ENEE 150: Intermediate Programming Concepts for Engineers

Spring 2019 **Handout #7**

Project #1: Chinese Checkers, Due: Feb. 20th, 11:59p.m.

In this project, you will build a program that allows 2 human players to play the game of chinese checkers. (Chinese checkers allows up to 6 players, but for this project, you will only support 2 players). Your program will graphically maintain the state of the game board and prompt the players for moves on the standard input/output. Your program will also know the rules of chinese checkers, and will check the validity of moves to maintain the integrity of the game board as the game progresses.

1 Chinese Checkers Overview

Chinese checkers is played on a board with 121 spaces arranged in a 6-pointed star. Each player has 10 "marbles" (each player's marbles bear a different color) that are initially placed on the board inside one of the 6 star points. Players take turns making moves. Two kinds of moves are allowed. A player can move one of his/her marbles to an unoccupied neighboring space in any direction. Alternatively, if a player's marble is next to another marble, the player can "jump" over the adjacent marble as long as the space on the other side of the jumped marble is unoccupied. It is possible to jump many times in a row on the same turn as long as after each jump, the player's marble lands next to another marble. The goal of the game is to move all of one's marbles from the star point where they are initially placed to the star point directly opposite on the board. The first player to do so wins the game.

2 The Game Board

At the beginning of the game, your program should print the initial state of the game board, and prompt the first player for his/her move. For example, the sample output at the beginning of a 2-player game looks like:

```
z: pr1
    0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
 0                                      R
 1                                   R     R
 2                                R     R     R
 3                             R     R     R     R
 4  -     -     -     -     -     -     -     -     -     -     -     -     -
 5     -     -     -     -     -     -     -     -     -     -     -     -
 6        -     -     -     -     -     -     -     -     -     -     -
 7           -     -     -     -     -     -     -     -     -     -
 8              -     -     -     -     -     -     -     -     -
 9           -     -     -     -     -     -     -     -     -     -
10        -     -     -     -     -     -     -     -     -     -     -
11     -     -     -     -     -     -     -     -     -     -     -     -
12  -     -     -     -     -     -     -     -     -     -     -     -     -
13                             G     G     G     G
14                                G     G     G
15                                   G     G
16                                      G

RED’s move:
```

As the above output shows, the game board is a 6-pointed star consisting of 121 spaces. Each star point is a 10-space triangle, and each triangle is positioned at one side of a 61-space hexagon. To provide the hexagonal and triangular shapes, the spaces in adjacent rows of the board are staggered. In this project, we will refer to each board space using a zero-based row and column ID, with the center of the star positioned at row 8 column 12 (8,12). In this coordinate system, many coordinates are not part of the game board. In particular, all coordinates that lie outside of the star are invalid–*e.g.*, (2,3) and (3,21). Also, due to the staggering of the rows, the even columns in the odd rows and the odd columns in the even rows are invalid as well–*e.g.*, (1,12) and (2,13).

In a 2-player game, the marbles are colored red and green, and are initially placed in the top and bottom star points of the board. All other spaces on the board are initially empty. When printing the game board, each coordinate in the board is printed as 3 characters. The first and last character is always a space character, ’ ’. The middle character depends on the coordinate. For invalid coordinates, the middle character is also a space character, ’ ’. For coordinates corresponding to empty spaces, the middle character is a ’–’ character. For coordinates corresponding to spaces occupied by red marbles, the middle character is ’R’. And for coordinates corresponding to spaces occupied by green marbles, the middle character is ’G’. Lastly, the row and column IDs for the board are printed along the left-hand side and top of the board, as shown in the above sample output.

The easiest way to represent the game board is by using a 2-dimensional array of size 17 rows by 25 columns. That way, elements of the array map directly onto coordinates in the coordinate system described above. You should store values into the array to signify whether the corresponding coordinate is valid or invalid, and if valid, whether the corresponding coordinate is empty or holds a red or green marble. Finally, the row and column ID are printed to the left and above the board, as shown in the above output, to allow players to easily identify different board positions.

