Halma Part1

Youssef Elmahdy

6398550

Instructor: Tayyaba Shaheen

CS 470 - Artificial Intelligence

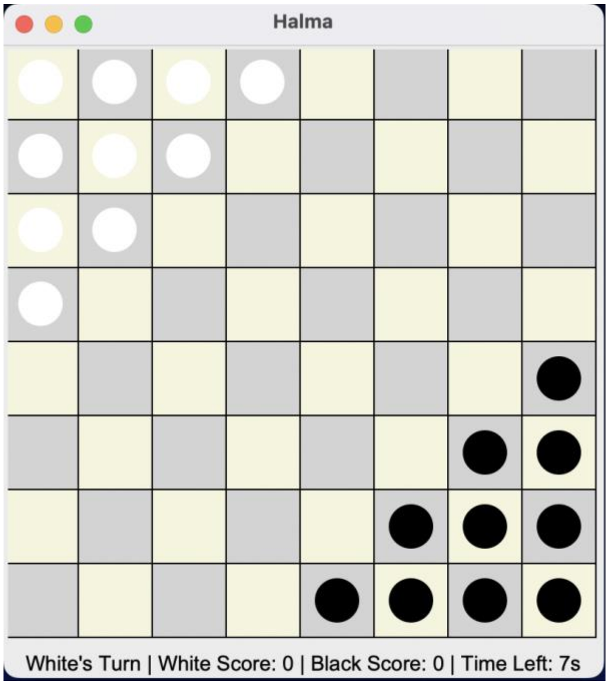
Northern Arizona University (NAU)

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.

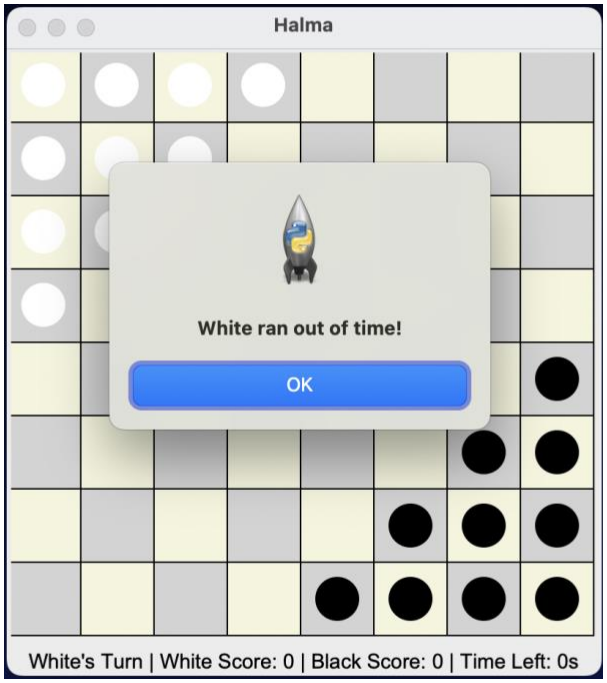
2) Demos:

a) Graphical board display:



White's Turn | White Score: 0 | Black Score: 0 | Time Left: 7s

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.

