ESTR1002

Problem Solving by Programming

2021 – 2022 Term 1

Course Project – Game “2048”

1. Introduction

2048 is a famous, single-player puzzle game. It is a web game, and you can enter the following URL and try the game: http://2048game.com/

Game Play

In this game, it starts with an empty 4 x 4 game board. The game goes with rounds.

- In each round, the game produces one number block of either '2' or '4' (chosen at random) at a random and empty location on the game board.
- Next, the player chooses either UP, DOWN, LEFT, or RIGHT to move all the number blocks towards the respective direction accordingly.
- After that, the game system will look for pairs of adjacent blocks of the same number, say the number is X, and combine them into a single block of number 2X. This is the only way for the player to eliminate blocks in the game board.
- If the game board is full of number blocks that cannot be eliminated, then the game finishes and the player loses.
- Otherwise, we start another round.
- To win the game, the player needs to produce a block with the number 2048.

The primary movement of the gameplay is to push/pull all the blocks towards a specific direction. After that, the game combines adjacent blocks of the same number.

Action #1: Push the blocks

1. The player can choose to push the blocks to any one of the four directions: UP, DOWN, LEFT, and RIGHT.

2. Then, all the blocks will move toward the chosen direction, so that:

    - There is no gap between any two adjacent blocks in the push direction.
    - After the game has applied the push, we still maintain the same order of the blocks along the push direction.

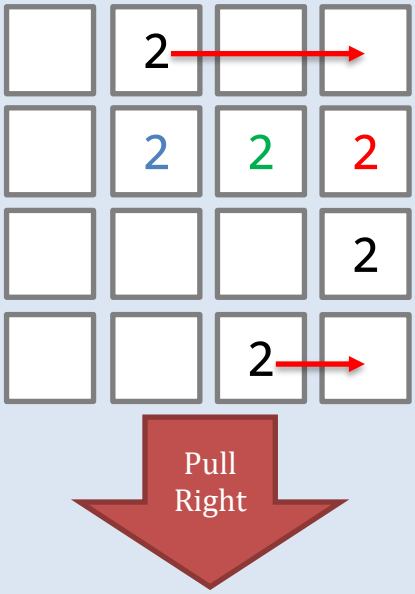
Action #2: Combine adjacent blocks

1. After pushing all the blocks, the game system should combine adjacent blocks of the same number along the push direction.

Action #3: Push the blocks again (after block combination)

Since the combination may leave gaps along the push direction, the game system should push all the blocks again. However, no secondary combination would be applied.

The figure on the next page shows what happens for a user to input the "**RIGHT**" command.



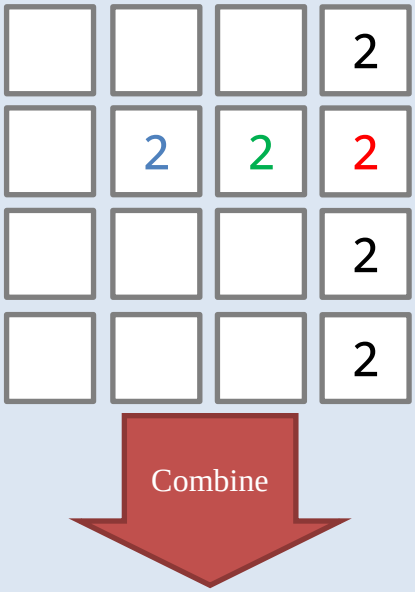
If the user command is "RIGHT", the program *scans each row from RIGHT to LEFT*.

Whenever an empty cell is found, the program scans from right to left:

- to find a non-empty cell, and
- to pull a nonempty cell to take the place of the empty cell.

For the second row, we cannot pull because the row does not have an empty cell.

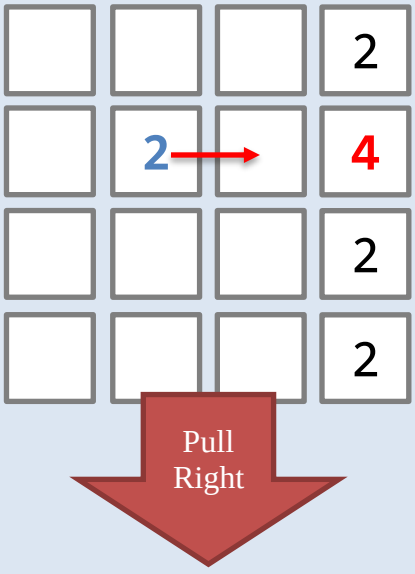
For the third row, the empty cells cannot find non-empty ones on their left.



