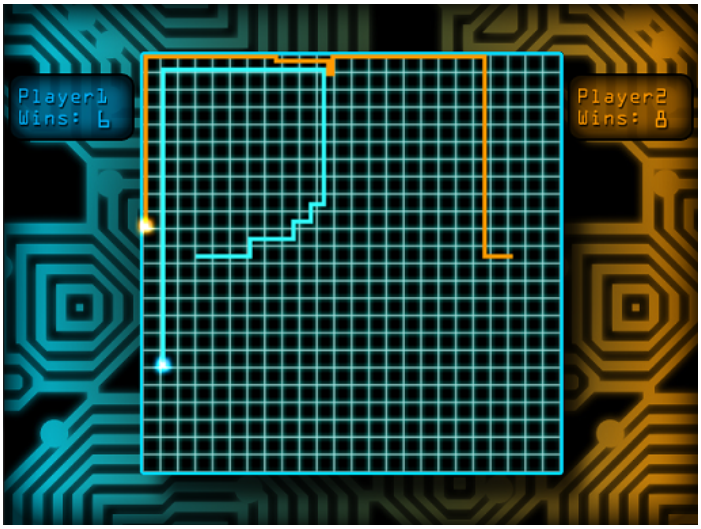
Assignment 1

Submit Deadline: 5:00pm Mon 9 Sep 2019

1. Problem: Tron Game

Tron was a Disney movie released in 1982 talking about a programmer that gets sucked into his computer and its electronic world. The film was centered on a game where the players had to cut each other off using motorbikes that left a line behind them. **Tron game** is a popular computer game, designed based on the motorbike game in the movie. Two players, say `Red` and `Blue`, move simultaneously around a grid board surrounded by walls. Visited cells cannot be visited again by either player. At each step both sides can move one square *up*, *down*, *left*, or *right*. If at least one player crashes into the wall, or previously visited cells, or both sides collide, the game ends. If one player crashes, the survivor wins. If both crash or collide, the game ends in a draw.



(The picture was acquired from the Internet)

In this assignment, you are invited to write a computer program that allows human players or computer players to play a Tron game. As a full solution, your program should be able to do the following:

1) Facilitate Tron games on any *m***m* board (m > 3) playing between human players, computer players, or human and computer players.

2) Display game status for each legal move by game players. The output must be text-based (the layout will be specified in the next section).
3) Accept moves from human players and generate moves automatically for computer players.
4) Follow the structure requirements as specified in the following sections.

You are required to write your program in C++ using object-oriented paradigm. You may translate your C++ code into Java as an option but Java code alone is not acceptable.

2. Task specification and requirements

2.1 Pass Level Task

To achieve a pass grade (up to 64%, more details from the marking sheet available on vUWS), accomplish the following task:

Task 1 (64%): *Write a C++ program that facilitates two human players to play Tron games on a fixed-size board.*

Requirements and hints:

(1). Create a class, named `Board`, containing at least one data member (two-dimensional array) to store game states and at least two member functions for adding players' moves and printing the game board. Write a driver to test your class.

(2). The data type of the two-dimensional array can be either `int` or `char`. As a pass-level program, the size of the board can be hardcoded to 10 or a constant with a pre-set value, anything between 4 and 20.

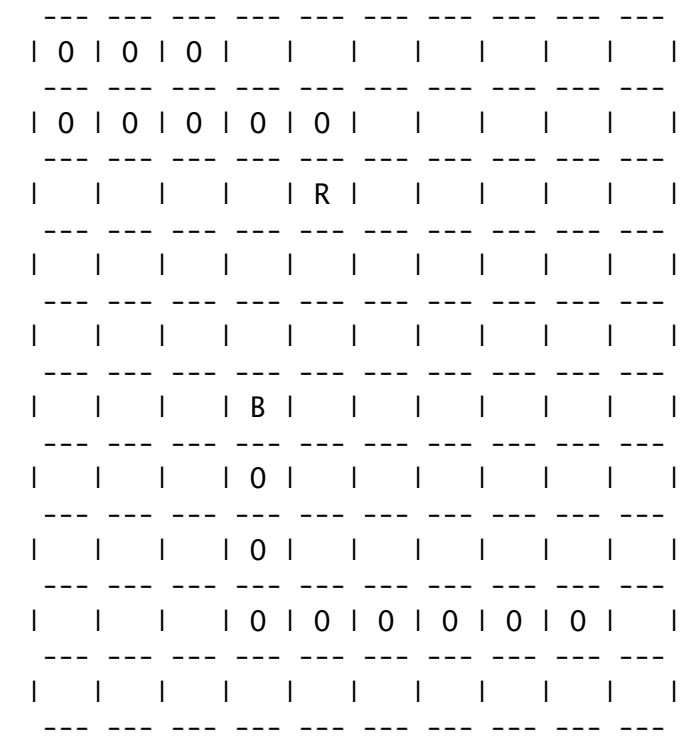
(3). Set the initial state of the board as follows: assuming the size of the board is *m***m*, the initial position of red player is at (*2*, *2*) and the blue player is at (*m-1*, *m-1*) as shown below (where m=5).

```
 --- --- --- --- ---
|   |   |   |   |   |
 --- --- --- --- ---
|   | R |   |   |   |
 --- --- --- --- ---
|   |   |   |   |   |
 --- --- --- --- ---
|   |   |   | B |   |
 --- --- --- --- ---
|   |   |   |   |   |
 --- --- --- --- ---
```