3 Moving Pieces

After printing the initial state of the game board, your program should prompt each player for a move, alternating between the red and green marbles. As the sample output from Section 2 shows, the red player moves first.

At the prompt, your program should expect the corresponding player to enter 4 numbers: the first 2 numbers specify the column and row, in that order (*i.e.*, an x-y coordinate), of a space containing the marble to move, and the second 2 numbers specify the column and row of an empty space to move to. After verifying that the move is valid (see Section 3.4), your program should update the state of the game board, print the updated game board, and then prompt for another move.

3.1 Neighbor Moves

As mentioned in Section 1, a player can make two types of moves. The first type is to move a marble to a neighboring space next to the space originally occupied by the marble. Each space on the game board has 6 neighbors: one to the left, one to the right, and 4 along diagonals. For example, using the coordinate system discussed in Section 2, the neighbors of (6,14) are (6, 12), (6, 16), (5, 13), (5, 15), (7, 13), and (7, 15). Here is some sample output for the first two moves of a 2-player game where the red and green players make moves to neighboring spaces. Notice after each move, the program prompts the opposing player for his/her move:

```
RED's move:  9 3 10 4

    0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
 0                                      R
 1                                   R     R
 2                                R     R     R
 3                             -     R     R     R
 4  -     -     -     -     -     R     -     -     -     -     -     -     -
 5     -     -     -     -     -     -     -     -     -     -     -     -
 6        -     -     -     -     -     -     -     -     -     -     -
 7           -     -     -     -     -     -     -     -     -     -
 8              -     -     -     -     -     -     -     -     -
 9           -     -     -     -     -     -     -     -     -     -
10        -     -     -     -     -     -     -     -     -     -     -
```

```
11     -     -     -     -     -     -     -     -     -     -     -     -
12  -     -     -     -     -     -     -     -     -     -     -     -     -
13                             G     G     G     G
14                                G     G     G
15                                   G     G
16                                      G
```

```
GREEN's move:  11 13 12 12
```

```
    0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
 0                                      R
 1                                   R     R
 2                                R     R     R
 3                             -     R     R     R
 4  -     -     -     -     -     R     -     -     -     -     -     -     -
 5     -     -     -     -     -     -     -     -     -     -     -     -
 6        -     -     -     -     -     -     -     -     -     -     -
 7           -     -     -     -     -     -     -     -     -     -
 8              -     -     -     -     -     -     -     -     -
 9           -     -     -     -     -     -     -     -     -     -
10        -     -     -     -     -     -     -     -     -     -     -
11     -     -     -     -     -     -     -     -     -     -     -     -
12  -     -     -     -     -     -     G     -     -     -     -     -     -
13                             G     -     G     G
14                                G     G     G
15                                   G     G
16                                      G
```

```
RED's move:
```

3.2 Jump Moves

The second type of move is a jump move. A marble can make a jump move if one of its 6 neighboring spaces (see Section 3.1) is occupied by another marble, and the space on the other side of the neighboring marble is empty. Note, the neighboring marble can be of any color (red or green). In this case, the player can jump over the neighboring marble into the empty space. For example, here is some sample output where the red player makes a jump move after the first 2 moves from Section 3.1. Notice, after the red player makes his/her jump move, the program prompts the green player for a move:

```
RED's move:  12 2 14 4

    0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
 0                                      R
 1                                   R     R
 2                                R     -     R
 3                             -     R     R     R
 4  -     -     -     -     -     R     -     R     -     -     -     -     -
 5     -     -     -     -     -     -     -     -     -     -     -     -
 6        -     -     -     -     -     -     -     -     -     -     -
 7           -     -     -     -     -     -     -     -     -     -
 8              -     -     -     -     -     -     -     -     -
 9           -     -     -     -     -     -     -     -     -     -
10        -     -     -     -     -     -     -     -     -     -     -
11     -     -     -     -     -     -     -     -     -     -     -     -
12  -     -     -     -     -     -     G     -     -     -     -     -     -
13                             G     -     G     G
14                                G     G     G
15                                   G     G
16                                      G

GREEN's move:
```

