```
import tkinter as tk

GRID_CELL_SIZE = 50
GAME_PIECE_PADDING = 5
MOVE_DIRS = [(1, 0), (-1, 0), (0, 1), (0, -1), (-1, 1), (1, 1),
(-1, -1), (1, -1)]

class GameManager:
    def __init__(self, board_size):
        self.board = {}
        self.board_size = board_size
        self.green_score = 0
        self.red_score = 0
        self.green_to_win = 0
        self.red_to_win = 0
        self.selected_piece = None
        self.green_goals = []
        self.red_goals = []
        self.turn = "red"

        for row in range(board_size):
            for col in range(board_size):
                self.board[(row, col)] = 'empty'

        half_board_size = board_size // 2
        for row in range(half_board_size):
            for col in range(half_board_size - row):
                self.red_goals.append((row, col))
                self.board[(row, col)] = 'red'
                self.red_to_win += 1

        for row in range(board_size - 1, half_board_size - 1, -
1):
            for col in range(board_size - 1, (half_board_size -
1) + (board_size - 1 - row), - 1):
                self.green_goals.append((row, col))
                self.board[(row, col)] ='green'
                self.green_to_win += 1

        self.score_label_red = tk.Label(tk_root, text=f"Red
Score: {self.red_score}", fg="red")
```

```
        self.score_label_red.grid(row=0, column=board_size + 1,
padx=10)
        self.goals_to_win_label_red = tk.Label(tk_root,
text=f"Goals until Red Wins: {self.red_to_win}", fg="red")
        self.goals_to_win_label_red.grid(row=1,
column=board_size + 1, padx=10)
        self.score_label_green = tk.Label(tk_root, text=f"Green
Score: {self.green_score}", fg="green")
        self.score_label_green.grid(row=2, column=board_size +
1, padx=10)
        self.goals_to_win_label_green = tk.Label(tk_root,
text=f"Goals until Green Wins: {self.green_to_win}", fg="green")
        self.goals_to_win_label_green.grid(row=3,
column=board_size + 1, padx=10)
        self.curr_player = tk.Label(tk_root,
text=f"{self.turn}'s turn", font=("Helvetica", 16),
fg=self.turn)
        self.curr_player.grid(row=4, column=self.board_size + 1,
padx=10)

        self.board_display = GameBoard(self.board, self,
board_size)

    def start_move(self, cell):
        self.selected_piece = cell
        possible_moves = MoveGenerator.get_moves(cell,
self.board, self.board_size)
        self.board_display.show_moves(cell, possible_moves)

    def execute_move(self, dest_cell):
        if self.board[self.selected_piece] == "red":
            if self.selected_piece in self.green_goals:
                self.red_score -= 1
                self.red_to_win += 1
            if dest_cell in self.green_goals:
                self.red_score += 1
                self.red_to_win -= 1
            self.turn = "green"

        elif self.board[self.selected_piece] == "green":
            if self.selected_piece in self.red_goals:
                self.green_score -= 1
                self.green_to_win += 1
            if dest_cell in self.red_goals:
                self.green_score += 1
                self.green_to_win -= 1
            self.turn = "red"

        self.score_label_red.config(text=f"Red Score: 
{self.red_score}")
        self.goals_to_win_label_red.config(text=f"Goals until 
Red Wins: {self.red_to_win}")
        self.score_label_green.config(text=f"Green Score: 
{self.green_score}")
        self.goals_to_win_label_green.config(text=f"Goals until 
Green Wins: {self.green_to_win}")
        self.curr_player.config(text=f"{self.turn}'s turn", 
fg=self.turn)

        self.board[dest_cell] = self.board[self.selected_piece]
        self.board[self.selected_piece] = "empty"
        self.selected_piece = None
        self.exit_move()
        self.board_display.update(self.board)

        self.check_winner()

    def reset_game(self):
        self.play_again_button.destroy()
        self.__init__(self.board_size)

    def show_play_again_button(self):
        self.play_again_button = tk.Button(tk_root, text="Play 
Again", command=self.reset_game)
        self.play_again_button.grid(row=5, 
column=self.board_size + 1, padx=10)

    def check_winner(self):
        if self.red_to_win == 0:
            self.curr_player.config(text=f"red wins!", fg="red")
            self.show_play_again_button()
        elif self.green_to_win == 0:
            self.curr_player.config(text=f"green wins!",
fg="green")
            self.show_play_again_button()
        else:
            return

        for cell in self.board_display.display_board.values():
            cell.canvas.unbind("<Button>")

    def exit_move(self):
        self.board_display.exit_move_state(self.turn)
        self.selected_piece = None

class GameCell:
    def __init__(self, row, col, game_board, manager):
        self.pos = (row, col)
        self.board = game_board
        self.canvas = tk.Canvas(tk_root, width = GRID_CELL_SIZE,
                                        height = GRID_CELL_SIZE,
bg='burlywood1',

highlightbackground='black')
        self.state = "empty"
        self.canvas.grid(row=row+1, column=col+1)
        self.highlighted = False
        self.green_goal = False
        self.red_goal = False
        self.manager = manager

    def clear(self):
        self.canvas.delete("all")
        self.canvas.unbind('<Button>')
        self.state = "empty"

    def set_red_state(self):
        self.canvas.create_oval(GAME_PIECE_PADDING,
GAME_PIECE_PADDING,
                        GRID_CELL_SIZE - GAME_PIECE_PADDING,
GRID_CELL_SIZE - GAME_PIECE_PADDING,
                                    fill = "black")
        self.state = "red"

    def set_red_goal(self):
        self.canvas.config(bg="red")
        self.red_goal = True

    def set_green_goal(self):
        self.canvas.config(bg="green")
        self.green_goal = True

    def set_green_state(self):
        self.canvas.create_oval(GAME_PIECE_PADDING,
GAME_PIECE_PADDING,
                        GRID_CELL_SIZE - GAME_PIECE_PADDING,
GRID_CELL_SIZE - GAME_PIECE_PADDING,
                                    fill = "white")
        self.state = "green"

    def make_moveable(self):
        self.canvas.bind('<Button>', lambda event:
manager.start_move(self.pos))

    def highlight(self):
        if self.state == "empty":
            dot_size = 10
            center_x = GRID_CELL_SIZE / 2
            center_y = GRID_CELL_SIZE / 2
            self.canvas.create_oval(center_x - dot_size / 2,
center_y - dot_size / 2,
                                center_x + dot_size / 2,
center_y + dot_size / 2,
                                fill="black")
            self.canvas.bind('<Button>', lambda event:
self.manager.execute_move(self.pos))
        elif self.state == "red":
            self.canvas.create_oval(GAME_PIECE_PADDING,
GAME_PIECE_PADDING,
                                    GRID_CELL_SIZE -
GAME_PIECE_PADDING, GRID_CELL_SIZE - GAME_PIECE_PADDING,
                                        fill = "black",
outline="yellow", width=3)
```