For combination, the program again scans each row *from RIGHT to LEFT*.

When two adjacent cells are of the same number, the program:

- deletes the value of the left block, and
- multiplies the value of the right block by two.



Last but not least, the combination may leave gaps in the rows. *The program applies "Pull Right" again*.

Note.
No further combination should take place.

2. Project Task

In this project, there are two tasks:

1. You are required to implement the game's logic, and we called this the **basic implementation**. This task allows a human player to input commands to the game in order to play the game until he/she loses the game.

2. You are required to write a computer player that plays the game automatically. The goal of this computer player is to stay in the game as far as possible.

Note: our official grading platform is in a Windows machine using gcc environment.

If you are used to writing code in other systems, such as Mac OS X, you should make sure that you can port and try your code in the gcc environment provided in the lab, and can run your code as expected.

Variations from the original game introduced in Section 1

- When the game first starts, there is only ***one block with the value 2 generated at a random location***.

- For each round, there is only ***one new block with the value 2 generated at a random, empty location***.

- The winning condition is removed (i.e., producing the final 2048 block), i.e., the game will continue until the game board has no empty space for more moves.

3. Suggested Program Design of "2048"

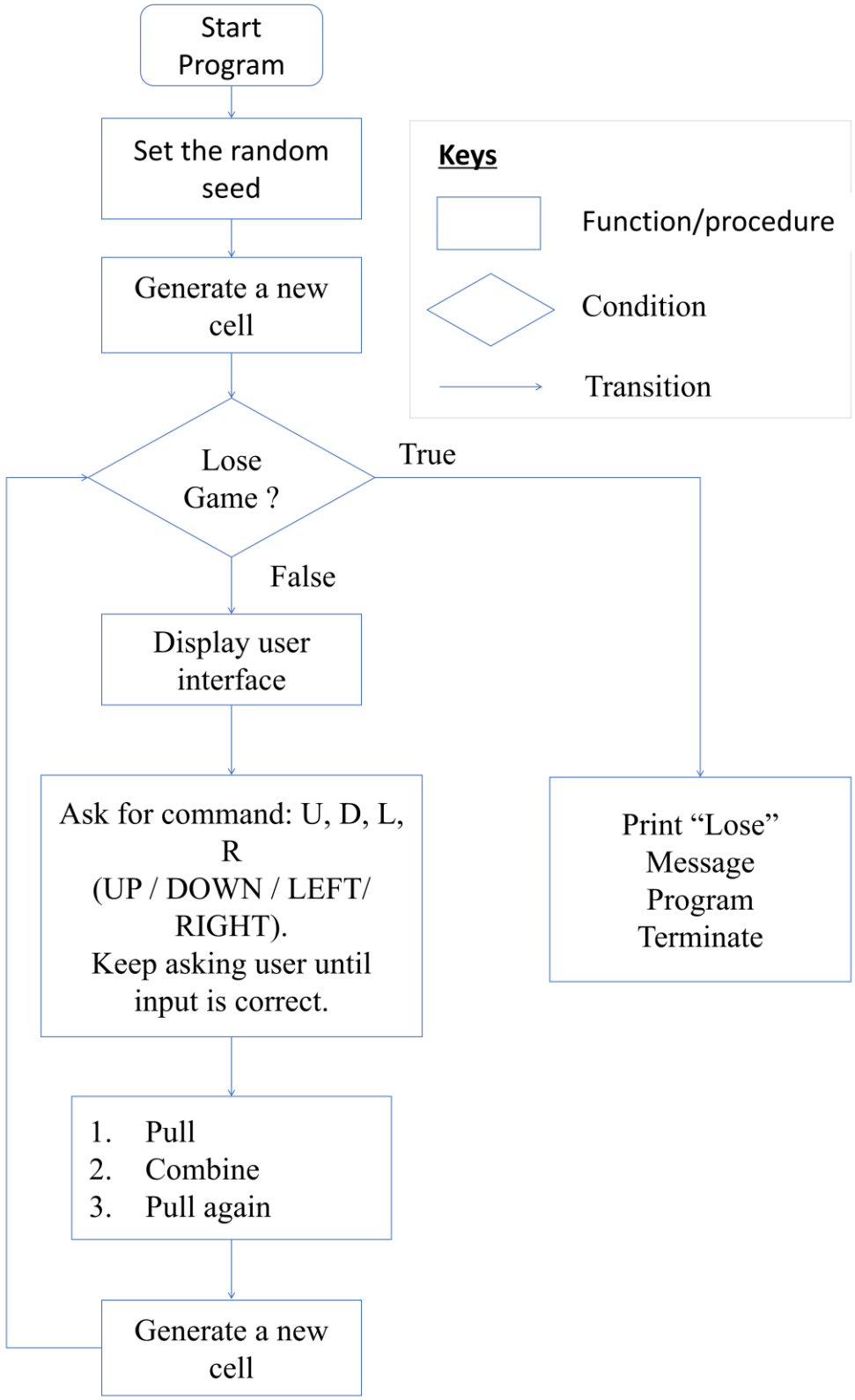


Figure 1. Suggested "2048" Program Flow.

The above figure shows the suggested program design. In the following, we describe the design and requirements of some important functions and conditions in the game.

4. Basic Part

4.1. Function: Set the random seed

You **must** provide a way for the user to set the random seed value by **"command prompt"** for two reasons:

- It allows you to effectively debug the program.
- It allows us to effectively check if your program produces the right outputs.

4.2. Function: Generate a new cell

A new cell of value 2 is generated at every new round. To ensure that the output is consistent with the same random seed, you **must use the pseudo-random generator provided by the TAs**. In particular, you **must**:

- First, ensure that both "tinymt32.h" and "tinymt32.c" files (provided by TAs) are in the same folder as your main program.
- Second, in your main program, you must include the header file "tinymt32.h" and set up the global variable "state":

```
#include "tinymt32.h"
tinymt32_t state ;
```

- Then, put the following at the beginning of your program, such that the pseudo-random number generator can be correctly initialized:

```
uint32_t pseudo_seed = seed ; // previous obtained seed
tinymt32_init( & state , pseudo_seed );
```

- You must use the following code to generate the location of a new cell of value 2 (note that you may create a new function for this piece of code):

```
int i , j ;  // i and j store the position of the new cell
while ( 1 )
{
    i = tinymt32_generate_uint32(&state) % 4 ;  // row num.
    j = tinymt32_generate_uint32(&state) % 4 ;  // col. num.
    if ( game_board[i][j] == EMPTY_CELL )       // e.g., 0
    {
        game_board[i][j] = 2 ;
        break ;
    }
}
```

Why do you have to follow the above procedure?

The "**tinymt32_generate_uint32**" function affects the entire gameplay: the **sequence of random numbers** and so the new **random locations** produced in the gameplay (see lecture "08c. Random"). By using it, we can ensure the same sequence of random numbers produced in your gameplay for the same random seed. Therefore, **you MUST call function "tinymt32_generate_uint32" to generate a new cell using the above procedure rather than using the generic function rand()**.

4.3. Function: Display User Interface

It is up to you on the look of the game board. Below, we provide you with a sample design:

```
+----+----+----+----+
| 128|    |    |    |
+----+----+----+----+
|    |   2|    |  16|
+----+----+----+----+
|    |    |2048|    |
+----+----+----+----+
|    |    |    |    |
+----+----+----+----+
```

The border is to be constructed by three characters: **+** (plus), **-** (minus), and **|** (pipe / vertical bar).

Within each cell:
- **empty cell**: print 4 space characters.
- **number cell**: `printf("%4d", ...);`

4.4. Function: Ask the user for a command (human player)

Then, your program should allow the human player to input a command. The inputs are as follows.