3.3 Multiple Jumps

Normally, after a player enters a move, your program should prompt the opposing player for a move. The exception is multiple jumps by a player on the same turn. If a player makes a jump move that lands in a space from which another jump move can be made, instead of prompting the opposing player for a move, your program should prompt the *same player* for another move. If the re-prompted player makes a move, it must be a jump move, and he/she must move the same marble from the previous jump move. (Note, the jump move cannot jump back to the same space from which the previous jump move started. Instead, the jump move must jump to a new space.) For example, after the red player's jump move from Section 3.2, the green player can make a multiple jump move consisting of 2 jumps in a row. In this case, your program should prompt the green player twice before prompting the red player again:

```
GREEN's move:  13 15 11 13

    0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
 0                                      R
 1                                   R     R
 2                                R     -     R
 3                             -     R     R     R
 4  -     -     -     -     -     R     -     R     -     -     -     -     -
 5     -     -     -     -     -     -     -     -     -     -     -     -
 6        -     -     -     -     -     -     -     -     -     -     -
 7           -     -     -     -     -     -     -     -     -     -
 8              -     -     -     -     -     -     -     -     -
 9           -     -     -     -     -     -     -     -     -     -
10        -     -     -     -     -     -     -     -     -     -     -
11     -     -     -     -     -     -     -     -     -     -     -     -
12  -     -     -     -     -     -     G     -     -     -     -     -     -
13                             G     G     G     G
14                                G     G     G
15                                   G     -
16                                      G

GREEN's move:  11 13 13 11

    0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
 0                                      R
 1                                   R     R
 2                                R     -     R
 3                             -     R     R     R
 4  -     -     -     -     -     R     -     R     -     -     -     -     -
 5     -     -     -     -     -     -     -     -     -     -     -     -
 6        -     -     -     -     -     -     -     -     -     -     -
 7           -     -     -     -     -     -     -     -     -     -
 8              -     -     -     -     -     -     -     -     -
 9           -     -     -     -     -     -     -     -     -     -
10        -     -     -     -     -     -     -     -     -     -     -
11     -     -     -     -     -     -     G     -     -     -     -     -
12  -     -     -     -     -     -     G     -     -     -     -     -     -
13                             G     -     G     G
14                                G     G     G
15                                   G     -
16                                      G

RED's move:
```

There is no limit to the number of jumps that can be made by a player on a single turn. As long as another jump can be made by the same moving marble, your program should continue prompting the same player for a move. A jump move terminates if no other jump moves can be made (as is the case in the above example), or if the user enters "-1 -1 -1 -1" when he/she is re-prompted. In other words, even if another jump can be made, the player is not required to make the jump, and can terminate the turn by entering "-1 -1 -1 -1".

3.4 Verifying Moves

After a player enters a move, but before the game board is updated, your program should verify that the move entered is a valid move. Your program should check several conditions:

1. The move should originate from and terminate to spaces on the board (*i.e.*, the move should stay on coordinates that are part of the game board).
2. The originating space must contain a marble belonging to the player making the move, while the terminating space must be empty.
3. The move must be a legal neighbor move or jump move, as described in Sections 3.1, 3.2, and 3.3.

If a move meets all these conditions, it is a valid move, and your program should update the game board accordingly. Otherwise, your program should print the error message "INVALID MOVE. TRY AGAIN!!", and prompt the same player for another move. Re-prompting continues until a valid move is entered. Note, this error checking should be performed during all types of moves: neighbor moves, jump moves, and on each step of a multiple jump move.

