CS-1181 Project 3: GUI-based Game

PURPOSE: The primary goal of this project is to develop a more in depth understanding of object-oriented programming by developing a project in an object-rich environment. Java Swing serves as an ideal platform where you can develop and demonstrate mastery of object inheritance, class contracts, multiple inheritance (via interfaces), event processing, and (eventually) concurrency. This project will require the development of a simple game using Java.

PROBLEM: The project will be a simple game of your choice. It can be a novel game or your own take on a classic. You have a great deal of freedom in the design of this game. You can do something as simple as "concentration", or you can do something more complex including multiple levels, sophisticated images, simple AI, and/or sound. Your only constraints are your knowledge and available time. **Additionally, your game may not be any kind of interpretation of cookie clicker.** A (small) portion of your overall grade will be based on a subjective measure of how your game stacks up to the other students'. Your game MUST meet, as a minimum, the specifications listed below.

REQUIREMENTS: Note, if you have done this project previously (for example because you are re-taking this course) you must implement a different game. Duplicate submissions will not receive any credit.

1. **Game Window:** There will need to be a game window implemented as a JFrame.
2. **Instructions:** Your game must provide instructions to the player.
3. **Visual Elements:** You must have at least three different types of visual elements (buttons, labels, text fields, images, drawn shapes, drop-down menu, etc.) – at a minimum! Please refer to your instructor's slides, if you are having trouble coming up with different visual elements.

   Additionally at least two *different* visual elements must be capable of movement or change. For example, clicking on a button that represents a playing card could flip the card over to show its value, or correctly typing in the answer to a riddle could cause a smiley face to appear.

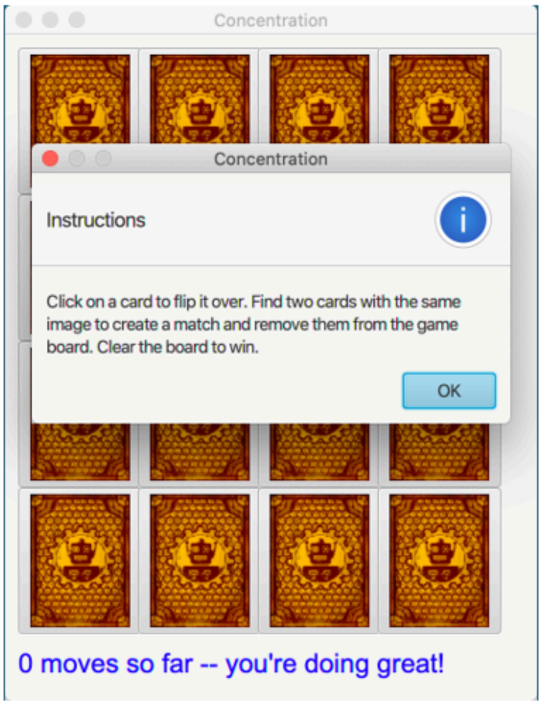
5. **User Interaction:** There must be a way for the user to interact with the game that directly correlates to their score or other measure of success. In other words, the game should not be playable without human involvement.
6. **Randomness:** Your game must have some degree of randomness. For example, if you are creating a memory game, the cards should be in a different layout each time. In a quiz game, the questions should be presented in a random order. This should be *clearly labeled*. If you are unsure if your game meets this requirement, please reach out to your professor or TA.
7. **Score:** Your game should have some sort of score or goal. The level of success/total points should be displayed.
8. **OOP:** Be sure to design your game using good object-oriented coding principles. If your game only has one class, you will receive a **50% deduction.** Note, to meet this requirement, you must have two *fully-developed* classes.

   Make appropriate use of collections like arrays or ArrayLists and loop over them to avoid unnecessary code repetition. If relevant, consider storing data needed by your game in a file so that it can be modified easily.

The specifications above are fixed. The example below is just an example. Feel free to use the example game below or develop your own.

EXAMPLE: Use the sample screenshots below as a guide for understanding the project. These screenshots are from a barebones implementation of the project requirements. If you have a hard time thinking of your own game idea in the time provided, feel free to base your project off of this example – however, you are not required to build the game below. This is just an example of a game that meets the project requirements if correctly implemented.

This example game is called Concentration (or sometimes Memory). The player flips over two cards at a time in an effort to find matched pairs. Once a matched pair is found, those two cards are removed from the game board. The goal is to clear all the cards from the game board in the smallest number of moves possible. The images below show, respectively, the game instructions, the beginning state of the game board, the state of the board in the middle of a turn, and the state of the board when the game is over.