Character(s) Input	Command
U	Pull Up
D	Pull Down
L	Pull Left
R	Pull Right

In particular, you have to write a function to read the user inputs with the standard function name as well as the standard parameter list as followed:

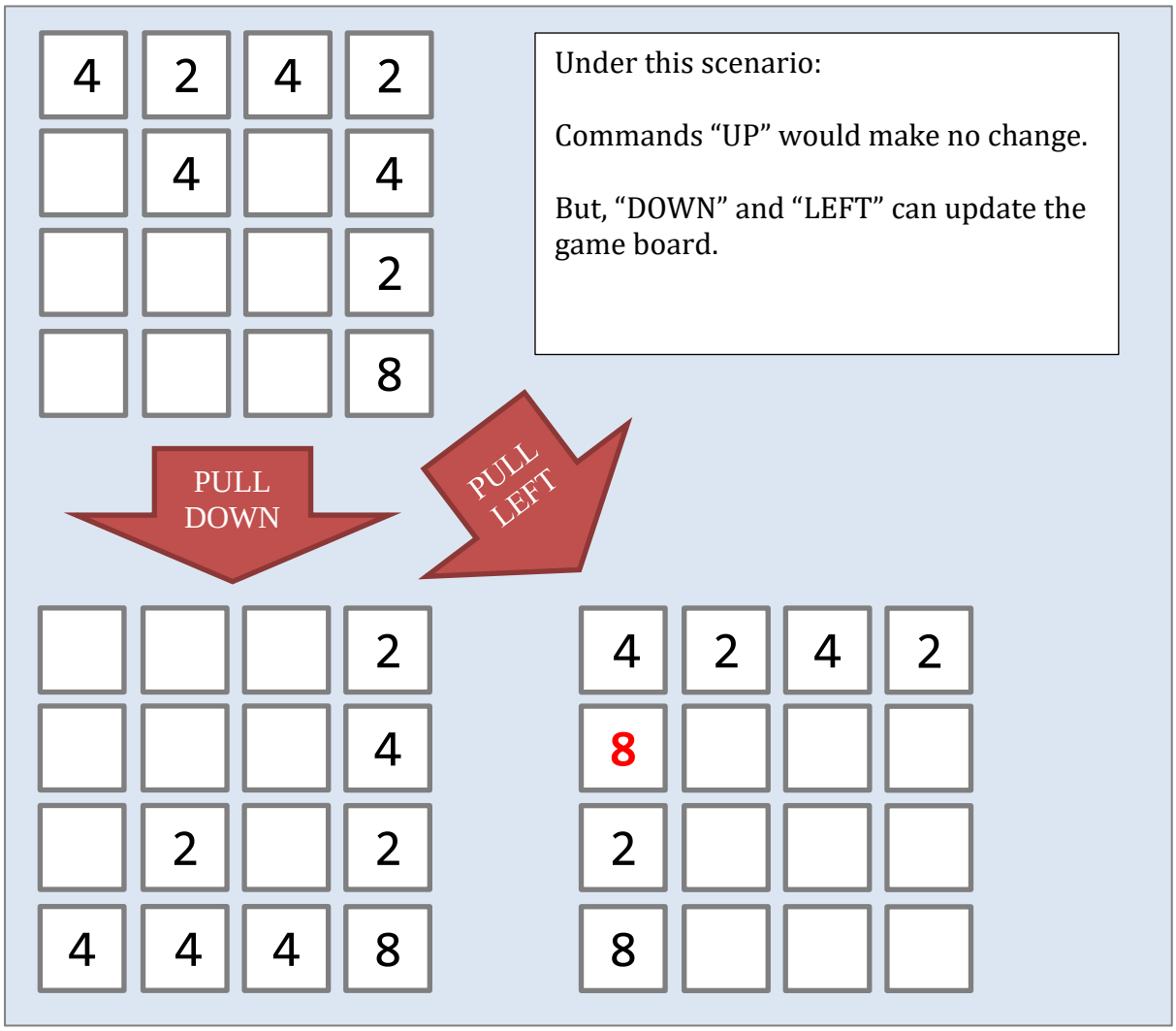
```
int human_player( int game_board[4][4] );
```

