BACLLLLKKK")
            # Activating the timer so that player knows how much
            # time is remaning to play a move.
            if p1.isalive():
                self.winn.timer3.config(
                    text=f" {self.player2.color} , Time left = {self.winn.curtime}")
            #                                   print("yes")
            #                                   pass
            else:
                # if a player unable to perform a move in given
                # specific time then
                # player lost his turn to play move. He she needs to wait until next turn.
                self.winn.t = 0
                time.sleep(1)
                self.winn.t = self.timelimit
                p1.join()
                self.game.selected_tuple = None
                # clearing all the moves if player not played any.
                for i in range(self.winn.board_size):
                    for j in range(self.winn.board_size):
                        self.winn.gamewin.itemconfigure(
                            self.winn.tiles[i, j], outline="black", width=0)
                self.winn.move = True
                # assigning turn to next player.
                self.turn = 1
```

```
                # activaiting timer for next player

                p1 = threading.Thread(target=self.winn.timeren)
                p1.start()

            #                                   self.timer()
            #                                self.winn.makechar()

            elif (self.turn == 1 and self.winn.move == True):
                # setting the staus for next player
                self.winn.status.config(
                    text="Player 1 Turn: ", bg=self.player1.color)

                self.winn.makechar()
                # if player makes his move then turn will be assigned to other player
                if p1.isAlive():
                    self.winn.timer3.config(
                        text=f" {self.player1.color} , Time left = {self.winn.curtime} ")
                #                                   print("yes")
                #
                #                                   #pass
                # if player cannot make his move. his chance lost.
                else:
                    self.winn.t = 0
                    time.sleep(1)
                    self.winn.t = self.timelimit
                    p1.join()
                    self.game.selected_tuple = None
                    # clearing all the highlighted board moves.
                    for i in range(self.winn.board_size):
                        for j in range(self.winn.board_size):
                            self.winn.gamewin.itemconfigure(
                                self.winn.tiles[i, j], outline="black", width=0)
                    self.winn.move = False

                    p1 = threading.Thread(target=self.winn.timeren)
                    p1.start()

            #                               print("else2")
            #                     t1 = threading.Thread(target=self.winn.timer())

            # self.winn.move = False
            self.turn = 2 if self.turn == 1 else 1
            self.game.turn = 2 if self.game.turn == 1 else 1

        # op=0
        # while(True)  :
        #     op+=1
        #     self.winn.status.config(text=f"Player {self.endgame()} Won The Game.")
        #     time.sleep(1)
        #     if op==5:
        #         break
        self.winn.t = 0
        self.winn.makechar()
        p1.join()
```

```
        pls = self.endgame()
        if pls == 2:

            colr = self.player2.color
        else:
            colr = self.player1.color
        self.winn.status.config(
            text=f"Player {self.endgame()} Won The Game.", bg=colr)
        self.winn.update()
        time.sleep(5)
        print(f"Player {self.endgame()} Won The Game.")
    # Fucnction to Check that game has a winner or not..

    def endgame(self):

        # this function called everytime in loop to check that a player won or not
        # after every single move made by player
        if (self.player1.all_char_opp() and not (self.player2.all_char_opp())):
            return 1
        # if player reach its goal that is home for other player then he/she wins
        elif (self.player2.all_char_opp() and not (self.player1.all_char_opp())):
            return 2
        # otherwise game continues until we get a winner
        else:
            return True  # 0 for notterminate

# Gui Class which is made using tkinter
# this class handels all the functionality of GUI to show the moves
# SHow players,staus,timer and evrything else.


class Halmaguimode(tk.Tk):
    def __init__(self, board, *args, **kwargs):
        # initialize  the parent class of tkinter
        tk.Tk.__init__(self, *args, **kwargs)
        # setting it to true so that our window can be resized
        self.resizable(True, True)

        self.configure(bg='#2F4F4F')  # setting Background color
        # initializing Our game Objects  which helps in calculating game.
        self.game = board
        self.time = 0
        self.board_size = board.gamesiz()
        self.t = 20
        self.backup = 0

     #  Assigning X and Y values to our grid bord so that we can
        # calculate each move by its Cordinate
        # that is X,Y in our case it is 1,A for Eg.
        for i in range(self.board_size):
            roww = tk.Label(self, text=i + 1, font='Times',
                            bg='#2F4F4F', fg='#DCDCDC')
            roww.grid(row=i + 1, column=0)

            coll = tk.Label(self, text=chr(i + 97).upper(),
```

Main.py > HalmaGame > play

```
            roww.grid(row=i + 1, column=0)

            coll = tk.Label(self, text=chr(i + 97).upper(),
                            font='Times', bg='#2F4F4F', fg='#DCDCDC')
            coll.grid(row=0, column=i + 1)
        # Making Staus bar to print the current status of that that is happening
        self.status = tk.Label(self, height=1, text="Welcome to Halma Game........Green Player Start Game", width=50,
                               relief="raised", font="Times", bd=3, bg="#001414",
                               fg="#DCDCDC")  # Customizing the status bar with some additional prameters
        self.timer3 = tk.Label(self, height=1, text="TIMER....", width=50, relief="raised", font="Times", bd=3,
                               bg="#001414",
                               fg="#DCDCDC")
        self.Info = tk.Label(self, height=1, text="TIMER....", width=50, relief="raised", font="Times", bd=3,
                             bg="#001414",
                             fg="#DCDCDC")
        # Making our Game timer so that each player has same time to play a move

