CSC 171 PROJECT 3

DUE SATURDAY MAY 2ND AT 11:59PM

Purpose

The purpose of this project is to bring all of the skills that you have learned throughout the semester to bear and solve a problem of your own design with limited criteria. Hopefully you will learn that programming can be fun and rewarding as you define and accomplish your own goals. Be creative!

By now this should be obvious, but **<u>any work that you submit must be your own</u>**. If you use code from the internet and provide a citation, you will receive a grade proportional to the code that you authored. In other words, if 90% of the code that you submit is from a solution that you found on the internet, your maximum possible grade for this project will be 10%. Please note that retyping code that you find **does not constitute authorship**. If you do not provide citation for code that you reuse, you will receive a grade of 0 and be reported to the Bureau of Academic Honesty.

Problem Domain: Gaming

Implement a graphical game. Below is a list of suggestions. These are here to give you some ideas. You may choose one of these, or pick another game provided that it meets the requirements below. You may even make an original game of your own design.

1. A card game, such as Poker or Solitaire.
2. A “match three” style game like Bejeweled or Candy Crush Saga.
3. A line-making game like Tetris.
4. A side scroller like Super Mario Bros.
5. A matching game, like Memory.
6. A board game, like Monopoly, or Checkers.
7. An “in-a-row” game like Tic-Tac-Toe or Connect Four.
8. A defense game like Asteroids or Space Invaders.
9. A “find the bombs” game like Minesweeper.
10. A maze game, like Pac-Man.

Please note that the level of complexity varies quite a bit from game to game in the list above. Some games are very simple and will be easily implementable using the material that you already have at hand. Others will require you to do additional research in order to make them work. It is up to you to decide whether or not you want to tackle one of the more complicated options.

There is only a handful of lectures remaining this semester, and therefore only a limited amount of material that we will be able to cover. It is likely that many of the games above will not be possible to implement based solely on what you have learned by the end of the semester. You are encouraged to do your own research and look up what you need to implement the game of your choice. If you do use advanced techniques not covered in class, mention them in your README.

Think carefully before choosing which game you want to implement. You will receive credit based on YOUR implementation of the game that YOU choose. If you choose a difficult game and fail to implement it, your grade will reflect that decision.

For full credit, your game must meet the following criteria.

1. If mimicking an existing game, such as one of those suggested above, you must make at least one fundamental and original change to the rules of the game. For example, if making a Tetris-clone you might consider changing the available shapes, or allowing players to make their own custom shapes before the game begins, or having the pieces move from the bottom up rather than from the top down. Note that a “fundamental rule change” is not something like using “As” and “Bs” instead of “Xs” and “Os” in a Tic-Tac-Toe game. It must be something other than cosmetic changes to the game. You MUST explain your rule changes in the README.

2. The game must allow at least two human players to compete against each other, either concurrently or by taking turns.

3. The game must determine when the game is over, either as a result of a loss, a win, or a draw.

4. The game must run in a window and support a graphical user interface of some kind. It must use at least one image, and must include at least one shape drawn programmatically using a Graphics object.

5. You must use at least one design pattern in your game. It does not have to be one of the patterns discussed in class. Describe the pattern, and why you chose it, in your README.

This is not an art project. While slick, pretty graphics are always appreciated, the goal of this assignment is not to spend 200 hours in PhotoShop or Manga Studio creating custom images for use in your game. This assignment is about coding. Please spend your time there. If you use images from the internet, please include a citation in your README and in the comments in your code.

Additional Enhancements (Extra Credit Opportunities)

You may earn extra credit for any enhancements above and beyond the most basic implementation of the game that you choose. You will also receive extra credit for choosing (and successfully implementing) a more complex game. You may receive up to 30% extra credit in total.

Grading Criteria

You will be graded according to the following criteria.

Class Definitions (20%)

You are expected to practice sophisticated class design including use of inheritance, polymorphism, and design patterns. Classes should have low coupling (i.e. as few dependencies as possible, especially bi-directional or circular dependencies) and high cohesion (i.e. a singular purpose implemented through highly specific and related state and behavior). Static methods and variables should only be used when appropriate. Include a sketch of your core class hierarchy in your submission. This could be a digital photo of a whiteboard or drawing on paper, or a diagram created in some other way. It does not have to represent your *final* design, but should demonstrate the thinking that went into your *initial* design *before* coding started.

Game Implementation (40%)

You will be graded based on your implementation of the requirements in the problem domain section. The first requirement, implementing some fundamental and original change to the rules, is absolutely required for *any* credit in this section. You must clearly describe the change that you have made in your README.

Testing (20%)

You should write tests for each of your non-GUI classes. It is strongly encouraged that the unit testing capability integrated into Eclipse is used to implement your tests (jUnit). You can find lots of documentation describing jUnit in Eclipse online. You may also simply write a separate test for each of your classes that includes a main method to test the class. For example, if you implement a “GameBoard.java” class you would create a “GameBoardTest.java” class that uses a main method to create and test a “GameBoard.” You will **lose credit** if your test methods are implemented in your classes (as opposed to a separate class).

Code Style (10%)

- Modularity of design - Objects that make sense. High cohesion, low coupling.
- Comments - Where appropriate!
- Names - Variables, Methods, Classes, and Parameters
- Indentation and White Space
- Packages, packages, packages.

Documentation (10%)

In addition to submitting the Java code (.java files), you are required to submit a README text document which specifies the following:

1. A one paragraph description of your class designs.
2. A description of the fundamental, original changes that you made to the rules of your game. **Cosmetic changes do not count**.
3. A one paragraph description of your testing strategy.
4. Detailed instructions on how to compile and run your application and your tests. This should include command line arguments (if any), appropriate responses to any prompts, and instructions for using your GUI.
5. Instructions describing how to play your game.
6. Full documentation for any enhancements that you wish to be considered for extra credit. YOU must explain what your enhancements are, and how the TA should enable them or otherwise detect their presence.

HAND IN

Projects will be submitted as ZIP archives via Blackboard. You will be given at least 2.5 weeks to complete the project, and there will not be a lab due the same week that the project is due. This is more than enough time to complete the project. You will also have unlimited attempts to submit your project (only the last submission will be graded), which means that you will be able to upload your solution as soon as you have something working, and try for improvements or extra credit afterwards. Because of this, late submissions **will not** be accepted.

You will be expected to submit:

- Java source files and compiled Java class files. You must also submit any additional files required to run the programs.
- Documentation described in the section above. If you submitted any additional files, make sure to explain their function in your documentation.

Grading

Each TA will compile and run your code using your instructions. If your program produces only partial functionality, providing your own test cases will give you partial credit for the parts that operate correctly. Each TA will also conduct a code review to evaluate the correctness of your code and the code design.