Assumptions and Requirements

- There is only one input parameter: the current game board, which is represented as a 4x4 2D integer array.
  - The first dimension indicates the row number in [0, 3];
  - The second dimension indicates the column number in [0, 3];
  - An element "`game_board[i][j]`" represents the current value at the (i-th row, j-th column) location.
  - ***Zero*** means an empty cell.
- The game must print the game board to the "**command prompt**".
- The user must use the keyboard as the input, e.g., press "U" and enter.
- You have to validate the inputs. If a user inputs something wrong, **you have to ask the user again** until the user gives one of the correct inputs.
- The return value indicates the chosen direction:

Direction	Return Value
UP	0
DOWN	1
LEFT	2
RIGHT	3

- There are occasions that the game board does not make any change after applying a particular command. Let us consider the following example:



If the game board cannot make any changes (in terms of using pull & combine to the user-selected direction), then ***the game <u>must</u> report an error message to notify the user***, and ask the game player to enter the command again.

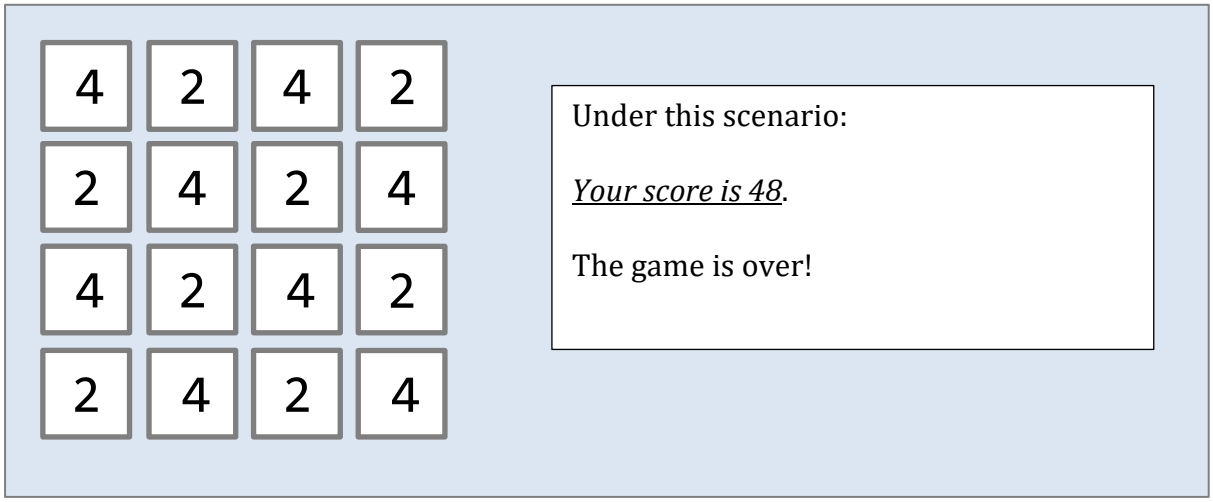
4.5. Function(s): Pull, Combine, and Pull again

This is the core of the game. Please follow the step we described earlier. If you are in doubt about the game rule,

- Visit the official game site and play the game, or
- Describe the scenario you have doubts about in the **Lab Section** or in our **Piazza discussion** group.

4.6. Function(s): Check lose-game and Report game score

When the game board is full of number blocks and no changes can be made to the game board, the player loses the game, see the figure below. The game should then terminate and report the **game score**, which is the **sum of all the numbers in the game board**, i.e., (4+2+4+2) x 4 = 48, for the game board shown below.



5. Computer Player

This is the most challenging part of this project. The goal is to replace the function: "**Ask the User For a Command (human player)**".

In a nutshell, your computer player gives the best choice, i.e., one of the four directions, based on the current game board. In order for your program to be graded fairly, we will use a standard game implementation and plug your computer play code for grading.

Therefore, your AI function should have the following standard name and parameters:

```
int aiplayer_studentID( int game_board[4][4] );
```

- The function name is called "***aiplayer_studentID***".
- The parameter and return value are the same and explained in Section 4.4 (***human_player*** function).