        #         status.grid(row=0,column=i+1)
        # initializing our Tiles for the Game.

        self.tiles = {}

        # Making a Blank Canvas of size 590*590
        # with some additonal Style Parameters
        self.gamewin = tk.Canvas(self, width=590, height=590, bd=4,
                                 relief="ridge", bg="#778899", highlightthickness=0)
        # using Grid insted of pack because we cannot use pack with grid.
        self.gamewin.grid(
            row=1, column=1, columnspan=self.board_size, rowspan=self.board_size)
        # similarly setting status bar using grid method
        self.status.grid(columnspan=self.board_size +
                         2, rowspan=self.board_size)
        # similarly setting Timer bar using grid method
        self.timer3.grid(columnspan=self.board_size +
                         2, rowspan=self.board_size)
        self.Info.grid(columnspan=self.board_size + 2, rowspan=self.board_size)
        self.title('Halma A.I Game')  # main title of the window
        # some additonal configuration
        self.columnconfigure(0, minsize=50)
        self.rowconfigure(0, minsize=50)
        self.columnconfigure(self.board_size + 1, minsize=50)
        self.rowconfigure(self.board_size + 1, minsize=50)
        self.gamewin.bind("<Configure>", self.makegrid)
        self.game.selected_tuple = None
    # our TImer which is called in each player move. so that each player
    # gets equal amount of time to make a move.

    def timeren(self):
        while self.t != 0:
            print("TIMEEEEEEEEE", self.t)
            # using divmod to ge the time in min and secs
            mins, secs = divmod(self.t, 60)
            self.curtime = '{:02d}:{:02d}'.format(mins, secs)
            # subtracting our decremented by 1 after a sec..
            time.sleep(1)
            #             print(self.ti)
```

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:
```

```
                    color = '#12C47A'
                # green color always on the bottom left of the game screen
                else:
                    color = '#0FA868'
                # all other tiles are maked as playzone of the agame with a tan collor
                else:
                    if ((row + col) % 2 == 0):
                        color = '#ECCB96'
                    else:
                        color = '#BAA077'
                # creating rectangle Checked Grids.
                checks = self.gamewin.create_rectangle(
                    x1, y1, x2, y2, tags="checks", width=0, fill=color)
                self.tiles[col, row] = checks
                # binding our checkers with our onpress function which is activated whenever we click to make move.
                self.gamewin.tag_bind(
                    checks, "<1>", lambda event, row=row, col=col: self.onpress(row + 1, col + 1))

        self.makechar()
    # Fucntion used to make Charcter players that is red player and Green player
    # we used Oval shape to assign a shape to a player of theri respectve color

    def makechar(self):

        canvas_width = 600
        heightcanva = 600
        marginnsiz = 10
        # similarly calucated box size as above
        boxx = int(heightcanva / self.board_size)
        # Initializing Our Tokens of the game. for both Players
        self.p1char = self.game.player1.charss
        self.p2char = self.game.player2.charss
        # delete previous tokens canvas
        self.gamewin.delete('character')
        c = 0

        for i in [range(len(self.p1char)), range(len(self.p2char))]:
            for i in i:
                if c == 0:
                    col = self.p1char[i].x - 1
                    row = self.p1char[i].y - 1

                else:
                    col = self.p2char[i].x - 1
                    row = self.p2char[i].y - 1
                # calculation all the cordinates so that we can create
                # a oval shape for each player with their respective colors.

                x1 = col * boxx + marginnsiz / 2
                y1 = row * boxx + marginnsiz / 2
                # Using SImple Mathematics
                x2 = (col + 1) * boxx - marginnsiz / 2
                y2 = (row + 1) * boxx - marginnsiz / 2
                # Creating Shapes for charcted with their respecitve colors
                if c == 0:
                    if (self.game.player1.color == "GREEN"):
```

Main.py > HalmaGame > play

```
                y2 = (row + 1) * boxx - marginnsiz / 2
                # Creating Shapes for charcted with their respecitve colors
                if c == 0:
                    if (self.game.player1.color == "GREEN"):
                        character = self.gamewin.create_oval(
                            x1, y1, x2, y2, tags="character", width=0, fill="#187c53")

                    else:
                        character = self.gamewin.create_oval(
                            x1, y1, x2, y2, tags="character", width=0, fill="#9b423c")

                    self.gamewin.tag_bind(
                        character, "<1>", lambda event, row=row, col=col: self.onpress(row + 1, col + 1))

                 # And Binding each of our tokens of game with our onpress fucntion
                # so that whenever a player click his token.
                # he can look upon all the
                # possible Moves
                else:
                    if (self.game.player1.color == "GREEN"):

                        character = self.gamewin.create_oval(
                            x1, y1, x2, y2, tags="character", width=0, fill="#9b423c")
                    else:
                        character = self.gamewin.create_oval(
                            x1, y1, x2, y2, tags="character", width=0, fill="#187c53")
                    self.gamewin.tag_bind(
                        character, "<1>", lambda event, row=row, col=col: self.onpress(row + 1, col + 1))
            c += 1
         # updating our game interface.
        self.update()
     # function works on clicking a token of a game.
    # if a player press one of his pice of token this funciton
    # shows all the possible moves in the interface with a boundary Highlighed
    # and if a player presses another tile to move.
    # this function is responsible to make a move using interface.

    def onpress(self, row, column):
        global ch
        checks = self.tiles[column - 1, row - 1]
        # all possible moves.
        movesboxes = []
        movesboxes.append(checks)
        #         print("CHHH",ch)
        if self.move:
            # if Player one thinking to make a move.
            #             print("0000000000000000000000000000000000000000")
            if (self.game.selected_tuple == None and self.game.player1.charlocCheck(column, row)):
                self.status.config(
                    text=f"......Player 1 is Thinking.............", bg=f"{self.game.player1.color}")
                # whenever he select one of his token
                if (self.game.player1.charlocCheck(column, row)):
                    character = self.game.player1.getChar(column, row)
                elif (self.game.player2.charlocCheck(column, row)):
                    character = self.game.player2.getChar(column, row)
```