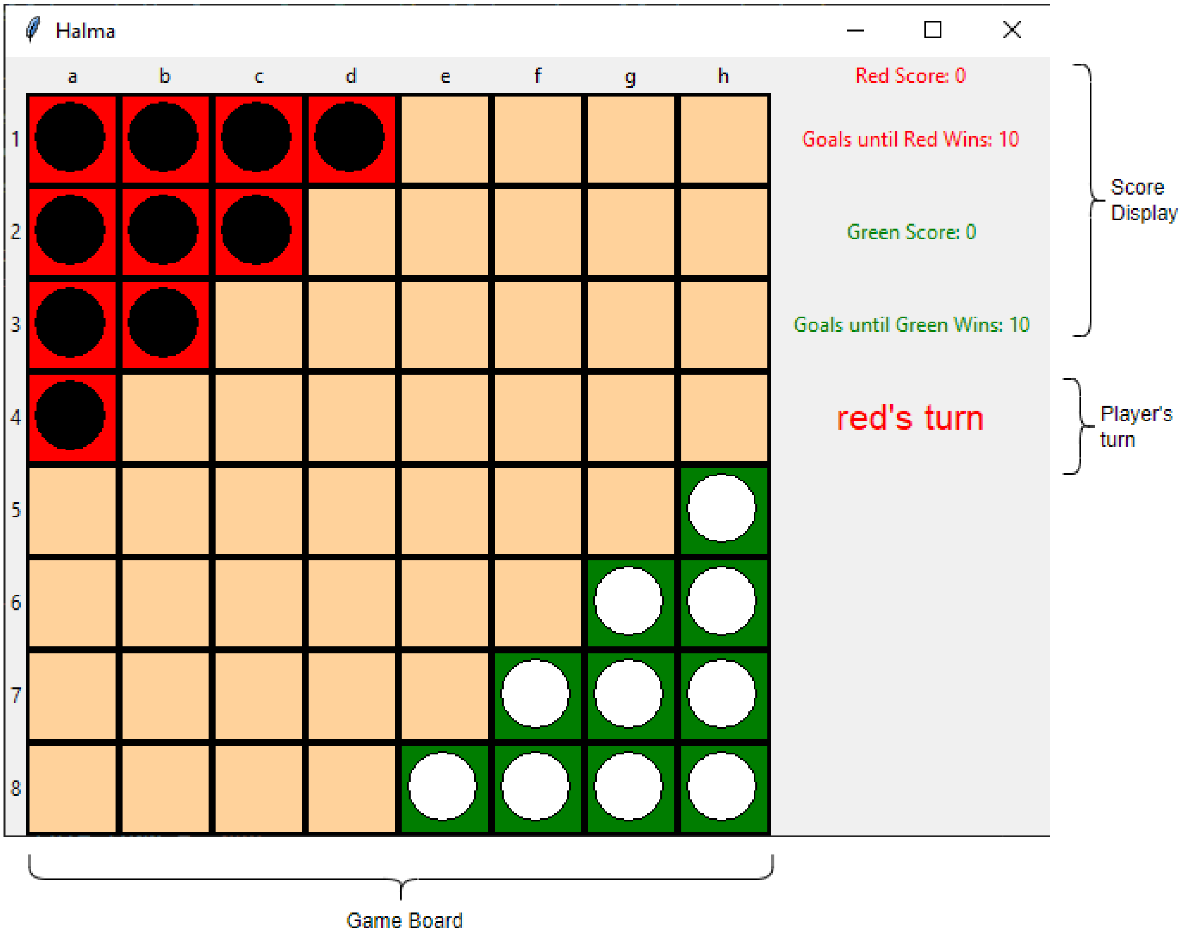
```
            self.canvas.bind('<Button>', lambda event:
manager.exit_move())
        else:
            self.canvas.create_oval(GAME_PIECE_PADDING,
GAME_PIECE_PADDING,
                                    GRID_CELL_SIZE -
GAME_PIECE_PADDING, GRID_CELL_SIZE - GAME_PIECE_PADDING,
                                        fill = "white",
outline="yellow", width=3)
            self.canvas.bind('<Button>', lambda event:
manager.exit_move())
        self.highlighted = True

    def unbind_click(self):
        self.canvas.unbind('<Button>')

class GameBoard:
    def __init__(self, board, manager, board_size):
        self.display_board = {}
        self.manager = manager
        self.red_score = 0
        self.green_score = 0
        self.turn = "red"
        col_labels = [chr(ord('a') + num) for num in
range(board_size)]

        for index, label in enumerate(col_labels):
            tk.Label(tk_root, text=label).grid(row=0,
column=index + 1)
        for index in range(board_size):
            tk.Label(tk_root, text=index + 1).grid(row=index +
1, column=0)
        for cell, state in board.items():
            row, col = cell
            self.display_board[cell] = GameCell(row, col, self,
self.manager)
            if state == 'red':
                self.display_board[cell].set_red_state()
                self.display_board[cell].set_red_goal()
                self.display_board[cell].make_moveable()
            elif state == 'green':
                self.display_board[cell].set_green_state()
                self.display_board[cell].set_green_goal()

    def exit_move_state(self, turn):
        for cell in self.display_board.values():
            state = cell.state
            if cell.highlighted:
                cell.highlighted = False
                cell.clear()
            if state == "red":
                cell.set_red_state()
                if turn == "red":
                    cell.make_moveable()
                else:
                    cell.unbind_click()
            elif state == "green":
                cell.set_green_state()
                if turn == "green":
                    cell.make_moveable()
                else:
                    cell.unbind_click()

    def show_moves(self, piece, moves):
        for cell, canvas in self.display_board.items():
            if cell in moves or cell == piece:
                canvas.highlight()
            else:
                canvas.unbind_click()

    def update(self, new_board):
        for cell, canvas in self.display_board.items():
            if new_board[cell] == "empty":
                canvas.clear()
            elif new_board[cell] == "red":
                canvas.set_red_state()
            else:
                canvas.set_green_state()


    def is_goal(self, cell):
        if self.display_board[cell].green_goal:
            return "green"
        elif self.display_board[cell].red_goal:
            return "red"
        else:
            return None

class MoveGenerator:
    def get_moves(cell, game_board, board_size):
        move_stack = [[cell, [cell]]]
        valid_moves = []

        while move_stack:
            curr_cell, path = move_stack.pop()
            row, col = curr_cell

            for row_change, col_change in MOVE_DIRS:
                move = (row + row_change, col + col_change)
                if MoveGenerator.valid_cell(move[0], move[1],
board_size) and move not in path:
                    if game_board[move] == "red" or
game_board[move] == "green":
                        jump_move = (move[0] + row_change,
move[1] + col_change)
                        if
MoveGenerator.valid_cell(jump_move[0], jump_move[1], board_size)
and game_board[jump_move] == "empty" and jump_move not in path:
                            valid_moves.append(jump_move)
                            if
MoveGenerator.check_for_surrounding_piece(jump_move, game_board,
board_size):

move_stack.append([(jump_move[0], jump_move[1]), path +
[jump_move]])
                    elif curr_cell == cell:
                        valid_moves.append(move)

        return valid_moves

    def check_for_surrounding_piece(cell, board, board_size):
        row, col = cell
        for row_change, col_change in MOVE_DIRS:
            adj_cell = (row + row_change, col + col_change)
            if MoveGenerator.valid_cell(adj_cell[0],
adj_cell[1], board_size) and \
            (board[(adj_cell)] == "red" or board[adj_cell] ==
"green"):
                return True
        return False

    def valid_cell(row, col, board_size):
        return row < board_size and row > -1 and col <
board_size and col > -1


tk_root = tk.Tk()

tk_root.title("Halma")
manager = GameManager(8)
tk_root.mainloop()
```

