Assignment 1

Submit Deadline: 10pm Fri 4 Sep 2020

1. Problem: Nine-Board Tic-Tac-Toe Game

Tic-Tac-Toe or Noughts and Crosses is a simple but popular game that have been played by humans for thousands of years (see Figure 1). It is also one of the earliest computer games with implementation of computer player that could play perfect games against a human opponent[1]. Due to the simplicity of the game, implementation of a perfect computer player for Tic-Tac-Toe is not hard. There are quite a few variations of Tic-Tac-Toe, either increase the size of board, dimension of the board or the number of boards, which are more challenging. Nine-Board Tic-Tac-Toe game[2] is one of the interesting ones.

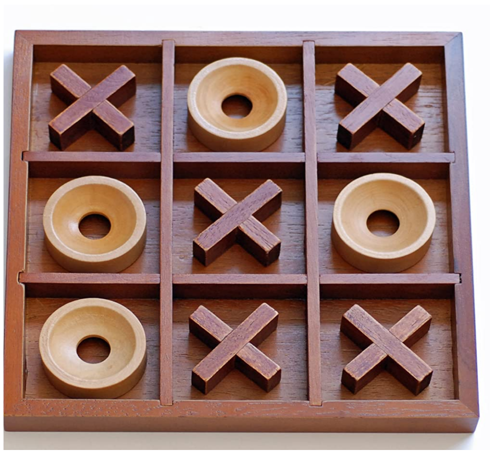
Figure 1: *Tic-tac-toe game*

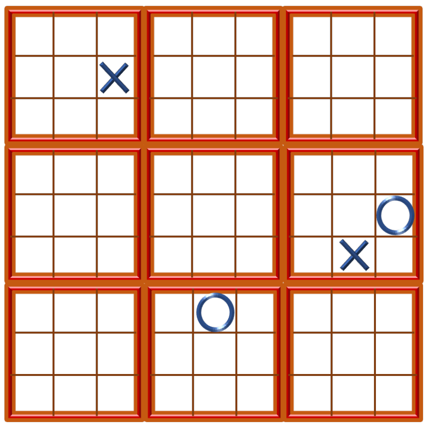
Figure 2: Nine-board Tic-tac-toe *game*

The Nine-Board Tic-Tac-Toe game contains nine identical tic-tac-toe game boards[3], arranged in a 3*3 grid as shown in Figure 2. The first player, X player, randomly picks up a board to play, for instance, the board on the top left corner or board (1,1). If he plays a move in that board at row 2 and column 3, the second player, player O, must play on the board (2,3), which is the board at right middle. If player O chooses to play also at row 2 and column 3

[1] https://en.wikipedia.org/wiki/Tic-tac-toe
[2] http://complaw.stanford.edu/complaw/exercises/nineboard.html
[3] Please note that the Nine-Board Tic-Tac-Toe game is different from the 3D Tic-Tac-Toe game (https://en.wikipedia.org/wiki/3D_tic-tac-toe) and the Ultimate Tic-Tac-Toe game (https://en.wikipedia.org/wiki/Ultimate_tic-tac-toe), which also contains nine Tic-Tac-Toe boards but the rules are totally different from the Nine-Board Tic-Tac-Toe.

on that board, player X has to play also on board (2,3). Game continues until one of the players has three in a row on any of the nine boards, who then wins the game and terminates the game. If a player is forced to go to a full board, the player can choose any other board to play. If all boards are full without a winner, the game terminates with a draw. You may try to play the game via the following website: http://complaw.stanford.edu/complaw/chapters/nineboard.html.

2. Task specification and requirements

In this assignment, you are required to write a computer program to facilitate nine-board Tic-Tac-Toe game playing between human players, computer players, or computer player vs human player. Your program must be written in C++ using object-oriented paradigm. You may use any code provided with the lectures and practicals of this unit or any code you have written for the practical tasks. If you use any existing code acquired from elsewhere, you must explicitly cite the sources in your code, report to your tutor while you demonstrate your code and be able to fully understand the code.

2.1 Pass Level Tasks

To achieve a pass grade (up to 64%), you need to perform the following tasks:

Task 1 (5%): Create a `struct`, named BoardCoordinate, with two public data members of `int` to represent the coordinate of a board in a Nine-Board Tic Tac Toe game.

Task 2 (10%): Combine your solution to Practical Tasks 3.3, 3.4, 4.3, 4.4 and 4.5 into a complete object-oriented solution to single board Tic-Tac-Toe game. The class name of a single Tic-Tac-Toe board should be TicTacToe as required in Practical Task 4.3.

Task 3 (45%): Create a class, named NineBoard, which contains at least two data members (5%):

- A 3*3 two-dimensional array of TicTacToe (or TicTacToe pointer) to store the objects of nine Tic-Tac-Toe game boards.
- An instance (object/variable) of BoardCoordinate to store the coordinates of the current board to play.