Main.py > HalmaGame > play

```
                # whenever he select one of his token
                if (self.game.player1.charlocCheck(column, row)):
                    character = self.game.player1.getChar(column, row)
                elif (self.game.player2.charlocCheck(column, row)):
                    character = self.game.player2.getChar(column, row)
                 # it shows all the validated moves which a playr can make
                validMoves = self.game.validatemove(character)

                for i in range(len(validMoves)):
                    (x, y) = validMoves[i]
                    checks = self.tiles[x - 1, y - 1]
                    movesboxes.append(checks)
                 # highlighting alll the possible moves with black boundary for better vision.
                for i in range(len(movesboxes)):
                    self.gamewin.itemconfigure(
                        movesboxes[i], outline="black", width=2)

                self.game.selected_tuple = (column, row)
            # if a player wants to make a move then he must need to click on otehr tile.
            elif (self.game.selected_tuple != None and (column, row) in self.game.validatemove(
                    self.game.player1.getChar(self.game.selected_tuple[0], self.game.selected_tuple[1]))):
                #                 print("IF2")
                self.status.config(
                    text="Player 1 Moved,Player 2 Turn", bg=f"{self.game.player2.color}")
                # getting the cordinate of other tile
                (x, y) = self.game.selected_tuple
                # movind the token from one tile to other using movement function

                self.game.char_movement((y, x), (row, column))
                # removing all the highlighhed block after making a move
                for i in range(self.board_size):
                    for j in range(self.board_size):
                        self.gamewin.itemconfigure(
                            self.tiles[i, j], outline="black", width=0)
                self.game.selected_tuple = None
                self.t = 0
                # Assiging turn to other player if a player made a move.
                time.sleep(1)
                self.t = self.backup
                p1.join()
                p1 = threading.Thread(target=self.timeren)
                thread.sleep(1)
                #                 self.t=20
                p1.start()
                self.move = False

            else:
                #                 print("Else2")
                self.game.selected_tuple = None
                for i in range(self.board_size):
                    for j in range(self.board_size):
                        self.gamewin.itemconfigure(
                            self.tiles[i, j], outline="black", width=0)
        else:
            # similarly if other player want to do same thins
            # all the conditions works similarly.
```

```
            # all the conditions works similarly.
            #             print("ELSEEEEEEEEEEEEEEEEEEEEEEEEEEEEEe")
            if (self.game.selected_tuple == None and self.game.player2.charlocCheck(column, row)):
                self.status.config(
                    text=f"......Player 2 is Thinking.............", bg=f"{self.game.player2.color}")
                #                 print("ELSEIF")
                if (self.game.player1.charlocCheck(column, row)):
                    #                     print("ELSEIF 2 player if")
                    character = self.game.player1.getChar(column, row)
                elif (self.game.player2.charlocCheck(column, row)):
                    character = self.game.player2.getChar(column, row)
                #                     print("ELSEIF 2 player else")

                validMoves = self.game.validatemove(character)

                for i in range(len(validMoves)):
                    (x, y) = validMoves[i]
                    checks = self.tiles[x - 1, y - 1]
                    movesboxes.append(checks)

                for i in range(len(movesboxes)):
                    self.gamewin.itemconfigure(
                        movesboxes[i], outline="black", width=2)

                self.game.selected_tuple = (column, row)

            elif (self.game.selected_tuple != None and (column, row) in self.game.validatemove(
                    self.game.player2.getChar(self.game.selected_tuple[0], self.game.selected_tuple[1]))):
                #                 print("ELIF")
                self.status.config(
                    text="Player 2 Moved,Player 1 Turn", bg=f"{self.game.player1.color}")
                (x, y) = self.game.selected_tuple
                self.game.char_movement((y, x), (row, column))

                for i in range(self.board_size):
                    for j in range(self.board_size):
                        self.gamewin.itemconfigure(
                            self.tiles[i, j], outline="black", width=0)
                self.game.selected_tuple = None
                #                 p1.stop()
                #                 self.t=0
                self.t = 0
                time.sleep(1)
                self.t = self.backup
                p1.join()
                del p1
                p1 = threading.Thread(target=self.timeren)
                #                 self.t=20
                #                 time.sleep(1)
                p1.start()
                self.move = True
            else:
                #                 print("Else")
                self.game.selected_tuple = None
                for i in range(self.board_size):
```

```
                    self.game.selected_tuple = None
                    for i in range(self.board_size):
                        for j in range(self.board_size):
                            self.gamewin.itemconfigure(
                                self.tiles[i, j], outline="black", width=0)

            self.update()


if __name__ == "__main__":
    # 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, "GREEN", "RED", "AI", "ON")  # For Human vs AI
    # game= HalmaGame(8,10,"RED","GREEN",None,"ON") # for human vs Human

