CITS 2401

Computer Analysis and Visualisation

Assignment 3
Tic Tac Toe

Worth: 5% of the unit

Submission: Answer the questions on the quiz server.
Deadline: 7 May 2020 5pm
Late submissions: late submissions attract 5% penalty per day up to 7 days (i.e., 14 May 2020 5pm). After, the mark will be 0 (zero). Also, any plagiarised work will be marked zero.

1. Outline

In this assignment, you will be implementing a well-known game - Tic Tac Toe. This game will allow a person to play a game of Tic Tac Toc against the computer.

The rules are simple:
1. Tic Tac Toe is played on an 3 by 3 grid always.
2. At the beginning of the game, a player (e.g. you) will select an appropriate shape, either 'X' or 'O'.
3. The player will always begin the game.
4. The computer will automatically get the remaining shape (i.e. if you selected 'X', the computer will play 'O').
5. The player who has his/her own shape 3 in a row, vertical, horizontal or diagonal, will win the game (the game will notify who is the winner and terminates).
6. In case of a draw, the game will notify the player that the game has drawn, and terminates.

Figure below shows the overall architecture of the game, which shows most of the functions you need to complete.

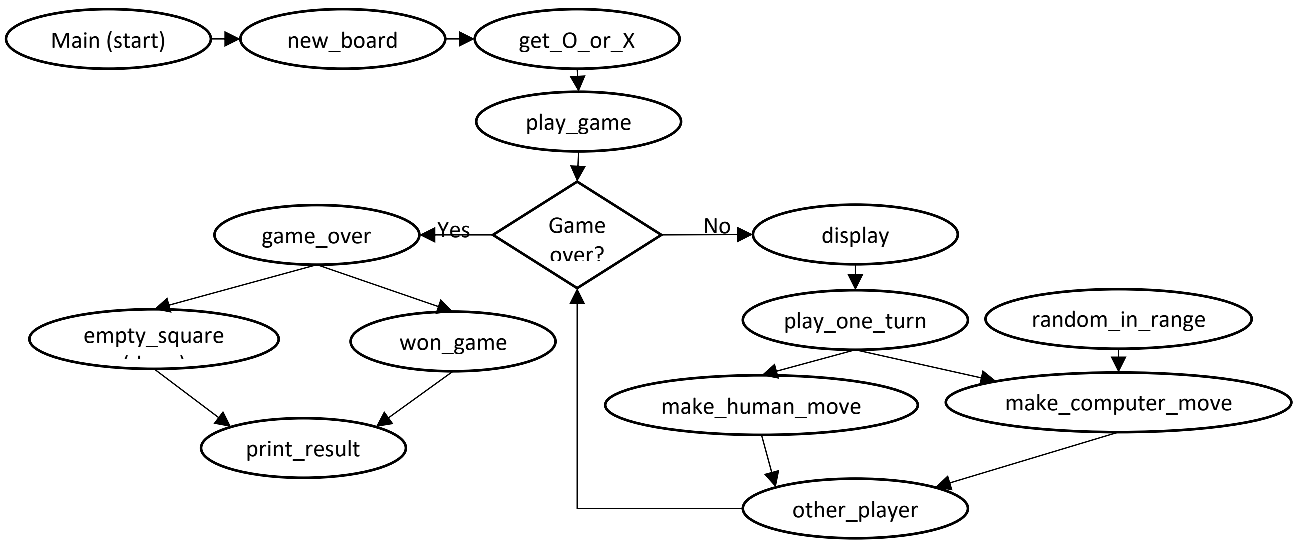

Figure 1. Tic Tac Toe game architecture

CITS 2401
Computer Analysis and Visualisation

2. Tasks

Task 1: Creating the board
Complete the function `new_board()`, which returns a nested list to represent the 2-dimentional tic tac toe board, with each row being the length of 3. When first created, the board should be empty. An empty board cell is defined as a single space character.

Task 2: Accessing the board
Next, we would like to access the square (a particular location on the board). the indices of squares are 0 0, 0 1, 0 2, 1 0, 1 1, 1 2, 2 0, 2 1, and 2 2 (total 9 squares).

Complete the function `square(board, row, column)`, which returns the content of the square at location (row, column).

Here, you can assume the values for row and column will be valid.

Task 3: Retrieving rows and columns
Next, we would like to retrieve the row and column information from the board, represented as a list. Below depicts an example board state.

```
+---+---+---+
| X |   | O |
+---+---+---+
|   | O |   |
+---+---+---+
| X |   |   |
+---+---+---+
```

For example, from the board above, the function `row(board, row_num)` will return ['X', ' ', 'O'] if `board` was [['X', ' ', 'O'], [' ', 'O', ' '], ['X', ' ', ' ']] (i.e. the board above) and `row_num` was 0.

Similarly, the function `column(board, column_num)` will return ['X', ' ', 'X'] if `board` was [['X', ' ', 'O'], [' ', 'O', ' '], ['X', ' ', ' ']] (i.e. the board above) and `column_num` was 0.