The class should contain sufficient methods to implement the following functionalities:

- Randomly generate the coordinates of the initial board to play in the class constructor (5%).
- Accept human input of moves or randomly generate moves (5%)
- Display all nine boards with information of the current board and update the display whenever a new move is added (10%).
- Automatically switch current board based on the last move (5%). If a board has full, randomly choose another available board as the current board (5%).
- Check if a player wins whenever a new move is added. Terminate the

game if a player wins (5%).

- Check if all nine boards are full. If yes, terminate the game as a draw (5%).

Task 4 (4%): Write a class driver which contains the main function to run the whole program. It is recommended that the driver is as simple as possible.

Requirements and hints:

1) You must use a two-dimensional array of TicTacToe (or pointers) to store the objects of nine Tic-Tac-Toe boards. You could convert the data in the nine boards into a single 9*9 array of char for an easy implementation of displaying boards (see Credit level), but the data member of the NineBoard class should be objects of TicTacToe or pointers of TicTacToe objects.
2) Your program should display board status after each move. The layout of display must be **text-based**. For pass level code, you may simply call the display function from TicTacToe with additional information about the current board. See a sample layout in the Appendix.

2.2 Credit Level Tasks

To achieve a credit grade (up to 74%), change your code for pass level to accomplish the following additional task:

Task 5 (10%): Improve the displayBoards method of the NineBoard class so that the layout of the boards looks exactly the same as the following:

```
   - - - - -    - - - - -    - - - - -
|   |   |   ||   |   |   ||   |   |   |
| -   -   - || -   -   - || -   -   - |
|   |   |   || O |   |   ||   |   |   |
| -   -   - || -   -   - || -   -   - |
|   |   |   ||   |   |   ||   |   |   |
  - - - - -    - - - - -    - - - - -
  * * * * *    - - - - -    - - - - -
*   |   |   *|   |   |   ||   | X |   |
* -   -   - *| -   -   - || -   -   - |
*   |   |   *|   |   |   ||   |   |   |
* -   -   - *| -   -   - || -   -   - |
*   |   |   *|   |   |   ||   |   |   |
  * * * * *    - - - - -    - - - - -
  - - - - -    - - - - -    - - - - -
|   |   |   ||   |   |   ||   |   |   |
| -   -   - || -   -   - || -   -   - |
|   |   |   ||   |   |   ||   |   |   |
| -   -   - || -   -   - || -   -   - |
|   |   |   ||   |   |   ||   |   |   |
  - - - - -    - - - - -    - - - - -
```

in which the board with stars indicates the current board (the board to play).

Requirements:

You may have some small variations on the layout, but the display must be text-based. Not graphical GUI is allowed.

2.3 Distinction Level Tasks

To achieve a distinction grade (up to 84%), reconstruct and extend your code for credit level to accomplish the following additional task:

Task 6 (10%): Create new classes, NBGame and Player (or more), for better abstraction and reconstruct the TicTacToe and NineBoard classes for better encapsulation. You may have multiple player classes for different type of players.

Requirements and hints:

1) Input of moves must be taken or generated in the Player class or its derived classes rather than in the TicTacToe or NineBoard class.
2) It is recommended to implement two separate player classes – one for human player and the other for random player. You will receive 5% bonus marks if these player classes are extended from a common abstract class Player using inheritance.
3) Class NBGame should contain one NineBoard object (or pointer) and two Player objects (or an array of two Player objects or pointers) as data members (may contain more data members and member functions).

2.4 High Distinction Level Tasks

To achieve a high distinction grade, reconstruct and extend your code for distinction level to accomplish the following additional task:

Task 7 (15%): Implement a smart computer player so that it can play against human player or random player. Your smart player should be able to:

- Identify if there is a winning move. If yes, take that move (5%).
- Avoid any move that would lead to immediate loss unless there is no other option (5%).
- Other strategies make the player smarter (5%).
- You may try to implement a smart player using AI algorithms, say Monte-Carlo Tree Search, Minimax or heuristic search. You will receive up to 10% bonus marks (up to 5% for each method if it works).

Requirements and hints:

1) You have freedom to organize your classes but highly recommend using inherence and polymorphism for different player classes.
2) Test your smart player against a random player. When your smart player

plays with a random player, it should win almost all the times. If it plays against a human player, it should behave rationally (catch up any winning chance and avoid immediate loss).

3) I will give a tutorial on the AI approaches during a PASS session if there is a demand.

2.5 Tasks for advanced students or students like more challenges

Task 5: Change your code to Java and embed it into the GGP game platform.

Requirements and hints:

(1). GGP platform is a game design platform implemented to facilitate General Game Playing Competitions (http://games.stanford.edu). The system can be downloaded from http://ggp.org.

(2). GGP has a built-in rule description and visualization for Nine-Board Tic-Tac-Toe game. You can create a player to play with other existing game players, especially the Monte Carlo player. Your players can also play the game against each other over the Internet via the platform.

(3). I can give a separate tutorial on GGP platform to the students who are interested in taking the challenge.

