COMP2014 Object-Oriented Programming

Assignment 1

Submit Deadline: 5:00pm Friday 16 Sep 2022

1. Problem: Eater Game

Eater is a simple board game for two players, Passer and Eater, played on an m*m grid board (m>2). An example of a 9x9 Eater board can be found in **Figure 1**. Red is painted on the top and bottom edges. Blue is painted along the left and right edges. Players may play with an unlimited number of red or blue markers.

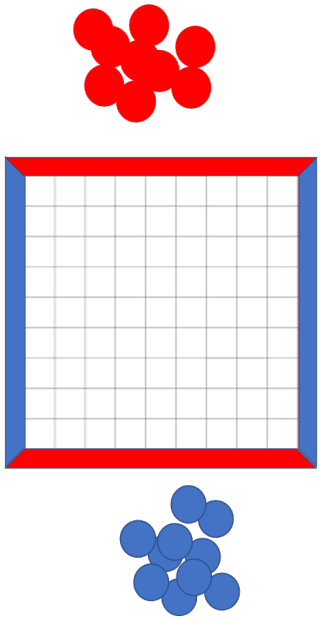


Figure 1: A *9*9 Eater Game board*

The rules of the game are as follows. The game is played simultaneously by two players, each placing a marker of their own color into a cell that has not yet been occupied. When both players choose the same cell to mark, the Eater's marker remains and the Passer's marker is eliminated. Otherwise, the markers remain on the board until the game is completed. From the top to the bottom, the Passer seeks to form a connected path of its own markers, which can go down, left, right, down-left, and down-right, but cannot return. Once a connected path has been established, the Passer wins. The game then terminates. As long as the Passer does not have a connected path and the board is full, the Eater wins.

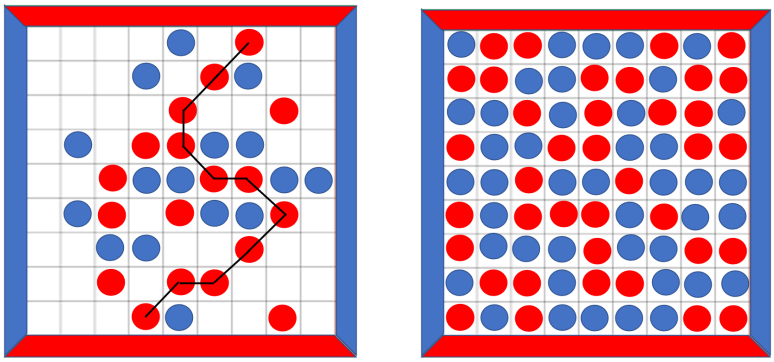


Figure 2: *Examples of winning situations:*

Passer player wins (left) and Eater player wins (right)

Two examples of winning situations are shown in the above figures. A case in which no player has yet won is shown in the following figure.

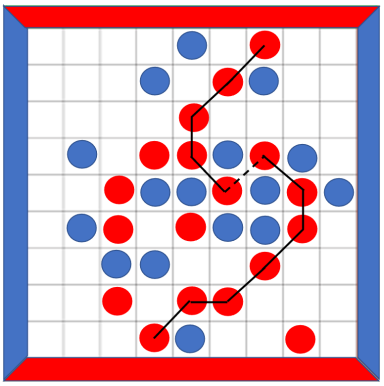


Figure 3: *The Passer player has not yet won*

(The path goes up thus is not viewed as a valid connected path)

Note that the game continues even if the board is not full and the Passer has not won, for instance when the Eater has built a solid line that occupies an entire column.

2. Task specification and requirements

In this assignment, you are required to develop a computer program that allows humans and computers to play the Eater game on any size board. In order to simplify your program and make the task unique, you are restricted to displaying the game board in purely text format as shown in Figure 4, where P indicates the moves of the Passer and E indicates the moves of the Eater. **No graphic output** is allowed.