(4). Your program should be able to accept human players' moves. The format of input can be a pair of characters from 'U', 'D', 'L' and 'R'. For instance, (U, L) means that the red player plays "up" and the blue player players "left".

(5). Your program should be able to check if a move terminates the game (either a player crashes or two players collide). If a move does not terminate the game, the board state should be updated in accordance with the move.

(6). Your program should redisplay board state after each legal move. The layout of display must be **text-based**, similar to the following, where 'R' stands for red player, 'B' for blue player and 'O' for visited cells.



(7). If feel hard to fit players' moves into the grid, you may start with a simpler layout without grids as shown below:

```
000-------
00000-----
----R-----
----------
----------
---B------
---O------
---O------
---000000-
----------
```

(8). Check out the tasks for higher levels to minimize possible changes for further extension.

2.2 Credit Level Tasks

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

Task 2: *Use a dynamic array to store board states so that the size of the board can be set at real time.*

Requirements and hints:

(1). Take an input from the user to set the size of the board. For easy display, you can place a restriction on the board size to 4 - 20. The size should be passed to the board class via its constructor.

(2). Since the dynamic array needed is two-dimensional, you might have to declare a pointer of pointers of either `int` or `char` to point to the array.

2.3 Distinction Level Tasks

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

Task 3: *Create a random computer player that can play a Tron game with a human player.*

Requirements and hints:

(1). The random computer player can be implemented as either a method (function) of the Board class or a separate class.

(2). The random player may generate moves randomly but should avoid suicide if other moves are available. A simple way of implementation is to find all feasible non-suicide moves and randomly choose one of them.

2.4 High Distinction Level Tasks

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

Task 4 (5%): Reconstruct your program so that it contains at least three classes, `Board`, `HumanPlayer` and `ComputerPlayer`. Moves should be generated from the player

classes and sent to the board class.

Task 5 (10%): Implement a smart computer player that equips with game playing strategies so that the player can outperform a random computer player[1].

Bonus tasks (up to 100% in total):

Task 6 (5%): Implement all player classes with inheritance: create an abstract class, `Player`, and let the `HumanPlayer` and `ComputerPlayer` as derived classes of `Player` class.

Task 7 (5%): Convert your C++ program into Java.

Requirements and hints:

(1). You have freedom to construct your player classes. You may try to create a base player class and extend it to two or more other player classes using inheritance (one for the human player and the others for the computer players). If you have more than one computer players, you may give the `ComputerPlayer` class different names, say `RandomPlayer` and `SmartPlayer`. You will receive 5% bonus marks if you correctly use inheritance unless your marks have reached 100%.

(2). Your smart computer player must outperform the random player. The following are examples of possible strategies your smart player could use:

- Give priority to the direction that leads you to large areas.
- Check a few steps ahead to see if a direction leads to dead end.
- Follow the other player to get less chance to lose.
- Create a heuristic function and search for a move with the highest heuristic value
- Implement Monte-Carlo Tree Search

The last two approaches are AI based. If any of you are interested in these approaches, I can give an introduction during the PASS meetings.

(3). You may choose to do Tasks 4, 6 and 7 instead of Task 5; or choose Tasks 4 and 5. Doing Task 5 without Task 4 is also possible. You may also complete all the tasks to secure 100%.

2.5 Tasks for Advanced students or students like more challenges

Task 6: Change your code to Java and embed it into the GGP game platform.

[1] Note that the random player you play against must be capable of avoiding suicide.

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.

The above tasks are arranged mainly based on their difficulty of implementation. You may view them as a guideline towards a complete solution. You can choose to complete these tasks in different orders to maximize your marks. Check the marking sheet for the details of marking criteria. Executable files of my solutions for each level will be available on vUWS.

3. Deliverables

3.1 Source code

Your program must be written in C++ with an optional translation in Java. 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.**

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 (.h, .cpp, .java) 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 9-13 Sep 2019. Your tutor will check your code and your understanding of the code. **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 solutions 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 discussion session will be created in vUWS to encourage you learn from each other. **Note that we encourage students to learn each other but work has to be done individually.**