```

Phase 1 Source code:- charutils.py

```python
import copy

# Charcated Util Class which is used to create
#Objects of players when the game begins.

class CharacterUtil:
  def __init__(self, color, boardSize):
    # Setting each player with its resepective colors
    self.color = color
    self.starposchar(boardSize)
    # Home and Goal Cordinates
    # home cordinate is Goal cordinate for the other player
    #and vice versa
    self.homeCoord = []
    self.goalCoord = []
  # Functon to get the starting postion of the charcters
  def starposchar(self, boardSize):
    # Creating empty list varibale to track the home
    # Goal and Tokens
    self.charss = []
    self.home = []
    self.goal = []
    # ASsigning Iteration based on the board size.
    if boardSize == 8:
      lenmax = 5
    elif boardSize == 10:
      lenmax = 6
    else: # default is 16 x 16
      lenmax = 7
    # Iterating though our borad to set the home GOal

    for i in range(1, lenmax):
      for j in range(1, lenmax):
        # and the positon of the player. so that we can keep a track later.
        if (i + j <= lenmax and i < lenmax and j < lenmax):
          self.charss.append(Utils(boardSize - j + 1, boardSize - i + 1))
          self.home.append((boardSize - j + 1, boardSize - i + 1))
          self.goal.append((j, i))
    # For debugging PUrpose
    print(self.charss)
    print(self.home)
    print(self.goal)


    if self.color == 'RED':
      # The other Player also have similar Cordinates the
      # only change is that
      temp = copy.deepcopy(self.home)
      self.home = copy.deepcopy(self.goal)
      # The goal cordinate for player 1 is Home Corindate of player 2

      self.goal = temp
      # and the Home Cordinate of player  1 is the Goal cordinate of player 2.
      for i in range(len(self.charss)):
        self.charss[i].x = self.home[i][0]
        self.charss[i].y = self.home[i][1]
```

```
58
59
60      # FUction 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 cordinate 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      # FUcntion To check the Home Cordinate of the Chracter.
85      def charHomeCheck(self,x,y):
86          koor =(x, y)
87          # If Givenn Cordinate is in our HOme 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 cordinate
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 chratcers if it is avalible
```

```
        # it returns the chratcers if it is avalible
        if (self.charss[i].x == row and self.charss[i].y == column):
            character = self.charss[i]
            return character
            found = True
        else:
            i +=1


    # Fucntion to move the chracter from one Grid Locatio
    # TO the other grid location
    def char_movement(self, gridfrom, gridto):
        # get the current Location
        (x, y) = gridfrom
        (x2, y2) = gridto# get the destination
        # Looping though our Tokens
        for p in self.charss:
            # if its x cordinates matches with the current cordinate
            if p.x == x and p.y == y:
                # and if it is in Home Cord then we can make a movement
                if (x, y) in self.home and (x2, y2) not in self.home:
                    p.char_gone = True
                # Similarly if it is goal then we cannot make a move
                elif (x, y) not in self.goal and (x2, y2) in self.goal:
                    p.char_arr = True
                # Assigning GOal Cordinate
                p.x = x2
                p.y = y2
                self.charss = sorted(self.charss, key=lambda p: (p.x, p.y))
                break
# Utils Class for More debugging Purpose to get each and ANy single variable.
class Utils:
    def __init__(self, x, y):
        #setting the Cordinates
        self.setCoordinate(x, y)
        #Print("CHAR INISITALIZE")
        self.setChar_Gone(False)
        self.setChar_arr(False)
    # TO set the cordinates again
    def setCoordinate(self,x, y):
        self.x = x
        self.y = y
    # to set Char Dest
    def setChar_Gone(self, char_gone):
        self.char_gone = char_gone
    # To set Char Currnet
    def setChar_arr(self, char_arr):
        self.char_arr = char_arr
    # To get the X cordinates of Player
    def getCoordinateX(self):
        return (self.x)
    # TO get Y cordinates
    def getCoordinateY(self):
        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 Constructer 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 > ...

```python
#importing other package
from Utilities import Utilities
#Our Game Class which handels and Genrate moves in the back
#It is also responsible to generate Jumps while making moves.
class Game:
  #initializing our construter of the game.
  def __init__(self, boardSize, timeLimit, p1, p2, human):
    #assiging all the parametres essential to play the game
    self.boardSize = boardSize
    self.timelimit = timeLimit
    self.player1 = p1
    self.player2 = p2
    #assiginng therir respective color to the players
    self.p1cop = p1 if p1.color == "GREEN" else p2
    self.p2cop = p2 if p2.color == "RED" else p1
    #assigning turn to a player
    self.turn = 1
    #getting the cordianes of our board
    self.coordinate = [[Utilities(i, j) for i in range(self.boardSize)] for j in range(self.boardSize)]
    self.depth = 2
    #humans as both players are playing the game.
    self.human = human

    #Assigning maximum iterations according to our board size.
    if self.boardSize == 8:
      lenmax = 4
    elif self.boardSize == 10:
      lenmax = 5
    else:
      lenmax = 6
#looping through all the iterations.
    for i in range(lenmax):
      for j in range(lenmax):
        if (i + j < lenmax and i < 6 and j < 6):
          #Making red player home  cordinate red and
          self.coordinate[i][j].color = "RED"
          self.coordinate[i][j].character = 2
          #making green player home cordinates green
          #Appending all the home and goal cordinates of
          #player 1 and Player 2

          self.p2cop.homeCoord.append(self.coordinate[i][j])
          self.p1cop.goalCoord.append(self.coordinate[i][j])
          #similarly for Player 2 Assigning COlor and Home goal cordinates
          self.coordinate[self.boardSize - 1 - i][self.boardSize - 1 - j].color = "GREEN"
          self.coordinate[self.boardSize - 1 - i][self.boardSize - 1 - j].character = 1
          self.p1cop.goalCoord.append(self.coordinate[self.boardSize - 1 - i][self.boardSize - 1 - j])
          self.p2cop.homeCoord.append(self.coordinate[self.boardSize - 1 - i][self.boardSize - 1 - j])


  #function to track Board Size
  def gamesiz(self):
    return self.boardSize
  #Function to Chehck that the cordinate is empty or not to make a move
  def checkboxemp(self,x,y):
```