Complete those two functions.

Task 4: Retrieving diagonals
Next, we would like to check the content of the diagonals. We have two diagonals: (1) top-left to bottom-right, and (2) top-right to bottom-left. These are identified using global constants TOP_LEFT_BOTTOM_RIGHT (value 0) and TOP_RIGHT_BOTTOM_LEFT (value 1), respectively (in your skeleton code, they are at lines 8 and 9).

Complete the function `diagonal(board, diagonal_selector)`, which returns the content of the diagonal as a list.

Task 5: Find empty cells
We would like to identify all the empty squares on board. The empty square locations on the board will be returned as a (row, column) tuple.

Complete the function `empty_squares(board)`, which returns the list of tuples, where each tuple is the (row, column) of the empty square.

Task 6: Check line winner
We now have set up the board and functions to retrieve the information from the board (squares, rows, columns and diagonals). Next, we set up a function `line_winner(line)`, which checks if the input `line` contains 3 elements that are the same other than a space character (i.e. a player has won). Parameter input `line` is a list, which can be row, column or diagonal (e.g. ['X', ' ', 'O']). This function will return the shape of the player (e.g. if line = ['X', 'X', 'X'] then it will return 'X'), otherwise returns `None`.

Task 7: Winner winner chicken dinner!
Using the function `line_winner(line)`, now we can implement the winner function!

The function `winner(board)`, will inspect every row, column and diagonals to check whether any player has won the game. If there is a winner (i.e. the function `line_winner(line)` returned something other than None) then function `winner(board)` also returns the winner. If no winner is yet to be found, it returns `None`.

Complete function `winner(board)`.

Task 8: Not everyone can be a winner
Sometimes, there can be no winners in this game. Hence, we need to constantly check that the board has free squares to play while there is no winner.

Write a *boolean* function `game_over(board)`, which returns true if and only if there is a winner or if the game has drawn.

Task 9: Game result
Now we can implement the top level functions that controls the game flow.

Write a function `game_result(board)`, which returns a string:

1. 'Draw' if there is no winner, or
2. 'Won by [player]' if a player [player] has won the game. (e.g. if 'X' has won, it prints 'Won by X').

This function should be using the function `winner(board)`. You can assume that this function is called only when a player wins or the board is full.

Task 10: Human player
Write a function `make_human_move(current_player, board)`, which does the following:

1. Print out the player's shape (e.g. if the player shape is 'X', the function prints "X's move").
2. Asks the human player for his/her move. The message asked is "Enter row and column [0 - 2]: ".
3. A move consists of two integer values separated by a space (e.g. '0 0'). You can assume the input will always be two integer values separated by a space.
4. The function checks if the move is valid or not.
    a. A valid move is when the integer value is between 0 and 2 for both row and column values.
    b. It also makes sure that a valid location on the board is empty (i.e. currently an empty space character ' ').
5. If a move is invalid, the function prints "Illegal move. Try again." and repeats from step 2.
6. Once a valid move is entered, the board state is updated.
7.

Hint: use the `empty_squares(board)` function.

CITS 2401

Computer Analysis and Visualisation

THE UNIVERSITY OF WESTERN AUSTRALIA

Task 11: Who's turn?
Write a function `play_one_turn(board, current_player, human_player)`, which calls function `make_human_move` if the current player is the human player, otherwise calls the `make_computer_move` function.

This function does not return anything.
Note: to generate the same computer output, set the random seed at 15.

Task 12: Taking turns
Write a function `other_player(player)`, which returns the shape of the other player. For example, if player is 'X', then returns 'O', otherwise return 'X'.

You can assume that the input player will always have value 'X' or 'O'.

Task 13: Human player chooses shape
Write a function `get_O_or_X()`, which asks the human player to select the shape to play, either 'X' or 'O'. while the choice is not 'X' or 'O', the function will keep asking the player to select the valid shape to play.

The input question is "`Would you like to play O or X? `".

Task 14: Putting everything together
Write a function `play_game(human_player, board)`, which controls the overall game flow. Steps are as follows.

1. set the current player as human (human player will always start).
2. continue playing until the game is over.
    a. you should display the board (using the built-in display function),
    b. allow the current player to make a move, and
    c. alternate the current player between human and computer.
3. Once the game is finished, the function returns the game result (hint: use the `game_result` function).

Task 15: Play time!
Finally, we will initiate the game by calling the main function.

Write a function `main()`. Steps are as follows (don't worry about what does blocks are doing for now).

Before the `try` block:

1. create the board
2. assign a shape to the human player

Inside the `try` block:

1. play the game
2. display the final state of the board

The function has been partially completed for you, do not touch the exception handler (out of scope in this unit).

CITS 2401

Computer Analysis and Visualisation

THE UNIVERSITY OF WESTERN AUSTRALIA

3. Submission

You should answer the questions related to the tasks above on the quiz server by the due date - 7 May 2020 5pm (drop dead due date 14 May 2020 with 5% penalty per day).