Functionality	**% Complete**	**Notes**
Graphical board display	100%	Gui generates neatly, displays necessary scores and win status appropriately
Board updating	100%	Spaces and selected pieces

		are highlighted appropriately when executing moves, pieces move appropriately when move is executed
Move generator	100%	Move generator adequately generates moves based on a piece position, handling edge cases.
Win detector	100%	Detects wins correctly, and displays a winning message appropriately
Extra functionality	90%	Added functionality to see how many goals are left until a given player wins, and a ‘play again’ option once a player wins. Styling for these functionalities could be improved, as they are functional but could be improved in style
Demos	100%	Screenshots included with labeling and annotations

See demos on following pages

Game Start



	a	b	c	d	e	f	g	h
1	●	●	●	●				
2	●	●	●					
3	●	●						
4	●							
5								○
6							○	○
7						○	○	○
8					○	○	○	○

Red Score: 0

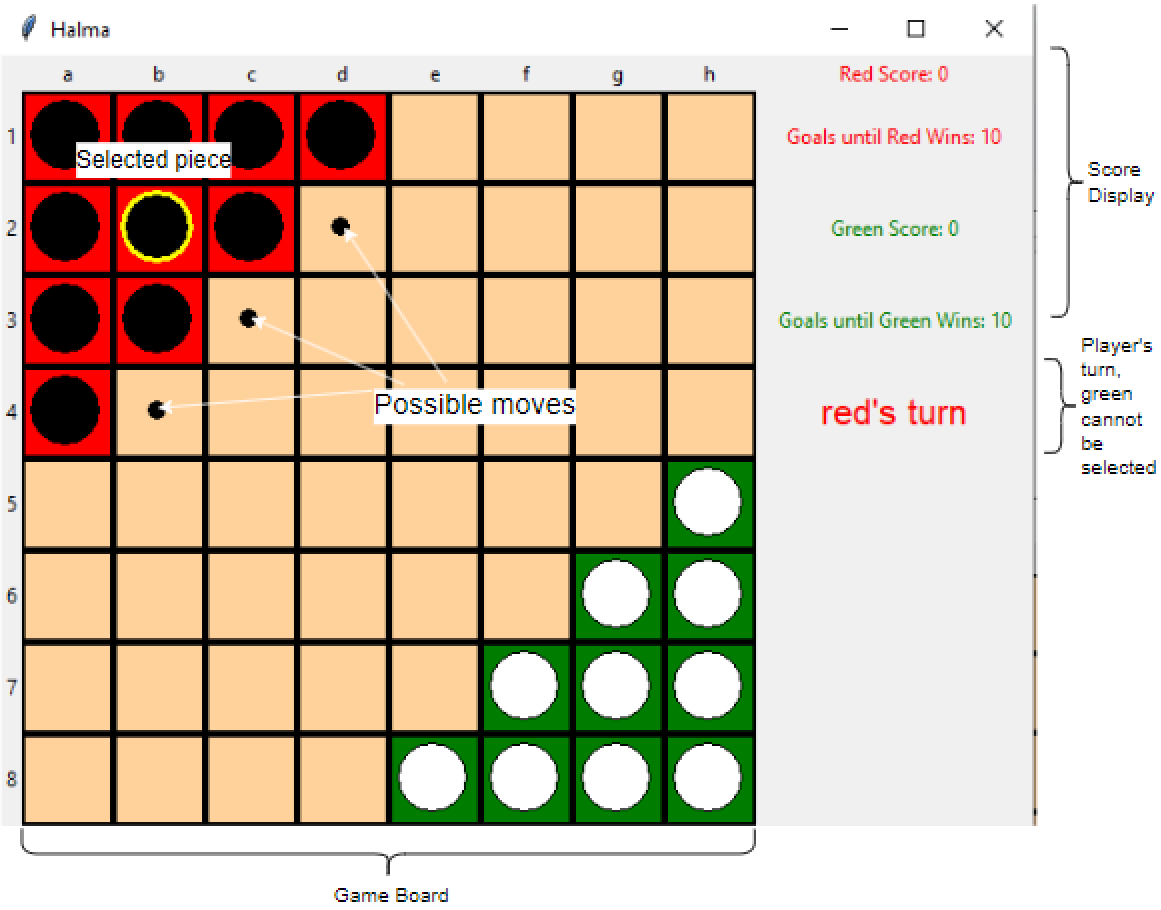
Goals until Red Wins: 10

Green Score: 0

Goals until Green Wins: 10

red's turn

Red Starts → Selected piece b2



Halma

	a	b	c	d	e	f	g	h
1	●	●	●	●				
2	●	●	●	•				
3	●	●	•					
4	●	•						
5								○
6							○	○
7						○	○	○
8					○	○	○	○