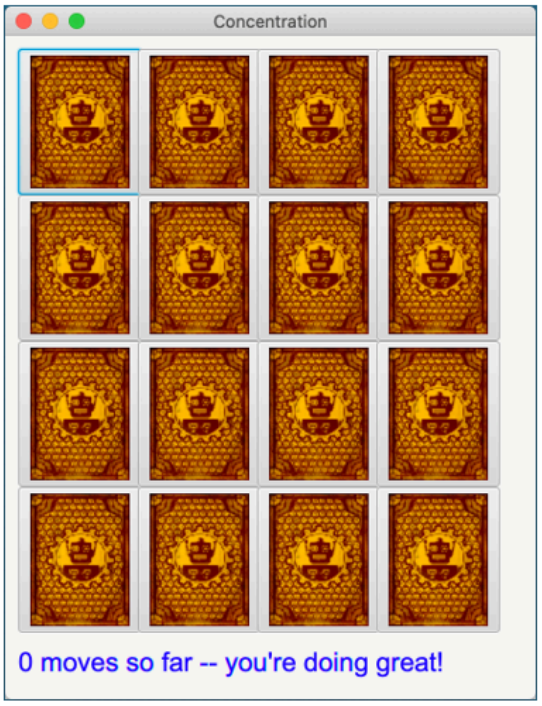

Concentration

Concentration

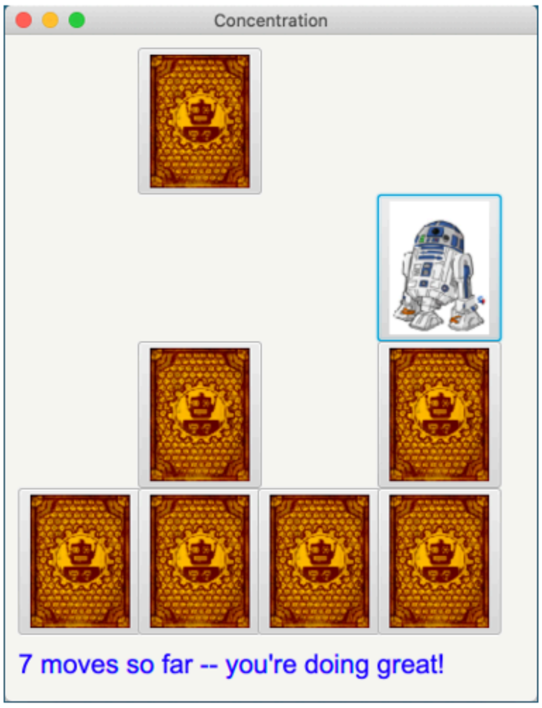
Instructions

Click on a card to flip it over. Find two cards with the same image to create a match and remove them from the game board. Clear the board to win.

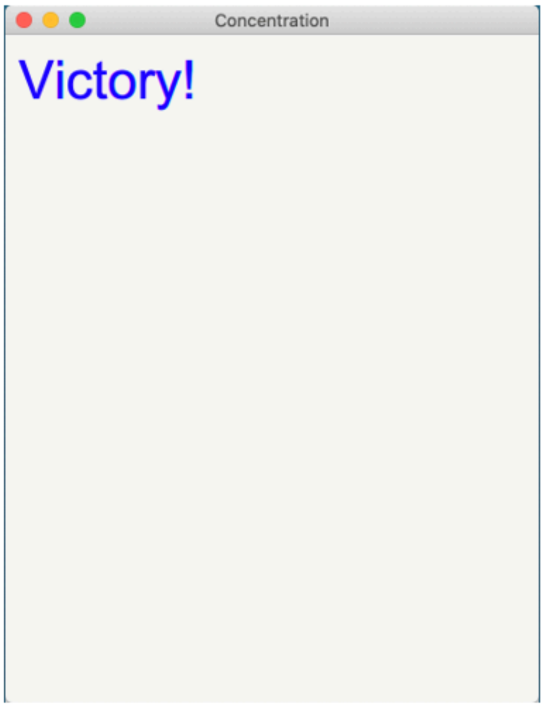
0 moves so far -- you're doing great!


Concentration

0 moves so far -- you're doing great!


Concentration

7 moves so far -- you're doing great!


Concentration

Victory!

ACADEMIC INTEGRITY NOTE: You are welcome to search online or ask others for help in implementing small pieces of your game, but the overall product must be your original work. For example, if you want to play a music clip if the user wins your game. You could look up “how to play a music file in Java” and use the four or five lines of code you find in your program. Be sure to include a comment indicating where you got the code from. It is not okay to google something like “how to write tic-tac-toe in Java” and use the 100+ lines of code you find in your program. Think of it like writing a paper — it’s fine to use a dictionary or a thesaurus to find a word you need, but it’s not okay to take most of your paper from something you find online. If you have any confusion about whether or not something is allowed, ask your professor

Be careful not to bite off more than you can chew on this assignment — start by getting an extremely basic game working and then work on making it more advanced if you have time. Focus on good software design principles — have a logical division of program functionality into classes and methods.

RUBRIC:

(60 pts) **Base Functionality**

[5] Game instructions clearly communicated

[10] Game window includes visual elements of at least three different types

[10] At least two different visual elements are capable of movement/change

[10] User input updates the game in a meaningful way

[10] Game features some component of randomness

[10] The game calculates and displays some ‘score’ or other measure of success

[5] Subjective coolness: this rating is subjective (based on a comparison with other projects)

(40 pts) **Style**

[10] Code is clearly written and follows standard conventions (variable names, indentation, etc.)

[10] Good use of classes and inheritance; classes are coherent and cohesive

[10] There is no unnecessary repetition of code

[10] The code is meaningfully commented

IMPORTANT NOTES:

- **Submissions that are identical to previous semester submissions will receive a zero**
- **Submissions with only one class will receive a 50% deduction**
- **Submissions that are a rendition of cookie clicker will receive a zero**

- **Submissions that do not compile will receive a zero**
- **Submissions with improperly cited AI will receive a zero and an academic integrity violation**
- **Submissions that are partially or fully copied from another submission will receive a zero and an academic integrity violation**