ITI 1121. Introduction to Computing II

Assignment 2
Submission **deadline**: See **Brightpsace**

Assignment: Group of 2 students Maximum.
All the rules of the first assignment apply.

Reminders: The submitted code must be yours.
You are also responsible to **not** expose your code to others.

The submission must be done on Brightspace, **before the deadline**.

- Only submit your **java** files, do **not** submit any .class files.
- You must test your programs to make sure they give the correct results.
- A program that gives a **compilation error or an execution error will receive a mark of zero.**
- You have to put **comments** in the programs. The comments are **not** needed for each line of code. However, they have to explain in **simple English** what the program does.
- Some java files are provided, **they should be used** and the content already there **cannot** be modified. Which means you need to **keep** the existing code and **add** your code to it.
- Important: one submission per group is enough, however both students are responsible to make sure the submission was done before the deadline.

Academic Integrity

https://www.uottawa.ca/administration-and-governance/academic-regulation-14-other-important-information
https://www.uottawa.ca/vice-president-academic/academic-integrity

By submitting this assignment, you acknowledge the following:
1. I have read the academic regulations of the University of Ottawa regarding academic fraud.
2. I understand the consequences of plagiarism.
3. With the exception of the source code provided by the professor for this course, all the source code is mine and my group partner only.

ITI 1121. Introduction to Computing II

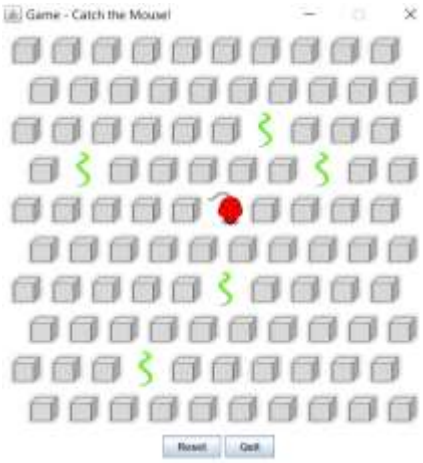
In this assignment, you will program a game, “Catch the mouse” with a simple user interface.

The game’s user interface shows a board, made of a cubes, with the same number of rows and columns. The board contains grey (free) cubes, green (snakes) cubes, and one red (mouse) cube.

The player tries to catch the red mouse before it reaches the edge. If the mouse (red cube) reaches the edge, the player loses. If the player clicks any square, there are four cases:

- If the square is already a green cube (a snake) nothing happens.
- If it is a square that is a neighbor of another snake, it turns into a snake too.
- If it is a square that is not a neighbor of any other snake, nothing happens.
- If it is a red cube (a mouse) and it is currently next to any snake, the game is won.

Note: a position is a “neighbor” of another one if we can go from the first position to the second in a single move, without skipping any cube.
The mouse cannot move to any of the squares already clicked by the user, which turn into snakes.
If the player clicks it before it reaches the edge, then player wins!
The image below is an example of the game starting screen.
The number of rows is equal to the number of columns.



The player wins when he can click on the red cube (the mouse). It can only be clicked if it is neighboring one of the green cubes (it is reached by the snake).
If the

At the start of the game, some of the cubes are already green. This is done with a random selection among all the board cubes. The maximum number of green cubes allowed is a constant number of five: 5. All other cubes will initially be free cubes, and one cube red.

The starting position of the red mouse is semi-randomly selected: which means random however it should be close to the center of the board. Here is the procedure to select starting red mouse position:
- If the board has an even number of rows (and columns), then the red cube is randomly positioned on one of the central four cubes.
- If the number of rows (and columns) is odd, then it is randomly positioned on one of the central nine cubes.

At each step the player can only select one of the six neighboring cubes of an already selected cube (green snakes cubes).

The player wins when the red cube is clicked on while neighboring a green cube.
However, if the red mouse is clicked while **no** green snake cube is its neighbor, then nothing happens.