```
    def checkboxemp(self,x,y):
      #if a box is empty then player can make a move there if eligible
      #if empty it return True
      if(self.player1.charlocCheck(x,y) or self.player2.charlocCheck(x,y)):
        return False
      #else It return false
      else:
        return True
    # Fucntion to Check the HOme Cordinates of the player
    def checkhomecord(self, player, x, y):
      return(player.charHomeCheck(x,y))
    #Function to check the goal cordinates of the player
    # so that we can check that a player reached goal or not
    def checkgoalcord(self, player, x, y):
      return(player.charGoalCheck(x,y))
    # Getting all the list of avaibalve positons to make a move.
    def allemppositions(self, position, alone):
      # mengecek semua yang berdelta 1 itu kosong, dan gak melebihi size board
      x, y = position
      list_of_positions = []# to retrive all the locaitions from the bord.
      # if player is in the center and no ther token of otehr player is in negibour
      # then it can move one step to any direction
      if (alone == 1):
        list_of_positions.append((x+1, y))
        list_of_positions.append((x+1, y+1))
        list_of_positions.append((x, y+1))
        list_of_positions.append((x-1, y+1))
        list_of_positions.append((x-1, y))
        list_of_positions.append((x-1, y-1))
        list_of_positions.append((x, y-1))
        list_of_positions.append((x+1, y-1))
    #if other player lies in the neghibouruing tile then we need to
      #calculate jumps all over the boards
      #appending all the avaiable positions to our positions list
      else:
        if (not(self.checkboxemp(x+1,y)) and (self.checkboxemp(x+2, y))):
          list_of_positions.append((x+2, y))
        if (not(self.checkboxemp(x-1,y)) and (self.checkboxemp(x-2, y))):
          list_of_positions.append((x-2, y))
        if (not(self.checkboxemp(x,y+1)) and (self.checkboxemp(x, y+2))):
          list_of_positions.append((x, y+2))
        if (not(self.checkboxemp(x,y-1)) and (self.checkboxemp(x, y-2))):
          list_of_positions.append((x, y-2))
        if (not(self.checkboxemp(x+1,y+1)) and (self.checkboxemp(x+2, y+2))):
          list_of_positions.append((x+2, y+2))
        if (not(self.checkboxemp(x+1,y-1)) and (self.checkboxemp(x+2, y-2))):
          list_of_positions.append((x+2, y-2))
        if (not(self.checkboxemp(x-1,y+1)) and (self.checkboxemp(x-2, y+2))):
          list_of_positions.append((x-2, y+2))
        if (not(self.checkboxemp(x-1,y-1)) and (self.checkboxemp(x-2, y-2))):
          list_of_positions.append((x-2, y-2))

      length = len(list_of_positions)
      i = 0
```

```
    #once we get all the position to make a move its time to get Valid Positons.
    while (i < length):
      (x, y) = list_of_positions[i]

      if(x<1 or y<1 or x>self.boardSize or y>self.boardSize):
        list_of_positions.remove(list_of_positions[i])
        length -= 1

      elif (alone == 1 and not(self.checkboxemp(x, y))):
        list_of_positions.remove(list_of_positions[i])
        length -= 1
      else:
        i += 1
    #all valid positons are returned to the player to make a move



    return list_of_positions
#Function to Check JUmp on the board based

  def checkjump(self, position, jumps, last_position):  # On the current  Positon of a player
    #jumps can be made is a plyer is in neghibouring tile and the next tile
    # to the other player is Empty
    lis_available_jump = self.allemppositions(position, 2)

    try:
      lis_available_jump.remove(last_position)
    except:
      pass

    if (len(lis_available_jump) ==  0):
      return jumps
    else:
      # Getting all the valid tiles to make a jump
      #from one plce to another

      for i in range (len(lis_available_jump)):
        # if a tile is not avaiable to jump it is removed.
        if lis_available_jump[i] not in jumps:
          jumps.append(lis_available_jump[i])
          #with the help of Recursion we check eachjumps for tiles.
          self.checkjump(lis_available_jump[i], jumps, position)
  # if a player is making a move then it is validated that player is making a correct move or
  def validatemove(self, character):
    if (self.player1.charlocCheck(character.x, character.y)):
        player = self.player1
        print("IF")
    else:

        player = self.player2
        print(" ELse")

    ##getting the current position of the Player
    current_position = (character.x, character.y)
    print("GET  CURENT",current_position)
```

```
165
166       # Once we have alll the available positons for the move. we
167       # can validte each move. and jUmps
168       list_of_positions = self.allemppositions(current_position, 1)
169       all_jumps = self.allemppositions(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 appened 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 retrived we can validate to the avaiable Jumps to make a mov
178         self.checkjump(all_jumps[i], jumps, current_position)
179         #looping untill we get all teh 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 a 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("PLYER WHILE",player)
193         #checking all the home and goal cordinates of a player if it is not then we removed tha
194         # the list of positos
195         if (character.char_arr and not(self.checkgoalcord(player, x, y))) or (character.char_go
196           list_of_positions.remove(list_of_positions[i])
197           length -= 1
198         else:
199           i += 1
200       # once we have alist we sorted it
201       list_of_positions = sorted(list_of_positions, key=lambda tup: (tup[0], tup[1]))
202       return list_of_positions
203
204
205     # fucntion in backend to move chracter from one player to another.
206     def char_movement(self, xpos, ypos):
207       #getting from cordinate
208
209       gridfrom = self.coordinate[xpos[0]-1][xpos[1]-1]
210       #getting the cordinate where we have to move the player
211       gridto = self.coordinate[ypos[0]-1][ypos[1]-1]
212       # Chekcing that the move is valid or not
213
214       if gridfrom.character == 0 or gridto.character != 0:
215         print("Invalidddddddddddddd")
216         return
217       # only a valid move can be played
218       # onces a move is validated
```

```
218     # onces 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("Invalidddddddddddd")
227       return
228     gridto.character = gridfrom.character
229     gridfrom.character = 0
```

Main.py

```python
# Import Nessecary Libraries
from charutils import CharacterUtil
from game import Game
import tkinter as tk
import threading
import time

# initialized a global variable
ch = 0

# main Game class HalmaGame


class HalmaGame:
    # Initialize Cosntrustor so that player can be assigned colors and their respective goal.
    def __init__(self, boardsize, timelimit, player1, player2):
        # assignining player from our characterUtils Class which is on Other script
        self.player1 = CharacterUtil(player1, boardsize)
        # assignining player from our characterUtils Class which is on Other script
        self.player2 = CharacterUtil(player2, boardsize)
        self.human = False
        # Initizlize Game object which is the main Class that Handels overall game moves and spawns.
        self.game = Game(boardsize, timelimit, self.player1,
                         self.player2, self.human)
        # Turn so that each player can play a move simolutnously
        self.turn = 1
        # Initilizing Time limit so that player has to play move within timelimit
        self.timelimit = self.game.timelimit
        # Initializing Our Game mode Which is GUI based
        self.mode = "GUI"
        self.play()
    # play function runs a loop until the game is not finished .. each player gets
    # one chance to play one move
    # if a user forget to play his move
    # within a time limit then he cannot make his move until his next turn

    def play(self):
        # initializing our GUI which is created using tkinter
        self.winn = Halmaguimode(self.game)
        # initizling first move so that a player can play his move
        self.winn.move = True

        self.turn = 2 if self.player1.color == "RED" else 1
        self.winn.move = False if self.player1.color == "RED" else True
        print(self.turn)
        # setting the Status bar which shows Player turn.
        # game alwways starts with Green Player as mentioned in the doc.
        self.winn.status.config(
            text=f"Green Player Start Game...", bg="#187c53")
        print("here")
        self.winn.t = self.timelimit
        self.winn.backup = self.timelimit
        # creating a threded object so that we can also monitor the time limit.
        p1 = threading.Thread(target=self.winn.timeren)
        p1.start()
        checks = 0
```