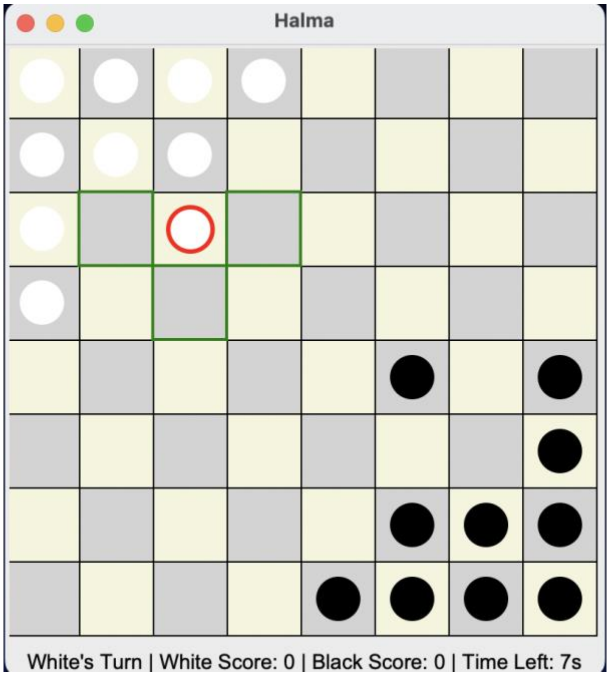


White ran out of time!

White's Turn | White Score: 0 | Black Score: 0 | Time Left: 0s

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."

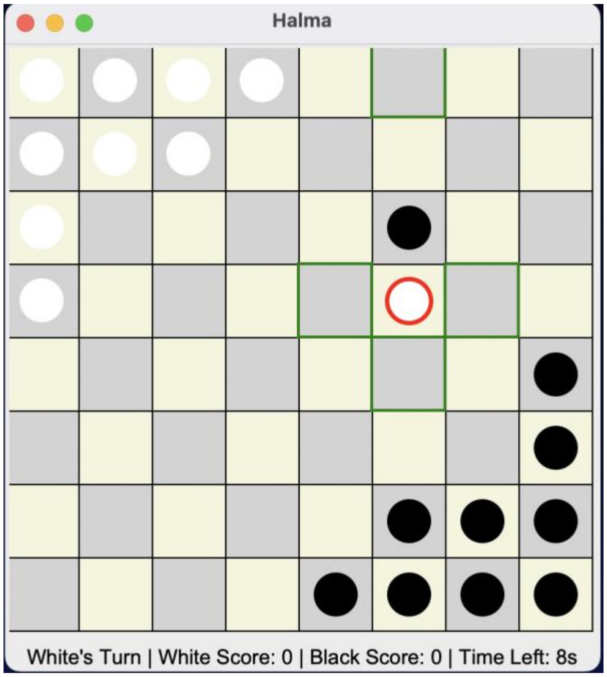
b) Board updating:



White's Turn | White Score: 0 | Black Score: 0 | Time Left: 7s

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.

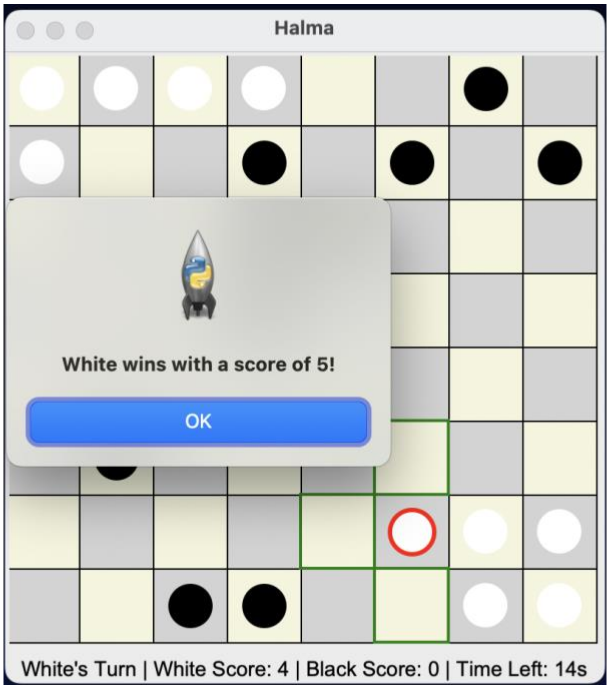
c) Move generator:



White's Turn | White Score: 0 | Black Score: 0 | Time Left: 8s

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.

d) Win detector:



White wins with a score of 5!

White's Turn | White Score: 4 | Black Score: 0 | Time Left: 14s

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.

3) 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()
```