Assignment 1

Submit Deadline: 5pm Wed 19 Sep 2018

1. Problem: m,k-Games

An *m,k*-game is an abstract board game in which two players, *black* and *white*, take turns in placing a stone of their colour on an *m*×*m* board, the winner being the player who first gets *k* consecutive stones of their own colour in a row, horizontally, vertically, or diagonally. Tic-tac-toe and Gomoku are special cases of m,k-games. See Wikipedia http://en.wikipedia.org/wiki/M,n,k-game for more general definition.

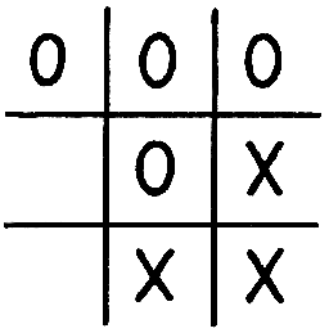

Figure 1: *Tic-tac-toe game*

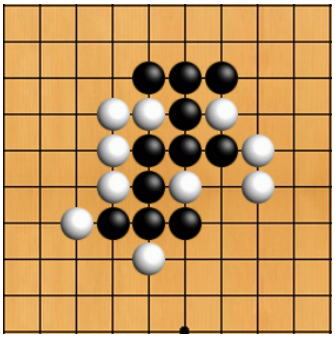

Figure 2: *Gomoku game*

Some *m,k*-games have obvious winning strategies, especially when k is small. For instance, if *m=5* and *k=3*, there is an obvious winning strategy for the player who has the first turn. However, when *m* and *k* are bigger, the games can be much more challenging. Researchers from mathematics, game theory and computer science have been trying to find solutions for complicated *m,k*-games with great success. The most famous result was that there is a winning strategy

for the standard Gomoku game (i.e.,15,5-game) designed by Victor Allis[1].

In this assignment, you are invited to write a computer program that simulates *m,k*-game playing between computer and/or human players. You are not required to develop winning strategies for complicated games but your program should be able to do the following:

1) Simulate game playing for any *m,k*-game where m>2 and k>1.
2) Display any state of a game. The output must be text-based (the layout will be specified in the next section).
3) Accept input from the human player.
4) Create playing strategies (generating moves) for the computer player.

You are required to write your program in C++ using object-oriented paradigm.

2. Task specification and requirements

2.1 Pass Level Task

To achieve a pass grade (up to 64%), your system should be able to perform the following task:

Task 1: Create a class, named Board, which contains at least one data member to store game states (stones on the board) and related member functions to perform the following:

- Accept input (coordinates) as human move
- Generate random coordinates as computer move
- Check if each move is valid
- Check if a board is full
- Display board after each move

Write a driver to test your class.

Requirements:

(1). You must use a two-dimensional array to store board states. The size of the board can be a constant but not be restricted to any prefixed number[2].

---

[1] H. Jaap van den Herik, Jos W.H.M. Uiterwijk, Jack van Rijswijck, *Games solved: now and in the future*, Artificial Intelligence, Volume 134, Issues 1–2, 2002.

[2] You may set:

```
const int M = 5;
const int K = 3;
```

But your program should work properly if the values change to other values, say

```
const int M = 15;
const int K = 5;
```

Hint: *The data type of the array can be integer with 1 for black, -1 for white and 0 for empty*.

(2). You may assume that the human player always takes the first move.

(3). Your program should reject invalid inputs (coordinates out of range or positions occupied).

(4). Your program should display board status after each move. The layout of display must be text-based, similar to the following, where `X` represents black stone and `O` represents white stone. The numbers represent coordinates of the board grid:

```
   1   2   3   4   5   6   7   8
 1 O--- --- --- --- --- --- ---
   |   |   |   |   |   |   |   |
 2  --- --- --- --- ---O--- ---
   |   |   |   |   |   |   |   |
 3  --- --- ---X--- --- --- ---
   |   |   |   |   |   |   |   |
 4  --- ---X---O--- --- --- ---
   |   |   |   |   |   |   |   |
 5  ---X---X--- ---X--- --- ---
   |   |   |   |   |   |   |   |
 6  --- --- --- --- --- --- ---
   |   |   |   |   |   |   |   |
 7  --- --- --- --- --- --- ---
   |   |   |   |   |   |   |   |
 8  --- --- --- --- ---O--- ---
```

Hint: *For beginners, it is not easy to control the position of the grid and numbers, especially the numbers with more than one digit. You may start with simpler layout without coordinates, say*

```
O-------
-----O--
---X----
--XO----
-XX-X---
--------
--------
-----O--
```

Then try to add coordinates. You may assume the size of board is less than 100. You will lose marks if you assume the size is less than 10.

2.2 Credit Level Task

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

Task 2: Add a member function to the Board class to check if a game has reached a winning state, i.e., any player has *k* consecutive stones of his colour in a row, horizontally, vertically, or diagonally.

Requirements:

(1). k can be a constant variable but not be restricted to any numeral. You can assume k>1.

(2). Your driver should contain a piece of code that automatically generates a number of moves to help testing each type of winning conditions.

Hint: *You may start with simply winning conditions, say horizontal or vertical lines, or assume k=3. You will receive marks for partial implementation or restricted version. See more details for the marking sheet.*

2.3 Distinction Level Task

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

Task 3: Create two new classes, MKGame and Player, for better abstraction and reconstruct the Board class for better encapsulation.

Requirements:

(1). Input of moves is taken or generated in the Player class rather than in the Board class.

(2). Class MKGame should contain one Board object and two Player objects (or an array of two Player objects) as data members (may have more data members and member functions).

(3). Use dynamic array to implement the two-dimensional array in the Board class so that m and k can be variables.

Hint: *Check out the task for HD to minimize any changes for further extension*.

2.4 High Distinction Level Task

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

Task 4: Implement a computer player to replace the random player so that it can play against the human player.

Requirements:

(1). You have freedom to organize your classes but highly recommend you to separate implementation of the player class into three classes: a bass class and two derived classes (one for the human player and the other for the computer player).

(2). Create strategies for the computer player. The following are examples of possible strategies:

- Give priority to the central positions.
- If filling a position leads to win, fill it.
- If the other player filling a position leads him to win, block it.

(3). Test your computer player against a random player and human player. When your computer player players with a random player, it should win almost all the times. If it plays against a human player, it should make reasonable (avoid immediate lose while expand existing lines).

Hint: *Try a number of games, such as 3,3-game, 5,3-game, 6,5-game or even 15,5-game to test your strategies*.

2.5 Task 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 Tron games on 10*10 board. 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. You are allowed and highly recommended to demonstrate your program on your laptop. **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. Your programs (such as .h, .hpp, .cpp) can be put in separate files. All these files 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.

No email submission is acceptable. Hard copy of source code is not required.

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 will be given as tasks in the practical sessions. Make sure you can do these tasks with the help of your tutor. A weekly peer-assisted study sessions (PASS) will be run to offer you more training in programming. Note that we encourage students to learn each other but work has to be done individually.