```
        p1.start()
        checko = 0
        # looping until a player wons the game
        while (self.endgame() == True):
            # endgame checks that a player has won or not.
            # a player won if his all the spawns are in opponent home.
            # which is goal in player case
            if (self.turn == 2 and not (self.winn.move)):
                print("PLAYER@@@@@")
                # playing Move and setting the status bar for player
                self.winn.status.config(
                    text="Player 2 Turn: ", bg=self.player2.color)

                self.winn.makechar()
                # Activating the timer so that player knows how much
                # time is remaning to play a move.
                if p1.isAlive():
                    self.winn.timer3.config(
                        text=f" {self.player2.color} , Time left = {self.winn.curtime}")
                #                                   print("yes")
                #                                   pass
                else:
                    # if a player unable to perform a move in given
                    # specific time then
                    # player lost his turn to play move. He she needs to wait until next turn.
                    self.winn.t = 0
                    time.sleep(1)
                    self.winn.t = self.timelimit
                    p1.join()
                    self.game.selected_tuple = None
                    # clearing all the moves if player not played any.
                    for i in range(self.winn.board_size):
                        for j in range(self.winn.board_size):
                            self.winn.gamewin.itemconfigure(
                                self.winn.tiles[i, j], outline="black", width=0)
                    self.winn.move = True
                    # assigning turn to next player.
                    self.turn = 1
                    # activaiting timer for next player

                    p1 = threading.Thread(target=self.winn.timeren)
                    p1.start()

            #                                   self.timer()
            #                                self.winn.makechar()

            elif (self.turn == 1 and self.winn.move == True):
                # setting the staus for next player
                self.winn.status.config(
                    text="Player 1 Turn: ", bg=self.player1.color)

                self.winn.makechar()
                # if player makes his move then turn will be assigned to other player
                if p1.isAlive():
                    self.winn.timer3.config(
                        text=f" {self.player1.color} , Time left = {self.winn.curtime} ")
```

```
                #                                   #pass
                # if player cannot make his move. his chance lost.
                else:
                    self.winn.t = 0
                    time.sleep(1)
                    self.winn.t = self.timelimit
                    p1.join()
                    self.game.selected_tuple = None
                    # clearing all the highlighted board moves.
                    for i in range(self.winn.board_size):
                        for j in range(self.winn.board_size):
                            self.winn.gamewin.itemconfigure(
                                self.winn.tiles[i, j], outline="black", width=0)
                    self.winn.move = False

                    p1 = threading.Thread(target=self.winn.timeren)
                    p1.start()

            #                             print("else2")
            #                     t1 = threading.Thread(target=self.winn.timer())

            # self.winn.move = False
            self.turn = 2 if self.turn == 1 else 1
            self.game.turn = 2 if self.game.turn == 1 else 1

        # op=0
        # while(True)  :
        #     op+=1
        #     self.winn.status.config(text=f"Player {self.endgame()} Won The Game.")
        #     time.sleep(1)
        #     if op==5:
        #         break
        self.winn.t = 0
        self.winn.makechar()
        p1.join()
        pls = self.endgame()
        if pls == 2:

            colr = self.player2.color
        else:
            colr = self.player1.color
        self.winn.status.config(
            text=f"Player {self.endgame()} Won The Game.", bg=colr)
        self.winn.update()
        time.sleep(5)
        print(f"Player {self.endgame()} Won The Game.")
    # Fucnction to Check that game has a winner or not..

    def endgame(self):

        # this function called everytime in loop to check that a player won or not
```

```
        # this function called everytime in loop to check that a player won or not
        # after every single move made by player
        if (self.player1.all_char_opp() and not (self.player2.all_char_opp())):
            return 1
        # if player reach its goal that is home for other player then he/she wins
        elif (self.player2.all_char_opp() and not (self.player1.all_char_opp())):
            return 2
        # otherwise game continues until we get a winner
        else:
            return True  # 0 for notterminate

# Gui Class which is made using tkinter
# this class handels all the functionality of GUI to show the moves
# SHow players,staus,timer and evrything else.


class Halmaguimode(tk.Tk):
    def __init__(self, board, *args, **kwargs):
        # initialize  the parent class of tkinter
        tk.Tk.__init__(self, *args, **kwargs)
        # setting it to true so that our window can be resized
        self.resizable(True, True)

        self.configure(bg='#2F4F4F')  # setting Background color
        # initializing Our game Objects  which helps in calculating game.
        self.game = board
        self.time = 0
        self.board_size = board.gamesiz()
        self.t = 20
        self.backup = 0

      #  Assigning X and Y values to our grid bord so that we can
        # calculate each move by its Cordinate
        # that is X,Y in our case it is 1,A for Eg.
        for i in range(self.board_size):
            roww = tk.Label(self, text=i + 1, font='Times',
                            bg='#2F4F4F', fg='#DCDCDC')
            roww.grid(row=i + 1, column=0)

            coll = tk.Label(self, text=chr(i + 97).upper(),
                            font='Times', bg='#2F4F4F', fg='#DCDCDC')
            coll.grid(row=0, column=i + 1)
        # Making Staus bar to print the current status of that that is happening
        self.status = tk.Label(self, height=3, text="Welcome to Halma Game........Green Player Start Game", width=50,
                               relief="raised", font="Times", bd=3, bg="#001414",
                               fg="#DCDCDC")  # Customizing the status bar with some additional prameters
        self.timer3 = tk.Label(self, height=3, text="TIMER....", width=50, relief="raised", font="Times", bd=3,
                               bg="#001414",
                               fg="#DCDCDC")
        # Making our Game timer so that each player has same time to play a move