Additional note:

- Your computer player function **must return a value within 5 seconds** (using the computer with gcc compiler in the faculty computer lab); otherwise, we can stop your program from that moment during the demo.
- You have to put your computer player function using a multi-file compilation paradigm with **only 2 files**: aiplayer_<studentID>.h and aiplayer_<studentID>.c

6. Suggested Schedule

Week	Date	Tasks Expected to be done
9	**Nov 8**	**Related knowledge from the lecture:** random number generator **In-home**: Play 2048 to get ideas on how to implement the pull operations and the cell combinations. Start implementing the rough game flow: design the whole project structure (see Section 3 earlier), and then implement and test function by function, e.g., random number generation, display the interface, gameplay, etc.
10	**Nov 10**	**Related knowledge from the lecture:** how to debug this big project with file I/O, and AI program design using the randomized approach as well as the min-max searching algorithm. **In the lab**: you should start designing and implementing the pull operations and the cell combinations. Sample test cases will be given.
11	**Nov 17**	Should try to complete everything for the human player, such that you can buy time to implement a more powerful computer player.
12	**Nov 24**	Complete the whole project (both the human player and the computer player) such that you can buy time to strengthen your computer player.
12	**Pre-test Nov 27 11:59pm**	If you submit your computer player before this pre-deadline, we will try your code with the TA's system and will let you know your results and also whether your code has any problem.
13	**Deadline Nov 30 11:59pm**	Please submit both the basic program (part 1) & the computer player (part 2) by this deadline.
13	**Dec 1**	**In the lecture**: Examination Overview. **In the lab**: **Project Demonstration** on both the basic program and the computer player. **We will do the project demo in the faculty computing lab, ERG 909 on Dec. 1 (15:30 to 17:30).**

7. Requirements

7.1. Requirements of the basic part

Your program starts by taking a random seed and follows a sequence of direction inputs **('U', 'D', 'L', 'R)** until the game ends. Though the online judge (codeSubmit) system is not provided, you have the following set of requirements for consistent and fair grading:

- Your program must read inputs from the keyboard (like the examples shown in Section 3.4), so we can test your code by entering a random seed and then a sequence of direction inputs **('U', 'D', 'L', 'R)** to see if your code produces the expected results.
- To ensure the numbers in different students' gameboards are the **same** for the same input sequence, you must follow **Section 4.2 on new cell generation**.
- We will try some moves that do not alter the gameboard to see if your code can find.
- Your program must print the results to the terminal/command prompt.
- During each round of the game, your program must:
    - Print the current game board;
    - Report if a player's move is illegal, print out a notification and ask the user to input again.
- At the end of the game, you must report the total score the player obtained.

7.2. Requirements of the Computer player

- You must follow the function prototype defined in **Section 5** to implement your AI.
- It should produce valid moves and it should not modify the input game board.
- Your "submitted" AI function shouldn't contain any **print statement**. Otherwise, the system may wrongly judge your results. Note: you may have print statements when you debug your code but remember to comment them out before submission.
- For a fair comparison, your AI function must return an output within a reasonable amount of time, i.e., **within 5 seconds on a desktop computer in the lab**.
- You need to carefully test your code to make sure your code won't crash, e.g., the "**array out of bounds**" error!!!
- No "main() function" in your submitted files is allowed for the AI player part.

8. Submission

We have created two submission boxes on Blackboard:

- one for you to submit the entire program that supports the human player and
- the other one for you to submit your code ONLY for the AI player.

8.1. Submission Format for Basic Part

Your submission for the basic program should be a single **zip file** named as

basic_<studentID>.zip (e.g., basic_1155123456.zip)

which **contains and should only contain** the **.h** and **.c** files in your project. The TA will download your submission files from Blackboard and compile them during the project demo to form an executable game using gcc for testing.

8.2. Submission Format for AI Player (AI function only)

Your submission for the computer player should contain only two files:

- aiplayer_<studentID>.h (e.g., aiplayer_1155123456.h)

  In this file, you **must** define the following function:
  int aiplayer_<studentID> (int game_board[4][4]);
  (e.g. aiplayer_1155123456(int game_board[4][4]);)

- aiplayer_<studentID>.c (e.g., aiplayer_1155123456.c)

  where <studentID> is your own student ID. In this file, you may have more functions for your main AI function above to call.