4 Termination

Normally, the game ends when either the red or green player successfully moves all of his/her marbles into the opposing star point on the game board. In this project, you do not need to detect this game-ending condition. Instead, your program should simply process moves, alternating between the 2 players. Your program should end if a player enters "-1 -1 -1 -1" at the prompt anytime other than during a multiple jump move. (In other words, "-1 -1 -1 -1" either terminates a multiple jump move, or it terminates the game).

5 Project Design

As discussed in class, functional decomposition is crucial for building large programs. It maximizes ease of understanding, incremental testing and debugging, as well as reuse of your code. For this project, **your code is required to implement the functions in Section 5.1**. The discussion in Section 5.1 not only describes what each required function does, it also specifies the arguments and return values for each required function. **Your code must implement these interfaces exactly as described.**

5.1 Required Functions

There are 5 required functions. These functions are described below, and are listed in the "chinese-checkers.h" header file provided on the course website (follow the hyperlink for "Project 1 Files"). In the description that follows, references are made to constants. These constants are in all uppercase letters, and are also defined in the chinese-checkers.h file.

1. `void print_board()`

This function prints the game board in the format described in Section 2, indicating the location of all the marbles. This function has no arguments and no return values.

2. `int check_neighbor(int x_from, int y_from, x_to, y_to)`

This function checks whether or not a move, specified by the four parameters x_from, y_from, x_to, and y_to, constitutes a neighbor move. Specifically, the function checks whether the space at coordinate (x_to, y_to) is one of the 6 neighbors of the space at coordinate (x_from, y_from), as described in Section 3.1. If the move is a valid neighbor move, the function returns VALID_MOVE; otherwise, it returns INVALID_MOVE.

3. `int check_jump(int x_from, int y_from, x_to, y_to)`

This function checks whether or not a move, specified by the four parameters x_from, y_from, x_to, and y_to, constitutes a jumping move. Specifically, the function checks whether the space at coordinate (x_to, y_to) is on the other side of one of the 6 neighbors of the space at coordinate (x_from, y_from), and that the neighbor in question is occupied by a marble, as described in Section 3.2. If the move is a valid jumping move, the function returns VALID_MOVE; otherwise, it returns INVALID_MOVE.

4. `int check_move(int color, int x_from, int y_from, x_to, y_to)`

This function checks whether or not an entered move meets several correctness criteria. The function takes 5 arguments. "color" specifies whether the mover is red or green (RE or GR, respectively). The arguments "x_from" and "y_from" specify the space to move from, and the arguments "x_to" and "y_to" specify the space to move to. The function checks the following: the move originates from and terminates to spaces on the board, the originating space contains a marble belonging to the player making the move, the terminating space is empty, and the move is either a valid neighbor move or jumping move. (For the latter, you should call check_neighbor and check_jump). If everything checks correctly, the function returns VALID_MOVE. Otherwise, it returns INVALID_MOVE.

Note, the check_move function does not know whether the current move is part of a multi-jump move. Hence, it should not check for the multi-jump criteria described in Section 3.3. In particular, it should not check that the marble being moved is the same marble during a multi-jump sequence. Also, it should not check that the terminating space is different from the originating space on the previous jump move during a multi-jump sequence. In the event of a multi-jump move, check_move can still be called, but these other correctness critiera specific to multi-jump moves should be verified outside of check_move.

5. `int is_jumper(int x_from, int y_from, int x_to, int y_to)`

Given that x_from and y_from specify the originating space and x_to and y_to specify the terminating space of a valid jumping move, this function determines whether the marble (now at coordinate (x_to, y_to)) can make a valid jumping move over any one of its neighbors. The function should not consider the jumping move back to the coordinate (x_from, y_from) as being valid (*i.e.*, there are only 5 neighbors that can potentially provide a valid jumping move). Also, the function should check that the potential jump move both originates from and terminates to valid spaces on the board. If everything checks correctly, the function returns TRUE. Otherwise, it returns FALSE. (Hint: this function should call check_jump).

This function should be called after a valid jumping move is made to determine whether another jumping move is possible (and hence, a multi-jump move should be initiated or continued).