        #         status.grid(row=0,column=i+1)
        # initializing our Tiles for the Game.

        self.tiles = {}
```

```
218
219         # Making a Blank Canvas of size 590*590
220         # with some additonal Style Parameters
221         self.gamewin = tk.Canvas(self, width=590, height=590, bd=4,
222                                  relief="ridge", bg="#778899", highlightthickness=0)
223         # using Grid insted 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 additonal 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     # our TImer which is called in each player move. so that each player
241     # gets equal amount of time to make a move.
242
243     def timeren(self):
244         while self.t != 0:
245             print("TIMEEEEEEEEE", self.t)
246             # using divmod to ge the time in min and secs
247             mins, secs = divmod(self.t, 60)
248             self.curtime = '{:02d}:{:02d}'.format(mins, secs)
249             # subtracting our decremented by 1 after a sec..
250             time.sleep(1)
251             #             print(self.ti)
252             if self.t == 0:
253                 break
254             self.t -= 1
255         print("HEER BREAKDE")
256
257     #             print(self.curtime)
258     #             if self.t==0:
259     #                 self.t=60
260     #             self.status.config(text="Timeleft")
261     # Function used to make grid in our window canvas
262
263     def makegrid(self, event=None):
264         self.gamewin.delete("checks")
265         heightcanva = 600
266         marginnsiz = 1
267         # Calculating each box size of our game as our game has 3 boards 8,10,16
268         boxx = int(heightcanva / self.board_size)
269         # looping thoug our board  row and cos 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):
```

```
        # looping thoug our board  row and cos so that we can create a interface
        # that is grided so that we can easily make moves.
        for col in range(self.board_size):
            for row in range(self.board_size):
                # calculcating all the cordiates
                x1 = col * boxx + marginnsiz / 2
                y1 = row * boxx + marginnsiz / 2
                x2 = (col + 1) * boxx - marginnsiz / 2
                y2 = (row + 1) * boxx - marginnsiz / 2

                if (self.board_size == 8):
                    player1 = 4
                    player2 = 10
                elif (self.board_size == 10):
                    player1 = 5
                    player2 = 13
                else:
                    player1 = 6
                    player2 = 24
                 # assiging colors to each player Home with their respective colors
                if ((row + col) < player1):
                    if ((row + col) % 2 == 0):
                        color = '#AC352E'
                    else:
                        color = '#D0352E'

                 # Red color always on the top left of the game screen
                elif ((row + col) > player2):
                    if ((row + col) % 2 == 0):
                        color = '#12C47A'
                # green color always on the bottom left of the game screen
                    else:
                        color = '#0FA868'
                # all other tiles are maked as playzone of the agame with a tan collor
                else:
                    if ((row + col) % 2 == 0):
                        color = '#ECCB96'
                    else:
                        color = '#BAA077'
                # creating rectangle Checked Grids.
                checks = self.gamewin.create_rectangle(
                    x1, y1, x2, y2, tags="checks", width=0, fill=color)
                self.tiles[col, row] = checks
                # binding our checkers with our onpress function which is activated whenever we click to make move.
                self.gamewin.tag_bind(
                    checks, "<1>", lambda event, row=row, col=col: self.onpress(row + 1, col + 1))

        self.makechar()
    # Fucntion used to make Charcter players that is red player and Green player
    # we used Oval shape to assign a shape to a player of theri respectve color

    def makechar(self):

        canvas_width = 600
        heightcanva = 600
        marginnsiz = 10
```

```
        marginnsiz = 10
        # similarly calucated box size as above
        boxx = int(heightcanva / self.board_size)
        # Initializing Our Tokens of the game. for both Players
        self.p1char = self.game.player1.charss
        self.p2char = self.game.player2.charss
        # delete previous tokens canvas
        self.gamewin.delete('character')
        c = 0

        for i in [range(len(self.p1char)), range(len(self.p2char))]:
            for i in i:
                if c == 0:
                    col = self.p1char[i].x - 1
                    row = self.p1char[i].y - 1

                else:
                    col = self.p2char[i].x - 1
                    row = self.p2char[i].y - 1
                 # calculation all the cordinates so that we can create
                # a oval shape for each player with their respective colors.

                x1 = col * boxx + marginnsiz / 2
                y1 = row * boxx + marginnsiz / 2
                # Using SImple Mathematics
                x2 = (col + 1) * boxx - marginnsiz / 2
                y2 = (row + 1) * boxx - marginnsiz / 2
                # Creating Shapes for charcted with their respecitve colors
                if c == 0:
                    if (self.game.player1.color == "GREEN"):
                        character = self.gamewin.create_oval(
                            x1, y1, x2, y2, tags="character", width=0, fill="#187c53")

                    else:
                        character = self.gamewin.create_oval(
                            x1, y1, x2, y2, tags="character", width=0, fill="#9b423c")

                    self.gamewin.tag_bind(
                        character, "<1>", lambda event, row=row, col=col: self.onpress(row + 1, col + 1))

                 # And Binding each of our tokens of game with our onpress fucntion
                # so that whenever a player click his token.
                # he can look upon all the
                # possible Moves
                else:
                    if (self.game.player1.color == "GREEN"):

                        character = self.gamewin.create_oval(
                            x1, y1, x2, y2, tags="character", width=0, fill="#9b423c")
                    else:
                        character = self.gamewin.create_oval(
                            x1, y1, x2, y2, tags="character", width=0, fill="#187c53")
                    self.gamewin.tag_bind(
                        character, "<1>", lambda event, row=row, col=col: self.onpress(row + 1, col + 1))
            c += 1
         # updating our game interface
```