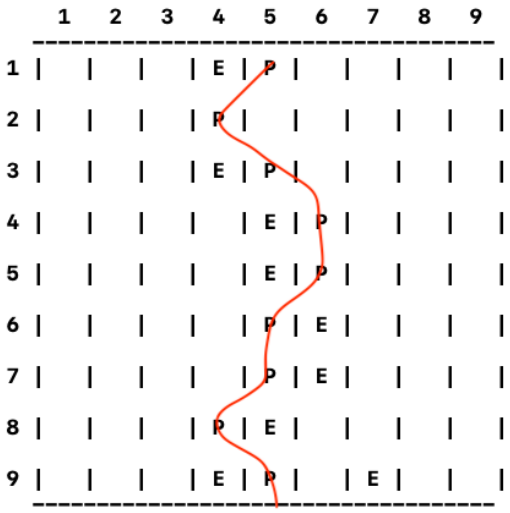


Figure 4: *An example of board layout*

Ultimately, you are expected to implement the following functionalities:

1) Display game board in any size (no less than 3 x 3).
2) Accept players' moves and update game board accordingly.
3) Terminate a game if the Passer wins or the game board is full.
4) Create a computer player that can automatically generate rational moves.
5) Create a smart computer player that can play the game with certain intelligence.

You are required to write your program in C++ using object-oriented paradigm. Other programming languages, such as Java or C, are not allowed in this assignment.

2.1 Pass Level

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

Task 1 (54%): Write a C++ program that displays the game board and accepts players' inputs of moves and update game board for each step until the board is full.

Task 2 (10%): Use a dynamic array to store board status so that the size of the board can be set in real time.

Requirements and hints:

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

(2). Your program should be able to accept human input moves. A move of one player is represented by two integers, *i.e*. the coordinates of a placement. Players make moves simultaneously. You may take Passer's move first and then take Eater's move, but the

board should not be updated until the moves from both players are received. Your program should reject invalid input of moves (coordinates out of range or positions occupied) and request for another attempt.

(3). For **Task** 2, your program should allow a user to input the size of the board. Input can be any integer no less than 3. To make the game board displayable on a computer screen, you may restrict the input of size to something up to 15. You are recommended to create the dynamic array using new statements in the constructor of your Board class and destroy the array using delete statements in the destructor of your Board class.

(4). Your code must be in OO-style. Purely procedural code will receive zero marks.

Hint:

1) To represent a gameboard, we may use 1 for the Passer's move, -1 for the Eater's move and 0 for an unoccupied cell.
2) The indices of an array start from 0 and end with *boardsize* -1. The coordinates of moves shown to human start from 1 and end with *boardsize*. You need to take 1 off from the user's input of a move to map the move into the cell of the two-dimensional array.
3) You are highly recommended to start on **Task 1** without using dynamic memory allocation. In this case, you may set the board size as a constant, say:

   ```
   const int boardSize = 5;
   ```

   It should be easy for you to change your code into a variable board size once dynamic memory allocation is used.
4) As pass-level code, moves can be taken from human input.

2.2 Credit Level

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

Task 3 (5%): Create a random player that can generate valid moves automatically for either player.

Task 4 (5%): Create a computer player that simply plays a straight line as its game strategy.

Requirements:

(1). Make at least two options:

- human player vs random player
- Straight line player vs random player

(2). Swap the roles of Passer and Eater for each type of players for testing purpose.

Hint:

1) In credit level, human player, random player, and straight-line player can be separate functions in Board class. These functions can generate valid moves.
2) Options of players can be hard coded in this level. You may change to other options by changing the code.

2.3 Distinction Level

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

Task 5 (10%): Add a member function to the Board class to check if the Passer wins, *i.e.*, the player has a connected path from the topmost column to the bottommost column.