Red Score: 0

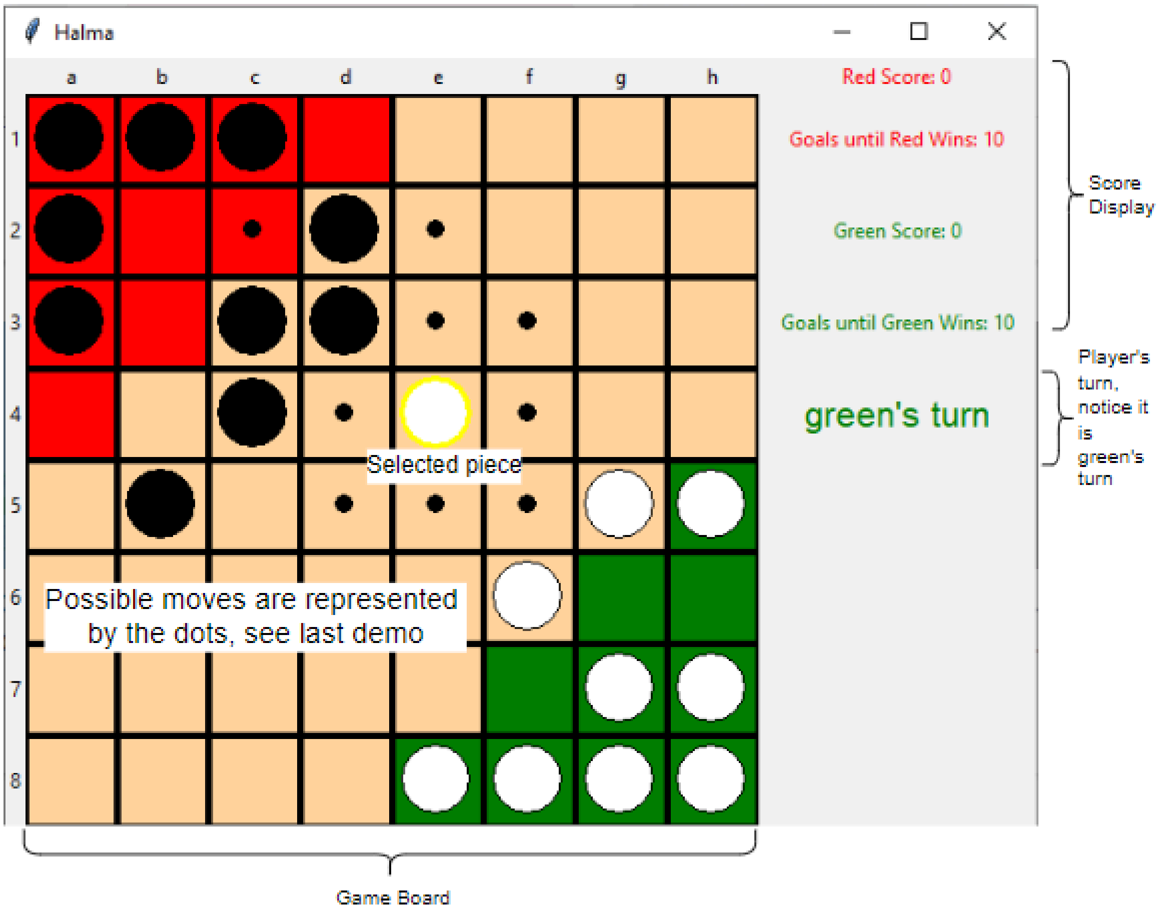
Goals until Red Wins: 10

Green Score: 0

Goals until Green Wins: 10

red's turn

After a few moves → Green’s turn, selected piece e4



Red Score: 0