```
                    character, "<1>", lambda event, row=row, col=col: self.onpress(row + 1, col + 1))
                c += 1
            # updating our game interface.
            self.update()
        # function works on clicking a token of a game.
        # if a player press one of his pice of token this funciton
        # shows all the possible moves in the interface with a boundary Highlighed
        # and if a player presses another tile to move.
        # this function is responsible to make a move using interface.

        def onpress(self, row, column):
            global ch
            checks = self.tiles[column - 1, row - 1]
            # all possible moves.
            movesboxes = []
            movesboxes.append(checks)
            #         print("CHHH",ch)
            if self.move:
                # if Player one thinking to make a move.
                #             print("0000000000000000000000000000000000000000")
                if (self.game.selected_tuple == None and self.game.player1.charlocCheck(column, row)):
                    self.status.config(
                        text=f"......Player 1 is Thinking.....", bg=f"{self.game.player1.color}")
                    # whenever he select one of his token
                    if (self.game.player1.charlocCheck(column, row)):
                        character = self.game.player1.getChar(column, row)
                    elif (self.game.player2.charlocCheck(column, row)):
                        character = self.game.player2.getChar(column, row)
                    # it shows all the validated moves which a playr can make
                    validMoves = self.game.validatemove(character)

                    for i in range(len(validMoves)):
                        (x, y) = validMoves[i]
                        checks = self.tiles[x - 1, y - 1]
                        movesboxes.append(checks)
                    # highlighting alll the possible moves with black boundary for better vision.
                    for i in range(len(movesboxes)):
                        self.gamewin.itemconfigure(
                            movesboxes[i], outline="black", width=2)

                    self.game.selected_tuple = (column, row)
                # if a player wants to make a move then he must need to click on otehr tile.
                elif (self.game.selected_tuple != None and (column, row) in self.game.validatemove(
                        self.game.player1.getChar(self.game.selected_tuple[0], self.game.selected_tuple[1]))):
                    #                 print("IF2")
                    self.status.config(
                        text="Player 1 Moved,Player 2 Turn", bg=f"{self.game.player2.color}")
                    # getting the cordinate of other tile
                    (x, y) = self.game.selected_tuple
                    # movind the token from one tile to other using movement function

                    self.game.char_movement((y, x), (row, column))
                    # removing all the highlighhed block after making a move
                    for i in range(self.board_size):
                        for j in range(self.board_size):
```

```
                    for j in range(self.board_size):
                        self.gamewin.itemconfigure(
                            self.tiles[i, j], outline="black", width=0)
                self.game.selected_tuple = None
                self.t = 0
                # Assiging turn to other player if a player made a move.
                time.sleep(1)
                self.t = self.backup
                p1.join()
                p1 = threading.Thread(target=self.timeren)
                thread.sleep(1)
                #                 self.t=20
                p1.start()
                self.move = False

            else:
                #                 print("Else2")
                self.game.selected_tuple = None
                for i in range(self.board_size):
                    for j in range(self.board_size):
                        self.gamewin.itemconfigure(
                            self.tiles[i, j], outline="black", width=0)
        else:
            # similarly if other player want to do same thins
            # all the conditions works similarly.
            #             print("ELSEEEEEEEEEEEEEEEEEEEEEEEEEEEEEe")
            if (self.game.selected_tuple == None and self.game.player2.charlocCheck(column, row)):
                self.status.config(
                    text=f"......Player 2 is Thinking.............", bg=f"{self.game.player2.color}")
                #                 print("ELSEIF")
                if (self.game.player1.charlocCheck(column, row)):
                    #                     print("ELSEIF 2 player if")
                    character = self.game.player1.getChar(column, row)
                elif (self.game.player2.charlocCheck(column, row)):
                    character = self.game.player2.getChar(column, row)
                #                     print("ELSEIF 2 player else")

                validMoves = self.game.validatemove(character)

                for i in range(len(validMoves)):
                    (x, y) = validMoves[i]
                    checks = self.tiles[x - 1, y - 1]
                    movesboxes.append(checks)

                for i in range(len(movesboxes)):
                    self.gamewin.itemconfigure(
                        movesboxes[i], outline="black", width=2)

                self.game.selected_tuple = (column, row)

            elif (self.game.selected_tuple != None and (column, row) in self.game.validatemove(
                    self.game.player2.getChar(self.game.selected_tuple[0], self.game.selected_tuple[1]))):
                #                 print("ELIF")
                self.status.config(
                    text="Player 2 Moved,Player 1 Turn", bg=f"{self.game.player1.color}")
```

```
                    text="Player 2 Moved,Player 1 Turn", bg=f"{self.game.player1.color}")
                (x, y) = self.game.selected_tuple
                self.game.char_movement((y, x), (row, column))

                for i in range(self.board_size):
                    for j in range(self.board_size):
                        self.gamewin.itemconfigure(
                            self.tiles[i, j], outline="black", width=0)
                self.game.selected_tuple = None
                #                 p1.stop()
                #                 self.t=0
                self.t = 0
                time.sleep(1)
                self.t = self.backup
                p1.join()
                del p1
                p1 = threading.Thread(target=self.timeren)
                #                 self.t=20
                #                 time.sleep(1)
                p1.start()
                self.move = True
            else:
                #                 print("Else")
                self.game.selected_tuple = None
                for i in range(self.board_size):
                    for j in range(self.board_size):
                        self.gamewin.itemconfigure(
                            self.tiles[i, j], outline="black", width=0)

        self.update()


if __name__ == "__main__":
    # initializing our main class to play game
    # Parameters:
    # Board size:8,10,16
    # Timelimit : in seconds
    #player1="RED"or "GREEN"
    #player2="RED or GREEN"
    game = HalmaGame(12, 20, "RED", "GREEN")

```