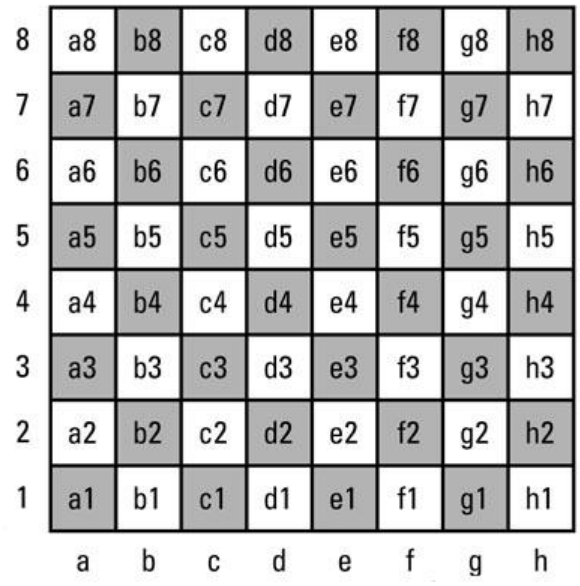
Lab 7a Planning Document

Pablo Barrientos
Andrew Woodworth
Carlo Stafford
Alec Estrada

Activity 1: Create a checkers-like game using a list of lists. The purpose of this assignment is to learn and get used to a list of lists.
Constraints: Use a list of lists. Cannot move on a light square.

User instructions: *Use this image below for reference:*



1. At any point the game can be stopped by inputting ‘stop’
2. Select a **dark** square by inputting board coordinates (ex: “e3”). This will be the piece you want to move.
3. Next, select and input any **dark** square to move your piece to (ex: “d4”). Your piece will now be moved.
4. Now you can repeat steps 2 & 3 until you enter ‘stop’

(Brainstorm General steps):

- First we make a “board” by creating a list of lists that holds our pieces
- Next we make a list of valid moves (all dark square board coordinates)
- Next we print the board
- User inputs move choice
- Determine if input is valid by comparing it to our valid moves list
- If not (Light), output error
- If valid (Dark), slice input into two components so we can use it to refer to the board list of pieces
- Convert the character component of the input to an integer
- Compare converted input to the board and determine if a piece exists in that coordinate
- If not, ask the user to input a different coordinate
- If yes, ask the user where they would like to move the piece
- Determine if their input corresponds to a dark square
- If not, ask them to select a different square to move to
- If yes, replace the selected coordinate in the board list with the piece held in user’s initial position and replace the initial position with an “empty” square string
- Loop back to when the board is printed

Variable list names:

board = list of pieces and empty square that will be manipulated throughout the program

valid_moves = List containing dark square coordinates, which are “valid moves”

Variable names:

p1 = holds special character value for our bottom pieces for when board is displayed

p2 = holds special character value for our top pieces for when board is displayed

row = this variable holds the lists in board in our print_board function

column = the variable holds the elements in row in our print_board function

board = holds the value of our board list in our print_board function

pos1 = this variable is accepted as an argument for our game_loop function. It then holds the user’s initial input once the game starts. This variable is then used in our on_board function.

pos2 = this variable holds the value of the user’s input that is given to our light_square function.

pos3 = this variable holds the value of the user’s input for the empty square check portion in our on_board function.

pos4 = holds the user's input for our piece movement portion in our on_board function
column_pos1 = holds the converted integer value of the letter substring in user input in our game_loop function
row_pos1 = holds the converted integer value of the number substring inputted by the user in game_loop
column_pos2 = holds the converted integer value of the letter substring in user input in light_square
row_pos2 = holds the converted integer value of the number substring inputted by the user in light_square
column_pos3 = holds the converted integer value of the letter substring in user input in on_board
row_pos3 = holds the converted integer value of the number substring inputted by the user in on_board
column_pos4 = holds the converted integer value of the letter substring in user input in on_board
row_pos4 = holds the converted integer value of the number substring inputted by the user in on_board

Function names:
game_loop = our main while loop that prints the board and prompts the user to select the piece they want to move
on_board = this function checks if the user inputs a valid square then prompts the user for where they want to move the piece.
stop = this function stops the program if called
light_square = a function that is called if the user selects a light square at any time

Code steps:

1. Create a list named valid_moves that holds strings that correspond to dark square coordinates on the game board (a1, a3, a5, etc)
2. Create a list of 8 lists to represent our game board. Sub lists are our rows and the elements in the lists are our columns. The board will hold special characters held in 'p1' and 'p2' to represent game pieces as well as periods to represent empty squares.

3. Define pos1 to hold an empty string to be used later in game_loop as the initial condition for the while loop.
4. Define our main loop function, game_loop. This function accepts pos1 which is initially an empty string.
5. Create a while loop that continues while pos1 does not hold the string ‘stop’. If the user inputs ‘stop’ the loop will break and continue to the print statement outside of the loop which will say “The game has been stopped”.
6. Within the while loop, call print_board function to print the board then ask the user to select a piece to move using board coordinates (ex. “a1”) and store it in pos1 and convert pos1 to lowercase.
7. If pos1 does not equal ‘stop’ then if pos1 is found in valid_moves (our list of dark squares) take the first element and convert it to an integer using ord() and subtracting 97 from it to get an integer that we can relate to our board list. This value will be stored in column_pos1
8. Because row 1 in this case should represent the bottom row of the board we must take the row number substring, convert it to an int, subtract 8 from it to make it comparable to our board list then multiply it by -1 to reverse the row order to make it represent the board list accurately. This value will be stored in row_pos1. Call on_board function and pass column_pos1 and row_pos1 to be used to determine if the selected square has a piece.
9. If pos1 is not in valid_moves (it is a light square) call light_square function and pass pos1 as an argument.
10. Define light_square function. light_square will accept an initial position as an argument and store it in pos2. light_square will be called any time the user selects a light square before they have selected a proper piece to move.
11. Create a while loop that loops while pos2 is not in valid_moves. Prompt the user to select a dark square and replace pos2 with the lowercase of the input. If the user inputs ‘stop’ for pos2, our stop function will be called. If pos2 is in valid_moves, convert the column and row as mentioned above and store in column_pos2 and row_pos2. Call on_board function and pass column_pos2 and row_pos2 as arguments.
12. Define on_board function. This function accepts column and row as arguments.
13. First, on_board will check if the square on the board holds a period (periods represent empty squares) using the row and column values. Within this if, have a while loop that loops while the selected square holds a period. Within this loop, tell the user the square is empty and prompt them for a different coordinate. Store this input in pos3 and convert it to lowercase. If the user inputs ‘stop’ then call stop