Goals until Red Wins: 10

Green Score: 0

Goals until Green Wins: 10

green's turn

Selected piece

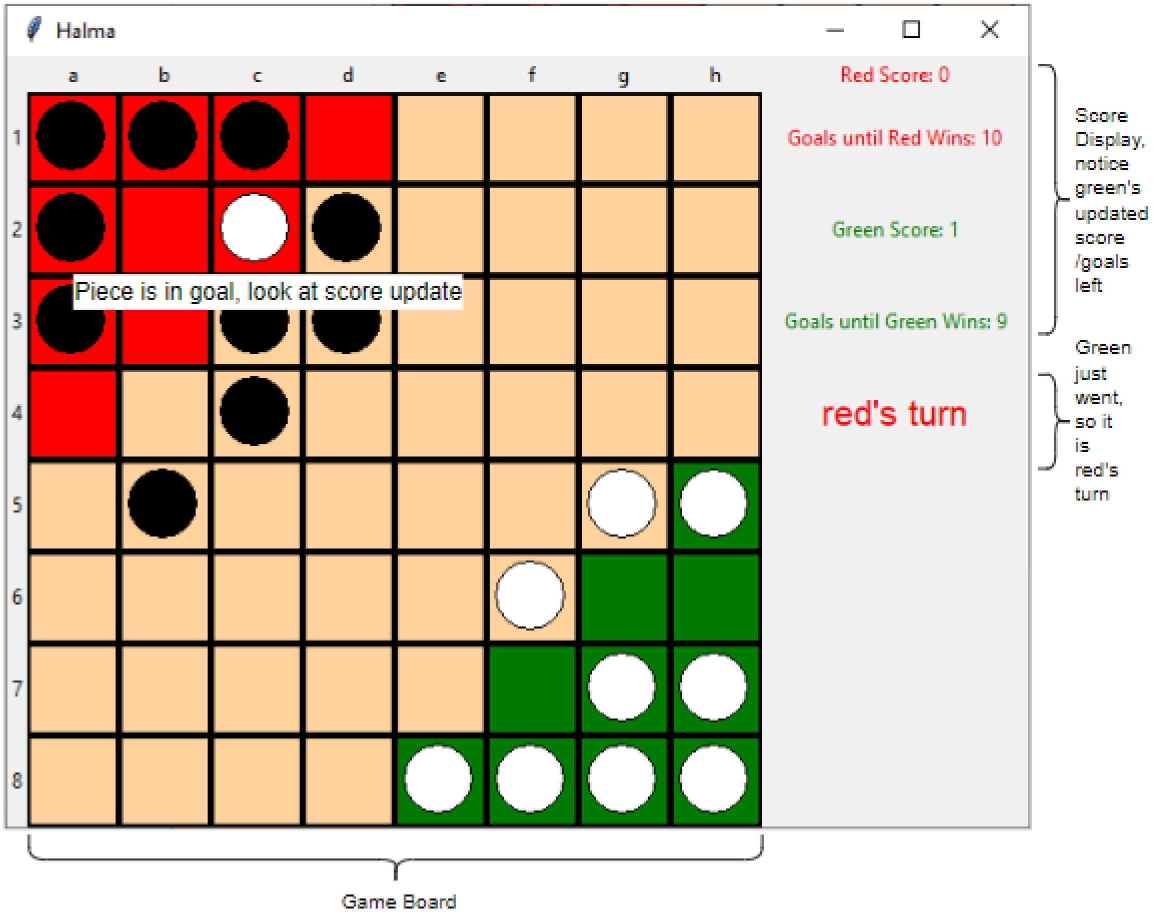
Possible moves are represented by the dots, see last demo

