Halma Part1

Youssef Elmahdy

6398550

Instructor: Tayyaba Shaheen

CS 470 - Artificial Intelligence

Northern Arizona University (NAU)

1. Functionality Table:

|  |  |  |
| --- | --- | --- |
| Functionality | % Complete | Notes |
| Graphical board display | 100% | Generates a nicely formatted GUI with Tkinter. |
| Board updating | 100% | Updates board smoothly, including move highlights. |
| Move generator | 100% | Generates legal moves including jumps accurately. |
| Win detector | 100% | Detects win conditions when a player occupies the goal area with the required pieces. |
| Demos | 100% | All screenshots included, with annotations and clear labeling. |

1. Demos:
2. Graphical board display:

A screenshot of a game

Description automatically generated

The image shows a well-formatted graphical display of the Halma game board, with an 8x8 grid using alternating light gray and beige cells for easy visibility. White and black pieces are placed in their respective starting corners, clearly distinguished by color representing 2 humans. Below the board, a status bar displays the current turn, each player’s score, and a countdown timer for move time limits. The interface highlights valid moves and selected pieces during play, and a win notification will appear when a player meets the win conditions.

A screenshot of a game

Description automatically generated

A message dialog informs that "White ran out of time," indicating the time limit for the current turn was exceeded. Below the board, the status bar updates dynamically, showing the turn, each player’s score, and a countdown timer, which here shows "Time Left: 0s."

1. Board updating:

A screenshot of a game

Description automatically generated

The image demonstrates the Board updating functionality in the Halma game GUI. The 8x8 board dynamically highlights the selected piece with a red outline and displays possible moves in green-outlined squares, helping the player choose legal moves smoothly.

1. Move generator:

A screenshot of a game

Description automatically generated

The image illustrates the Move generator functionality in the Halma game. The selected black piece is highlighted with a red outline, and the possible moves generated by the game are displayed as green-outlined squares. These moves include adjacent steps as well as potential jumps over nearby pieces, demonstrating the generator’s capability to provide a complete and correct list of legal moves based on the current board state and the active player’s turn.

1. Win detector:

A screenshot of a game

Description automatically generated

The image demonstrates the Win detector functionality in the Halma game. A dialog box appears, displaying the message "White wins with a score of 5!" to indicate that the white player has successfully met the win condition by placing the required number of pieces in the target area.

1. Code:

import tkinter as tk

from tkinter import messagebox

class Position:

def \_\_init\_\_(self, row, col, color, canvas, cell\_size):

self.row = row

self.col = col

self.color = color

self.canvas = canvas

self.cell\_size = cell\_size

self.piece\_id = self.create\_piece()

def create\_piece(self):

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

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

x2 = x1 + self.cell\_size - 20

y2 = y1 + self.cell\_size - 20

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

def move\_to(self, row, col):

self.row, self.col = row, col

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

def set\_outline(self, color="red", width=3):

self.canvas.itemconfig(self.piece\_id, outline=color, width=width)

def clear\_outline(self):

self.canvas.itemconfig(self.piece\_id, outline="", width=1)

def delete(self):

self.canvas.delete(self.piece\_id)

class HalmaBoard:

def \_\_init\_\_(self, root, size=8, seconds\_limit=10):

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.seconds\_limit = seconds\_limit

self.time\_remaining = self.seconds\_limit

self.timer\_id = None

self.create\_grid()

self.pieces = {}

self.selected\_piece = None

self.valid\_moves = []

self.current\_turn = 'white'

self.white\_score = 0

self.black\_score = 0

self.initialize\_pieces()

self.status\_bar = tk.Label(root, text="White's Turn | White Score: 0 | Black Score: 0 | Time Left: 10s",

font=("Arial", 14))

self.status\_bar.pack()

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

self.start\_timer()

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

color = 'beige' if (row + col) % 2 == 0 else 'lightgray'