function. Convert user input to coordinates as mentioned above and store in column_pos3 and row_pos3. If pos3 is not in valid_moves, call light_square function and pass pos3 as an argument. If pos3 is in valid_moves, call on_board to continue through the rest of the function. Place a break after valid_moves if statement to prevent the loop to continue after if statement procedures.

14. Second, on_board will prompt the user to select a dark square to move to if the square they selected does not contain a period. Because of the previous checks on the user's input, this portion of on_board will only activate once the user has properly selected a piece. Store user input in pos4 and convert to lowercase. If the user enters 'stop' for pos4, call stop function. Convert user input to coordinates as mentioned above and store values in column_pos4 and row_pos4. If pos4 is in valid_moves, if pos1(the initial position) is the same as pos4, call game_loop as the board will not need to be changed. Else, take the element in board[row_pos4][column_pos4] (the new position element) and replace it with board[row][column] (the initial position element). Replace the initial position element with a period.
15. If pos4 is not in valid_moves, have a while loop that loops while pos4 is not in valid_moves. Tell the user that their selection is not a dark square and prompt them to enter a new coordinate. Replace value in pos4 with this value and convert it to lowercase. If the user inputs 'stop' call stop function. Convert user input to coordinates as mentioned above and store values in column_pos4 and row_pos4. If pos4 is in valid_moves, then if pos1 is equal to pos4, call game_loop as the board does not need to be changed. Else, take the element in board[row_pos4][column_pos4] (the new position element) and replace it with board[row][column](the initial position element). Replace the initial position element with a period.
16. Define stop function. This function does not accept an argument and prints "The game has been stopped" and uses the exit() function to end the program.
17. Define print_board function. This function takes board list as an argument. For each list (row) in board and for each element (column) in each list, print the elements. Replace the separators with extra spaces using end= to make the board more visually pleasing. Print '\n' at the end of each row.
18. After all function definitions, call the game_loop function.
19. Comment code will be included throughout the program.

Summary:
All we are doing with our code is taking the user input and making sure the inputs are valid by comparing it to our list of valid moves. If the input is valid we update our board list elements with the user's input. The program will continue to loop until the user enters ‘stop’.

Test Cases:

1. Input: (d4, a1, h8) Output: Board printed, enter a piece to move, this square is empty pick another, enter a dark square to move to, a1 piece replaces h8 piece, print board, enter a piece to move (edge)
2. Input: (a1, h1, a1) Output: Board printed, enter a piece to move, enter a dark square to move to, this square is light pick another, board does not change, print board, enter a piece to move (edge)
3. Input: (h1, c5, a1, c5) Output: Board printed, enter a piece to move, this square is light, enter a piece to move, this square is empty enter a piece to move, select a square to move to, replace c5 square with a1 piece, print board (edge)
4. Input: (a1, c5) Output: Board printed, enter piece to move, select square to move to, replace c5 with a1 piece, print board (typical)
5. Input: (c5, h1, c5 h1) Output: Board printed, enter a piece to move, this square is empty pick another, this square is light pick another, this square is empty pick another, select a square to move to (edge)
6. Input: (a1, h1, d4, d4, h1, h8) Output: Board printed, enter a piece to move, select a square to move to, this square is light pick another, a1 piece replaces d4, print board, enter a piece to move, select a square to move to, this square is light pick another, d4 piece replaces h8 piece, print board, enter a piece to move (edge)
7. Input: (h2, h8, h8) Output: Board printed, enter a piece to move, select a square to move to, print board, enter a piece to move, select a square to move to, print board, enter a piece to move (typical)
8. Input: (a3, a1) Output: Board printed, enter a piece to move, select a square to move to, a3 square now blank, print board (typical)
9. Input: (b8. a3, a3, c5) Output: Board printed, enter a piece to move, select a square to move to, b8 piece replaces a3 piece, print board, enter a piece to move, select a square to move to, replace c5 with a3 piece, print board, enter a piece to move (typical)

10. Input: (c5, d4, h8) Output: Board printed, enter a piece to move, this square is empty pick another, this square is empty pick another, this square is light pick another (edge)

Further test cases

Trial	Piece selected	Piece moved	result
1. typical	g7	g1	pass
2. typical	b6	h4	pass
3. typical	e3	c5	pass
4. typical	h6	g7	pass
5. typical	d2	a5	pass
6. edge	‘stop’		game stopped
7. typical	d6	g1	pass
8. edge	f4	g4	invalid move
9. typical	b2	c1	pass
10.typical	b8	d4	pass
11.typical	g7	h6	pass
12.typical	c7	f4	pass
13.typical	g3	e1	pass
14.typical	h6	b2	pass
15.typical	b2	d2	pass
16.edge	c1	stop	game stopped
17.typical	d2	d6	pass
18.typical	d6	h8	pass
19.typical	h8	b8	pass
20.edge	g1	h1	invalid move