Score Display

Player's turn, notice it is green's turn

Game Board

Move from previous screenshot → e4 moved to c2



	a	b	c	d	e	f	g	h
1	●	●	●					
2	●		○	●				
3	●		●	●				
4			●					
5		●					○	○
6						○		
7							○	○
8					○	○	○	○

Piece is in goal, look at score update

Red Score: 0

Goals until Red Wins: 10

Green Score: 1

Goals until Green Wins: 9

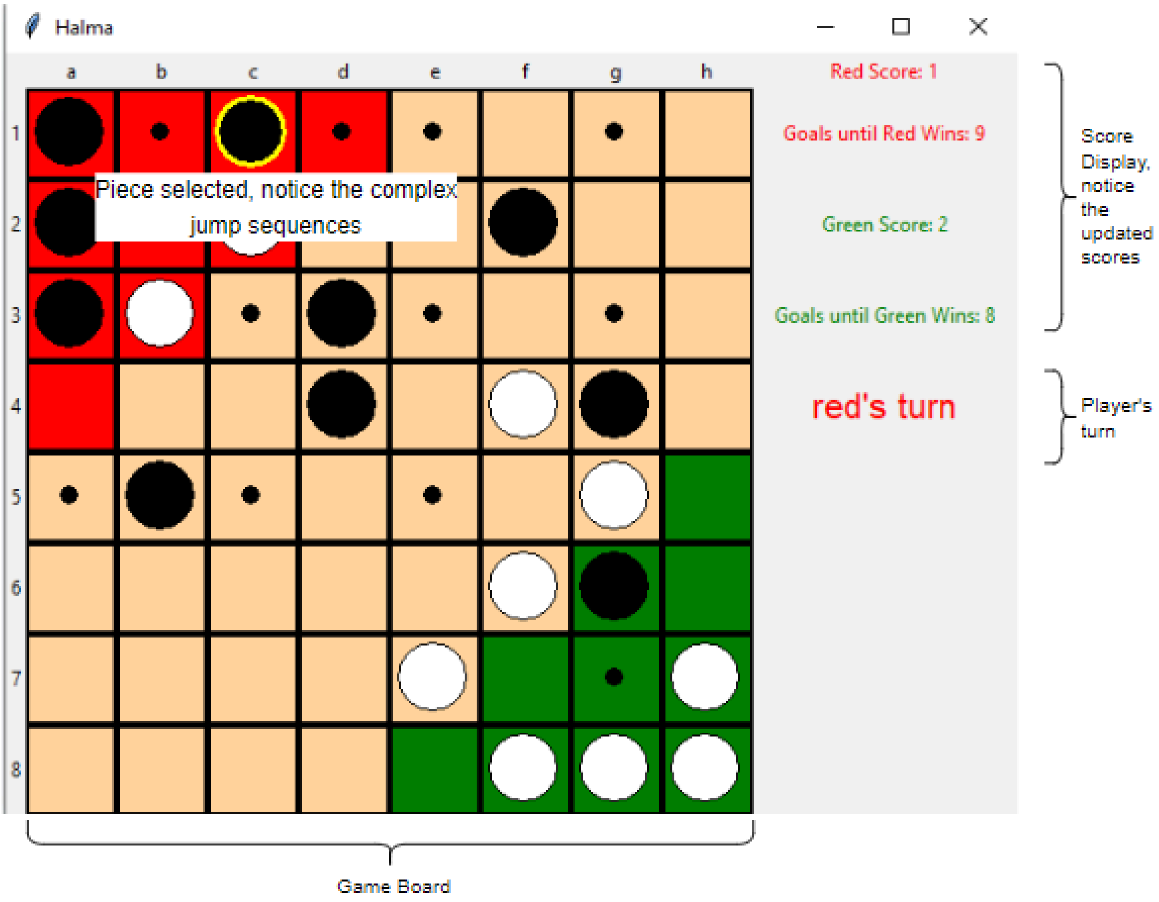
red's turn

Score Display, notice green's updated score /goals left

Green just went, so it is red's turn

Game Board

Further into game → Complex jump sequence selecting piece c1



Piece selected, notice the complex jump sequences