In the submitted .zip file for AI player, you need to include the above .h file and all the necessary functions inside the above .c file for your AI player to work. To implement a stronger computer player, you may define additional functions in your .c file. However, you **must** follow the above naming convention (i.e. add your own "*_studentID*" at the end of all your function names), so that your TA can take your .c file and compile it with other student's .c files without having any ambiguity, i.e., same function name from different students. We may deduct your project marks if you fail to follow this convention. Also, use .c rather than .cpp or .C as the extension of your source code file.

Submission Note:

- After preparing your submission file in the above format, you need to upload one zip file (see above) to each of the two submission boxes on Blackboard before **Nov 30, 23:59**. Please see **Blackboard** for the details.
- You may join the pre-test by submitting the zip-file before the pre-deadline: **Nov 27, 23:59.** Please see the **Suggested Schedule** for the details. You can still modify your code before the hard deadline. **Please note that the scores obtained in this pre-test are for reference only. Only the scores in the final testing would be counted towards the final grading.**
- You may submit multiple times for each submission box, and we only take the last submission before the hard deadline as your submission for grading.
- Note that the deadline is ***STRICT*** and the project is a BIG COMPONENT in the overall grading scheme of this course. Please start early and prepare early.

9. Grading

This project is **individual work**. Each student has to submit his or her own implementation with the requirements listed in this specification.

Grading of the project is done by an interactive demonstration of your work with the teaching team on **2021 Dec 1 (Wed)**. We will try your game in front of you so that you can learn the defects of your program (if any). However, **no code modification** would be allowed during the demonstration.

9.1. Marks

The total marks of the project are 20% of the course grade.

- 16% of the course grade goes to a correct implementation of the basic game logic. During the project demo (on the demo day during the last lecture), the TAs will compile your program, and then test your program in the following ways:
  - interactively try your program to play games; and
  - pipe some sample text files in the above format into your game executable and see if the results are correct, as expected (you will learn the meaning of "pipe" in class later).
- 4% of course grade goes to the computer player with a ***game score of at least 100***. In case that your computer player cannot achieve a game score of 100, we give partial marks based on the following scheme:

Score (x)	Partial Marks
x < 40	0
40 <= x < 60	1
60 <= x < 80	2
80 <= x < 100	3

  Since you may be using random numbers in your computer player implementation, we will give you **five trials** and choose the one with the highest game score to determine the fate of 4% of the course grade for the computer player.

9.2. Bonus Ranking Marks

In order to encourage you to achieve the best possible AI program, we will give at most 3% of the **course grade** as bonus marks. We will run your computer players 5 times with different random seeds, and take the highest score among them. Bonus scores will be given to the top 15 students:

Ranking	Bonus Marks
1	3%
2-3	2.2%
4-7	1.4%
8-15	0.6%

9.3. Academic Honesty and Project Archival

Every submission will be inspected using plagiarism detection software. The worst possible punishment is to have this course failed. For regulations and details, please refer to the following URL:
https://www.cuhk.edu.hk/policy/academichonesty/Eng_htm_files_(2013-14)/p06.htm

Last but not least, you are required to include the following comment block at the beginning of every source code inside your submission package.

```
/**
 * ESTR 1002 Problem Solving by Programming
 *
 * Course Project
 *
 * I declare that the assignment here submitted is original
 * except for source material explicitly acknowledged,
 * and that the same or closely related material has not been
 * previously submitted for another course.
 * I also acknowledge that I am aware of University policy and
 * regulations on honesty in academic work, and of the disciplinary
 * guidelines and procedures applicable to breaches of such
 * policy and regulations, as contained in the website.
 *
 * University Guideline on Academic Honesty:
 *   http://www.cuhk.edu.hk/policy/academichonesty/
 * Faculty of Engineering Guidelines to Academic Honesty:
 *  http://www.erg.cuhk.edu.hk/erg-intra/upload/documents/ENGG_Discipline.pdf
 *
 * Student Name : xxx <fill in yourself>
 * Student ID   : xxx <fill in yourself>
 * Date         : xxx <fill in yourself>
 */
```

-- END --