The roles of the classes is different, basically we have three categories:

I- The state of the game at any given moment in time
II- The user interface
III- The processing/logic of the game

I- The Game State (30 points)

First, build the state of the game using the class GameState and the class Point. There is only one instance of the class GameState (instantiated by GameLogic as we will specify later).
GameState is responsible for holding the state of the game at any moment in time. This includes

- The three possibilities for each square in the game board:
  1- It is free: the cube occupying it is grey, it hasn't been selected yet by the player
  2- It is already selected: the cube occupying it is a green snake
  3- It is occupied by the red mouse cube

- The location of the red mouse cube: The starting location of the red cube, as explained earlier: on an even board, the position is randomly selected among the four central cubes, and on an odd board, it is randomly selected among the nine central locations.

- The current status of every cube on the board
- The size of the board.

It also provides the necessary setters and getters, so the the GameLogic and the user interface can check the status of any cube, and the GameLogic can change the status of a cube or move the red mouse cube. Finally, it provides a way to initialize the game, placing the red cube randomly as explained earlier, and preselecting some of the other cubes as green snake cubes. In our case, we are going to use 10% as the probability that a cube that isn't the red cube is initially selected as a snake cube.

II- The User Interface (35 points)

We build a simple User Interface for the game. This will be done with two classes.

- The first class is GameUserInterface, which extends JFrame. It is the main window of the application. It shows two buttons at the bottom (see the image in the first page). These buttons are to reset and to quit the game. GameUserInterface instantiates the second class, BoardUserInterface.

- The second class is BoarUserInterface, it extends JPanel. This class is the one that contains the board.
  A board, of size n, is made of a series of cubes: n lines and n columns of cubes.
  To implement the cubes, we use the class Cube which extends the class JButton.
  So the class BoardUserInterface provides the user interface of the board. It extends JPanel and lays out a two dimensional array of Cube instances.

Layout of the game board.

A challenging aspect of the User Interface is the layout of the board. If the board was a normal square, it you be really easy to layout the Cube instances on the Panel. However, the actual board of the game is **not** a perfect square. Instead, each line is shifted left or right when compared to the previous line. Your board layout should look like the one in the image shown on the first page. We don't have a way to create that Layout directly.

In order to achieve this layout, we will use two steps within **BoarUserInterface**:

1- We will have each row of Cube instances on its own instance of a JPanel class, on which a FlowLayout manager will be used. If you look at the documentation of JPanel, you will see that it is possible to add a border to the panel.
   The icons used by the Cube instances are squares of size 40 pixels. So we can add a border of size 20 for example, at the correct location, to obtain the desired layout.
2- The set of JPanels can then be added on the BoarUserInterface JPanel using for example an appropriate GridLayout manager.

Note: instances of both GameUserInterface and BoarUserInterface classes have buttons. However, these instances are not the listeners for the events generated by these buttons. This is handled by the gameLogic.

III- The GameLogic (35 points)

The class GameLogic is the one responsible for the logic/processing of the game. The constructor of the class receives as parameter the size of the board.
The instance of the GameState and the GameUserInterface are created in this class.
The GameLogic is the class handling the events/clicks generated by the player. It implements the logic of the game. During the game, after the player selects a new cube, the GameLogic must decide what to do next.
After each click of the player, the three cases need to be covered: the player won the game, the player lost the game, or the game continues.
If the player won: a message box should be displayed to inform the player, when the player clicks "ok", the game is reset.
If the player lost: a message should be displayed accordingly.
The green snake cubes areas grows one cube at a time, in one of the six cubes that surround a green snake cube.

Cases of winning or losing the game.

In the two cases, the GameLogic must inform the player of the result. In order to achieve this, the gameLogic can use the class method showMessageDialog of the class JOptionPane.
To continue the game, the gameLogic must implement a strategy for a selection of the next move for the red cube.
You may make a selection to move the red cube to any randomly selected cube that is not occupied by a green cube. If the red mouse reaches the edge, the player lost.

CubesGame.java is the java class containing the **main** program to start the game.

CubesGame creates the instance of GameLogic and starts the game. CubesGame is the class that contains the main and should be run by the user. At the execution of the game, the user should pass a parameter to main. This parameter is the size of the game board. This size is valid if it is between 4 and 20. If a valid size is not passed, an error message is simply displayed, and the user should try to run the program again with the correct parameter.