3. Deliverables

3.1 Source code

Your program must be written in C++. You can use any IDE to demonstrate your program provided it is available during your demonstration. **All comments must be deleted from your source code when you demonstrate.** The code should be purely written by you. **No part of the code can be written by any other persons or copied from any other source. In the case part of your code is copied from an existing resource, you must mark it out and show it to your tutor.**

3.2 Declaration

There is no requirement for documentation. However, you are required to place the following declaration on the top of your .cpp file:

```
/*********** Declaration*******
I hereby certify that no part of this assignment has been copied from
any other student's work or from any other source.  No part of the
code has been written/produced for me by another person or copied
from any other source.

I hold a copy of this assignment that I can produce if the original
is lost or damaged.
**************************/
```

4. Submission

The source code should be submitted via vUWS before the deadline for documentation purpose. Executable file is not required. All these files (including .h/.hpp and .cpp) should be zipped into one file **with your student id as the zipped file name**. Submission that does not follow the format is not acceptable. Multiple submissions are allowed.

5. Demonstration

You are required to demonstrate your program during **your scheduled** practical session in Week 8 between 18-21 Sep 2018. Your tutor will check your code and your understanding of the code. You may show your code to Dongmo before the deadline and ask him to mark your work. **You will receive no marks if you fail the demonstration, especially if you miss the demo time and fail to demonstrate your full understanding of your code.** Note that it is your responsibility to get the appropriate compilers or IDEs to run your program. **The feedback to your work will be delivered orally during the demonstration.** No further feedback or comments are given afterward.

The program you demonstrate should be the same as the one you submit except that the comments in your program should be taken off before the demonstration. If you fail this assignment at your first demonstration, you are allowed to improve your work in the following week (**maximal grade is Pass in this case**).

6. Additional information

You can download the executable files of my solution from vUWS (no guarantee runnable in your machine). I will also demonstrate my solution for each level during my lectures. Additional training is given as tasks in the practical sessions (Practical Tasks 3.3, 3.4, 4.3, 4.4 and 4.5). Make sure you complete these tasks before you start the assignment. Treat all the series of tasks, including the tasks for the assignment as a pathway towards a better OO programming paradigm rather than things have to do.

A weekly peer-assisted study sessions (PASS) will be run to offer you more training in programming. Note that we encourage students to learn from each other, but work has to be done individually.

Appendix: Example of layout for pass level code

```
Current Board (2,2)
Player X's move: (2, 1)
Board (1,1)
   |   |
 -   -   -
   |   |
 -   -   -
   |   |
Board (1,2)
   |   |
 -   -   -
   |   |
 -   -   -
   |   |
Board (1,3)
   |   |
 -   -   -
   |   |
 -   -   -
   |   |
Board (2,1)
   |   |
 -   -   -
   |   |
 -   -   -
   |   |
Board (2,2)
   |   |
 -   -   -
 X |   |
 -   -   -
   |   |
Board (2,3)
   |   |
 -   -   -
   |   |
 -   -   -
   |   |
Board (3,1)
   |   |
 -   -   -
   |   |
 -   -   -
   |   |
Board (3,2)
   |   |
 -   -   -
   |   |
 -   -   -
   |   |
Board (3,3)
   |   |
 -   -   -
   |   |
 -   -   -
   |   |

Current Board (2,1)
Player O's move: 1
1

Board (1,1)
   |   |
 -   -   -
   |   |
 -   -   -
   |   |
Board (1,2)
   |   |
 -   -   -
   |   |
 -   -   -
   |   |
Board (1,3)
   |   |
 -   -   -
   |   |
 -   -   -
   |   |
Board (2,1)
 O |   |
 -   -   -
   |   |
 -   -   -
   |   |
Board (2,2)
   |   |
 -   -   -
 X |   |
 -   -   -
   |   |
Board (2,3)
   |   |
 -   -   -
   |   |
 -   -   -
   |   |
Board (3,1)
   |   |
 -   -   -
   |   |
 -   -   -
   |   |
Board (3,2)
   |   |
 -   -   -
   |   |
 -   -   -
   |   |
Board (3,3)
   |   |
 -   -   -
   |   |
 -   -   -
   |   |

Current Board (1,1)
Player X's move: (3, 1)
Board (1,1)
   |   |
 -   -   -
   |   |
 -   -   -
 X |   |
Board (1,2)
   |   |
 -   -   -
   |   |
 -   -   -
   |   |
Board (1,3)
   |   |
 -   -   -
   |   |
 -   -   -
   |   |
Board (2,1)
 O |   |
 -   -   -
   |   |
 -   -   -
   |   |
Board (2,2)
   |   |
 -   -   -
 X |   |
 -   -   -
   |   |
Board (2,3)
   |   |
 -   -   -
   |   |
 -   -   -
   |   |
Board (3,1)
   |   |
 -   -   -
   |   |
 -   -   -
   |   |
Board (3,2)
   |   |
 -   -   -
   |   |
 -   -   -
   |   |
Board (3,3)
   |   |
 -   -   -
   |   |
 -   -   -
   |   |
```