Halma Part0

Youssef Elmahdy

6398550

Instructor: Tayyaba Shaheen

CS 470 - Artificial Intelligence

Northern Arizona University (NAU)

1. Overview:

The HalmaBoard class provides a structure for a simple Halma game board using the tkinter library for GUI. The class has been designed with modularity in mind, encapsulating functionalities like board creation, piece placement, move highlighting, and piece movement. Below is an overview of the key objects and methods:

* **Initialization (\_\_init\_\_)**
  + Sets board size and canvas, creates a grid (create\_grid), and initializes pieces (initialize\_pieces). Manages piece locations (self.pieces) and tracks valid moves.
* **Grid and Piece Setup**
  + **create\_grid**: Draws an 8x8 beige grid.
  + **place\_piece** & **initialize\_pieces**: Position initial game pieces and store them in self.pieces.
* **Move Highlighting (highlight\_moves & clear\_highlights)**
  + Highlights valid moves for a selected piece, ensuring they’re within bounds and unoccupied. Clears highlights when a new piece is selected.
* **Piece Movement (on\_click & move\_piece)**
  + on\_click handles selecting pieces or moves; move\_piece updates a piece’s position, managing its coordinates and data storage.

This structure organizes board creation, piece management, and interactions, making the code easy to extend or modify.

1. Screenshot of initial board: white or black circles in one corner, representing the pieces for one player

A screenshot of a graph

Description automatically generated

1. Screenshots of possible moves after clicking on a piece. Clicking on a circle should hilight that piece, and make squares appear, representing where moves are possible for that piece.

A screenshot of a game

Description automatically generated

1. Screenshots after having moved some pieces. Clicking on a square should move the hilighted piece

A screenshot of a cell phone

Description automatically generated

1. Screenshot of moves available from several consecutive jumps. Start by implementing moves to adjacent squares. After that is working, implement the jumping.

A screenshot of a game

Description automatically generated

1. Code:

import tkinter as tk

class HalmaBoard:

def \_\_init\_\_(self, root, size=8):

self.size = size

self.cell\_size = 50

self.canvas = tk.Canvas(root, width=self.size \* self.cell\_size, height=self.size \* self.cell\_size)

self.canvas.pack()

self.create\_grid()

self.pieces = {}

self.selected\_piece = None

self.valid\_moves = []

self.initialize\_pieces()

def create\_grid(self):

for row in range(self.size):

for col in range(self.size):

x1 = col \* self.cell\_size

y1 = row \* self.cell\_size

x2 = x1 + self.cell\_size

y2 = y1 + self.cell\_size

self.canvas.create\_rectangle(x1, y1, x2, y2, fill='beige')

def place\_piece(self, row, col, color):

x1 = col \* self.cell\_size + 10

y1 = row \* self.cell\_size + 10

x2 = x1 + self.cell\_size - 20

y2 = y1 + self.cell\_size - 20

piece = self.canvas.create\_oval(x1, y1, x2, y2, fill=color)

self.pieces[(row, col)] = piece

def initialize\_pieces(self):

initial\_positions = [

(0, 0), (0, 1), (0, 2), (0, 3),

(1, 0), (1, 1), (1, 2),

(2, 0), (2, 1),

(3, 0)

]

for row, col in initial\_positions:

self.place\_piece(row, col, 'white')

def highlight\_moves(self, row, col):

self.clear\_highlights()

possible\_moves = [(row - 1, col), (row + 1, col), (row, col - 1), (row, col + 1)]

valid\_moves = []

for r, c in possible\_moves:

if 0 <= r < self.size and 0 <= c < self.size and (

r, c) not in self.pieces:

x1 = c \* self.cell\_size

y1 = r \* self.cell\_size

x2 = x1 + self.cell\_size

y2 = y1 + self.cell\_size

move\_id = self.canvas.create\_rectangle(x1, y1, x2, y2, outline='green', width=2)

valid\_moves.append((r, c, move\_id))

self.valid\_moves = valid\_moves

def clear\_highlights(self):

for move in self.valid\_moves:

self.canvas.delete(move[2])

self.valid\_moves = []

def on\_click(self, event):

row, col = event.y // self.cell\_size, event.x // self.cell\_size

if (row, col) in self.pieces:

if self.selected\_piece:

self.canvas.itemconfig(self.selected\_piece, outline="")

self.selected\_piece = self.pieces[(row, col)] # Select the new piece

self.canvas.itemconfig(self.selected\_piece, outline="red", width=3)

self.highlight\_moves(row, col)

# If a valid move is clicked, move the piece

else:

for move in self.valid\_moves:

if (row, col) == (move[0], move[1]):

self.move\_piece(self.selected\_piece, (move[0], move[1]))

break

def move\_piece(self, piece\_id, to\_pos):

from\_pos = [pos for pos, p\_id in self.pieces.items() if p\_id == piece\_id][0]

row, col = to\_pos

x1 = col \* self.cell\_size + 10

y1 = row \* self.cell\_size + 10

x2 = x1 + self.cell\_size - 20

y2 = y1 + self.cell\_size - 20

self.canvas.coords(piece\_id, x1, y1, x2, y2)

del self.pieces[from\_pos]

self.pieces[to\_pos] = piece\_id

self.clear\_highlights()

root = tk.Tk()

game\_board = HalmaBoard(root)

game\_board.canvas.bind("<Button-1>", game\_board.on\_click)

root.mainloop()