CSC 214 PROJECT 1

DUE TUESDAY MARCH 7TH AT 11:59PM

The goal of the projects in the course is to provide students with an opportunity to combine many different aspects and features of Android into a single, large scale application. Projects are much larger than assignments, which is why students are given at least two full weeks to complete the project, and no assignments are due the same week that the project is due.

As with assignments, you may lose 10% of your grade for style, and another 10% for poor or missing documentation. You may gain up to 20% for doing something particularly clever or "above and beyond." In order to be eligible for this additional credit you **must** explain your extra efforts in your README file. Unlike assignments you do not get partial credit for submitting anything for a project: if your app does not compile and run you will not get credit.

The goal of Project 1 is to implement an application that allows the user to choose one of at least three simple games to play. Each game should allow two human players to compete against each other, and should be fully functional. The application must use a custom launcher icon of your own design (although stealing an image from the internet is acceptable). It is up to the programmer to choose appropriate layouts and widgets to implement the requirements below.

1. The Main Activity

The main activity must include at least the following features:

a. A mechanism for both players to specify their names.
b. A display of the current, cumulative score. This score must represent points won by both players across multiple rounds of any combination of games. It should obviously start at 0 for both players.
c. The players must be presented with a set of widgets used to launch games.

2. Hotter/Colder

The first required game is Hotter/Colder. The following requirements must be met:

a. The player names and the current cumulative score should be displayed somewhere on the screen while Hotter/Colder is active.
b. For each round a random number will be generated for two players to guess. The players must be presented with a message indicating the range of possible values (e.g. "Guess a number between one and a million."). I suggest using a small number for testing (e.g. 0-10).
c. Play begins when the first player enters a guess. The feedback the player gets must follow these rules:
    i. Correct! - If they guess correctly.
    ii. Cold/Colder/Freezing/Absolute Zero! - if their guess is very far off and/or getting farther away.

        iii. Warm/Warmer/Hot/On Fire! - if their guess is pretty close and/or getting closer.

d. When the first player is finished, the second player begins their turn and tries to guess the same random number using fewer guesses.
e. The number of guesses for both players should be displayed as they play.
f. The player that guesses the number with the fewest guesses wins a number of points equal to 100 x the difference in the scores. For example if the first player needs 10 guesses and the second player only needs 5, the second player wins 500 points.

3. Hangman

The second required game is Hangman. You may hardcode a small list (e.g. 10) of words for players to guess. It is not recommended that you use a full dictionary as more often than not players won't know the unusual words that are often randomly selected. The following requirements must be met.

a. The player names and the current cumulative score should be displayed somewhere on the screen while Hangman is active.
b. Players are allowed to guess until they "miss" 8 guesses. A "missed" guess is when the player guesses a letter that does not appear in the mystery word.
c. Each time the player guesses a character, that character should be revealed wherever it appears in the mystery word. For example, if the word is "ever" and the player guesses "e", they should see that "e" appears in two places: "e_e_"
d. If the player guesses a character that has already been guessed, this should not count as a correct guess or a miss. It may be a good idea to display the message to them, though.
e. One player controls the device at a time, taking turns until they either guess the word or miss 8 times. They then pass the device to the second player, who tries to guess the same word. Obviously, the status of the display should be reset before the device is passed to the second player so that they can't see the first player's guesses.
f. The winner is the player that guesses the word with the fewest missed guesses. If both players fail to guess the word, it is a draw (no score).
g. Players should be able to quit at any time and return to the main activity. They should be warned before quitting, and asked to use the quit widget again to confirm, e.g. click the "quit" button, show a Toast, click the button again to confirm.
h. Any messages should be displayed to the players using Toasts. Error messages (e.g. invalid move) should use a short duration, game status messages (like win/loss/draw reporting) should use a longer duration.
i. If you choose to "draw" the hangman (perhaps for extra credit), I suggest using one image to represent each stage of the game (rather than trying to "assemble" a hanged man from pieces). It is easier to swap a single image with another in an `ImageView` than to try to piece together several images.

4. Tetris

The third required game is Tetris. This requires fully animated pieces that drop from the top of the screen and I am just kidding.

5. Connect 4

The third required game is Connect 4. The following requirements must be met.

a. The player names and the current cumulative score should be displayed somewhere on the screen while Hangman is active.
b. Game pieces should use custom icons that are **not** the same as the pieces used for the Tic-Tac-Toe game.
c. The game grid must be at least 8x8.
d. Players alternate similar to Tic-Tac-Toe, except that they tap the column that they'd like to move in. Their piece should be "dropped" into the column that the player taps on, "landing" in the lowest empty space in the column. **Animation is not required**. It is sufficient that the piece simply appear in the right spot in the column. Rather than making every spot a button, I recommend placing a single button at the top of each column that the user presses to "drop" a checker into that column.
e. If a player taps on a column that is already full (i.e. no room to play), this is considered an invalid move. A toast should be displayed indicating the error, and their turn should continue until they make a valid move.
f. Play continues until one player manages to play 4 pieces in a row (horizontally, diagonally, or vertically) or the game ends in a draw (no available moves, and no "connect 4").
g. Players should be able to quit at any time and return to the main activity. They should be warned before quitting, and asked to use the quit widget again to confirm, e.g. click the "quit" button, show a Toast, click the button again to confirm.
h. Any messages should be displayed to the players using Toasts. Error messages (e.g. invalid move) should use a short duration, game status messages (like win/loss/draw reporting) should use a longer duration.

Additional Enhancements (Extra Credit Opportunities)

You may earn extra credit for any enhancements above and beyond the most basic implementation of the games above or by adding an additional game. You may receive up to 20% extra credit in total.

Grading Criteria

You will be graded according to the following criteria.

Class Definitions (20%)

Note that there is no partial credit for submitting a broken application as there is with the weekly assignments. You are expected to submit a functioning app that does not crash on startup even if you do not implement all of the required games.

You are expected to practice the Model View Controller (MVC) design pattern. This means that any classes related to application logic (e.g. the game rules) should be implemented in model classes and NOT in widgets (the view) or the controllers (the activities). Each game should define its own model. As much as possible, the controller should receive events from the view and translate these into commands (method calls) on the model. The controller should then respond to changes in the model by updating the view.

Main Activity (10%)

Your main activity is simple, but should be able to launch at least 3 additional game activities, and respond appropriately when the game activities finish executing.

Game Implementations (70% total)

- Hotter/Colder (20%)
- Hangman (20%)
- Connect-4 (30%)

Code Style (Lose up to 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

Documentation (Lose up to 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. The typed version of the academic honesty pledge: “I affirm that I did not give or receive any unauthorized help on this project, and that all work will is my own.”
3. 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 2 weeks to complete the project, and there will not be an assignment 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.

If you wait until the weekend that the project is due before starting, it is **very likely** that you will not finish, and/or will be tempted to cheat by finding solutions online or with your fellow classmates. **Do not procrastinate**. Students that stop by my office hours with questions, or who email me throughout the two week interval, are much more likely to find flexibility as far as the grading is concerned. Students that email me for the first time on the weekend of March 5th may be more disappointed.

You will be expected to submit:

- Your Android Studio project complete with all resources and source files. Name your file with your last name, as usual, e.g. “stjacques_project01.zip”.
- 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 run your code in Android studio on a virtual device. If your program produces only partial functionality, you will receive a grade for the features that you did complete. Each TA will also conduct a code review to evaluate the correctness of your code and the code design and to determine whether or not it was likely stolen.