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

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

position = Position(row, col, color, self.canvas, self.cell\_size)

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

def initialize\_pieces(self):

white\_positions = [

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

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

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

(3, 0)

]

black\_positions = [

(7, 7), (7, 6), (7, 5), (7, 4),

(6, 7), (6, 6), (6, 5),

(5, 7), (5, 6),

(4, 7)

]

for row, col in white\_positions:

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

for row, col in black\_positions:

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

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

self.clear\_highlights()

possible\_moves = self.get\_possible\_moves(row, col)

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 get\_possible\_moves(self, row, col):

moves = []

directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]

for dr, dc in directions:

nr, nc = row + dr, col + dc

if 0 <= nr < self.size and 0 <= nc < self.size and (nr, nc) not in self.pieces:

moves.append((nr, nc))

elif 0 <= nr < self.size and 0 <= nc < self.size and (nr, nc) in self.pieces:

jr, jc = nr + dr \* 2, nc + dc \* 2

if 0 <= jr < self.size and 0 <= jc < self.size and (jr, jc) not in self.pieces:

moves.append((jr, jc))

return 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 and self.pieces[(row, col)].color == self.current\_turn:

if self.selected\_piece:

self.selected\_piece.clear\_outline()

self.selected\_piece = self.pieces[(row, col)]

self.selected\_piece.set\_outline("red", 3)

self.highlight\_moves(row, col)

elif self.selected\_piece:

for move in self.valid\_moves:

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

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

self.switch\_turn()

break

def move\_piece(self, position, to\_pos):

from\_pos = (position.row, position.col)

position.move\_to(\*to\_pos)

del self.pieces[from\_pos]

self.pieces[to\_pos] = position

position.clear\_outline()

self.clear\_highlights()

self.update\_score()

self.check\_for\_win()

def switch\_turn(self):

self.current\_turn = 'black' if self.current\_turn == 'white' else 'white'

self.reset\_timer() # Reset the timer for the new turn

self.update\_status()

def start\_timer(self):

if self.time\_remaining > 0:

self.time\_remaining -= 1

self.update\_status()

self.timer\_id = self.canvas.after(1000, self.start\_timer)

else:

messagebox.showinfo("Time's up!", f"{self.current\_turn.capitalize()} ran out of time!")

self.switch\_turn()

def reset\_timer(self):

if self.timer\_id:

self.canvas.after\_cancel(self.timer\_id) # Stop the current timer

self.time\_remaining = self.seconds\_limit

self.start\_timer() # Start a new timer for the next turn

def update\_score(self):

white\_goal = [(7, 7), (7, 6), (7, 5), (6, 7), (6, 6)]

black\_goal = [(0, 0), (0, 1), (1, 0), (1, 1)]

self.white\_score = sum(1 for pos in white\_goal if pos in self.pieces and self.pieces[pos].color == 'white')

self.black\_score = sum(1 for pos in black\_goal if pos in self.pieces and self.pieces[pos].color == 'black')

def update\_status(self):

self.status\_bar.config(text=f"{self.current\_turn.capitalize()}'s Turn | White Score: {self.white\_score} | "

f"Black Score: {self.black\_score} | Time Left: {self.time\_remaining}s")

def check\_for\_win(self):

if self.white\_score >= 5:

messagebox.showinfo("Game Over", f"White wins with a score of {self.white\_score}!")

self.canvas.unbind("<Button-1>")

self.stop\_timer()

elif self.black\_score >= 5:

messagebox.showinfo("Game Over", f"Black wins with a score of {self.black\_score}!")

self.canvas.unbind("<Button-1>")

self.stop\_timer()

def stop\_timer(self):

if self.timer\_id:

self.canvas.after\_cancel(self.timer\_id)

self.timer\_id = None

root = tk.Tk()

root.title("Halma")

game\_board = HalmaBoard(root, seconds\_limit=15) # Set a 10-second limit for demonstration

root.mainloop()