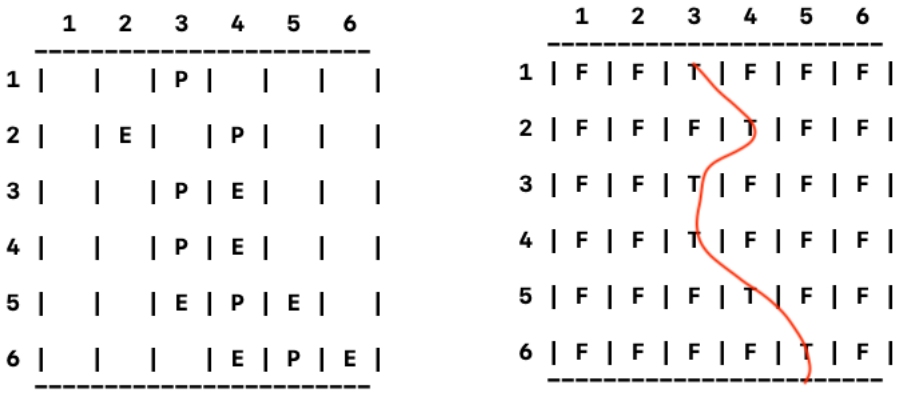
Requirements:

(1). A game should terminate as soon as the Passer player wins. The game continues if the Passer has not won and the game board is not full.

(2). Set both players to be random to test your check-winning function.

Hint:

1) Many algorithms can be used to check the Passer's winning status. A typical algorithm is to create a two-dimension bool array of the board size (m*m array of **`bool`**). A cell is **`true`** if the Passer has a connected path from the top side to this cell.



Note that a path can only go from top to right, *left*, *down*, down-*left,* or *down-right*. Any **`true`** in the bottom row represents a winning case for the Passer.

2) As a practice, you may start with the situation when the Passer player has a straight path from the top side to the bottom side. Extend it to more complicated cases. You will receive partial marks if your code can't identify all winning situations.
3) You are encouraged to design your own algorithm to solve the problem.

2.4 High Distinction Level

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

Task 6 (10%): Implement a computer player with certain intelligence more than simply

creating a straight line.

Task 7 (5%): Reconstruct your code to present good abstraction, encapsulation, and inheritance.

Requirements:

(1). Your implementation must contain at least three classes, **`Board`** class, **`EaterGame`** class and **`Player`** class to demonstrate your skill of class communication.

(2). You may separate the implementation of players into several classes: a base class (generic player) and a few derived classes, such as the human player, the random player and computer players (simple or smart, either as Passer or Eater).

(3). Your computer player does not need to be super smart but should be able to win a random player in any situation. The more intelligent your computer player is, the more marks you will receive. We will check the strategies you implement for your smart players.

Hint:

1) Learn some strategies by playing the game yourself and then see if you can implement these strategies into your computer player (your computer player will then have your intelligence :-).
2) Many AI based approaches, such as heuristic search and Monte Carlo tree search, may be used for this game.
3) To make your code more OO, you may start with a less OO program and improve it with better abstraction, encapsulation, and inherence. As a general practice, it takes more time in coding if you keep working on a bad OO program than to rewrite your program with better OO-style.

2.5 Tasks for Advanced students or students like more challenges

If you are an advanced student, you may conduct the following tasks for your advanced activities. Otherwise, you receive bonus marks for the following tasks (see more details in the marking sheet).

Task 8: Implement a computer player using Monte Carlo approach.

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. 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 except the IDE library.** No group work is allowed in the assignment. You may use part of code I provided in the lectures and/or practical tasks, but no part of your code is taken from any other resources unless specified in a separate document.

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.

**************************/
```

In case your code contains any existing code that is not written by you, you must specify that part of code and its resources in a separate document.

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) 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 9. Your tutor will check your code, especially your understanding of your code. **All comments must be removed from your source code when you demonstrate. You will receive no marks if you fail the demonstration, including the case that you miss the scheduled demo time.** Note that it is your responsibility to get the appropriate compilers and/or IDEs to run your program. **The feedback on your work will be delivered orally during the demonstration.** No further feedback or comments will be 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 during your first demonstration, you are allowed to improve your work in the following week (**maximal grade is 50% in this case**). If you show us your code earlier than the deadline, we can mark your work earlier (subject to the availability of our time) and we can possibly allow you to improve your work if you do not satisfy the marks we give.

6. Additional information

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 forum will be created in vUWS to encourage you to learn from each other. Note that we encourage students to learn from each other but work has to be done individually.