Red Score: 1

Goals until Red Wins: 9

Green Score: 2

Goals until Green Wins: 8

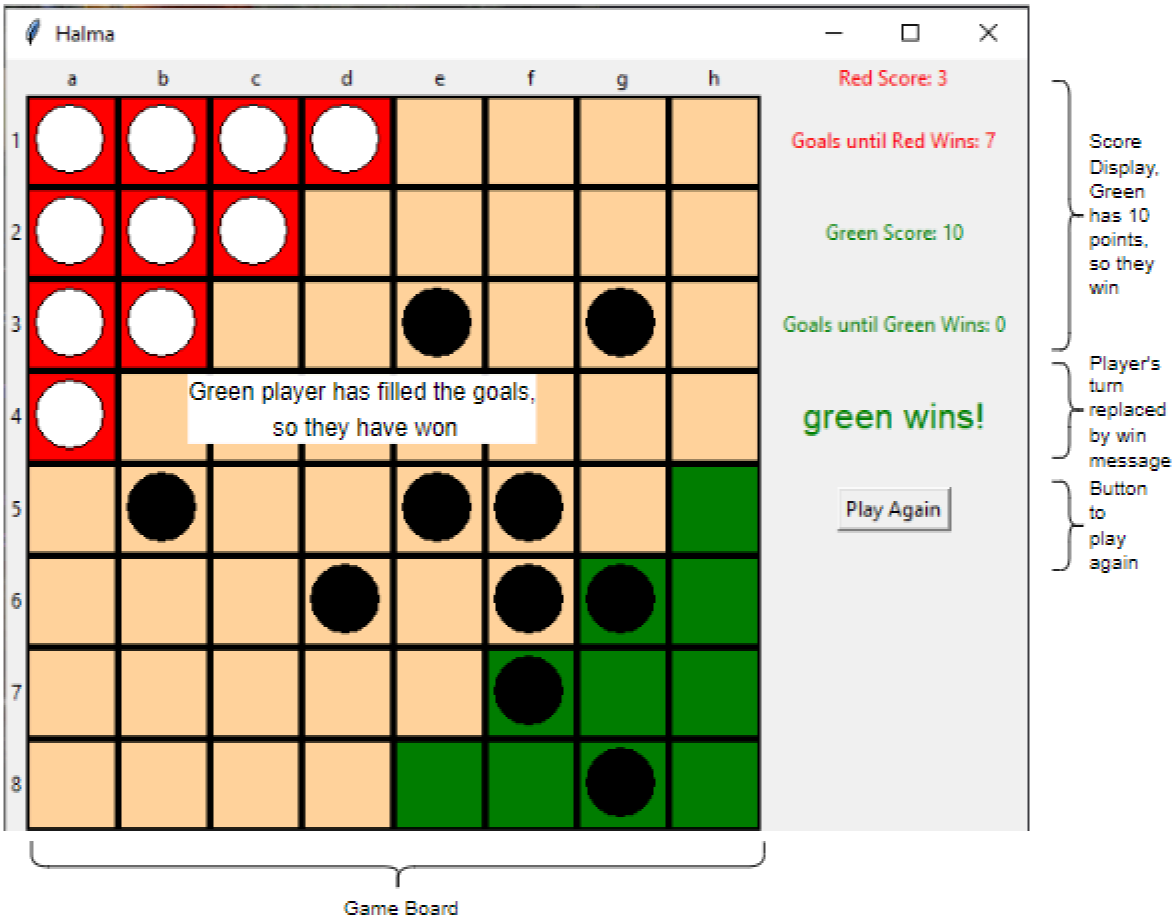
red's turn

Score Display, notice the updated scores

Player's turn

Game Board

Game is finished, green has won



	a	b	c	d	e	f	g	h
1	○	○	○	○				
2	○	○	○					
3	○	○			●		●	
4	○							
5		●			●	●		
6				●		●	●	
7						●		
8							●	

Green player has filled the goals, so they have won

Red Score: 3

Goals until Red Wins: 7

Green Score: 10

Goals until Green Wins: 0

green wins!

Score Display, Green has 10 points, so they win

Player's turn replaced by win message

Button to play again

Game Board