Consolidation Projects

Overview

You will submit **ONE** Consolidation Project, due **November 22**. This project is:

- an opportunity to score points on Patterns and correct "bad habit" Antipatterns
- selected from one of the Topics given below
- open-ended in terms of how you approach it and what features you want to add

Submission Requirements

In order for your project to even be accepted for grading, it must satisfy the following:

1. Your code should work if run using the `python` command in a terminal.
2. Your project must be submitted as a Git repository.
    - This can be a Git repo (folder) that you zip up as a `.zip` file and upload to ELMS,
    - **or** it can be a remote repo (e.g., on GitHub), and you can just supply the repo URL/link.
    - See video tutorials showing both of these options for demonstrations of how to do this.
3. Your project must have **sufficient** commits (using git).
    - **Three commits** at the very minimum.
    - The commits should correspond with **meaningful**, **incremental progress**. If you go from having very little code to a complete code solution, that is not good.
    - The **commit messages** should make sense and should reflect the progress in each commit.
    - If the commits show an extremely suspicious history (for example, "too clean"), I may ask you to explain your code and your process before we grade your project.
4. Your project must include a `README` **document**, either `README.md`, `README.txt`.
5. The `README` should be included in the top-level folder of the repo you submit.
6. Your `README` must explain what the program does and how to run/use it. Think of it as the "user manual" to your code.
    - If your `README` does not explain how to run your code, your project will not be graded.
    - If you inaccurately describe one or more features of your project, you may not earn points for those features.
    - Make sure you update your `README` to accurately reflect the final version of your program!

Grading

- Positive points will be scored based on the **Patterns** you include.
- **Antipatterns** will be applied as negative points.
- The entire set of Patterns (and the corresponding points) can be found on ELMS.
- A sample of possible Antipatterns is also on ELMS, but this is not necessarily exhaustive.
- You will have an opportunity to **erase** any negative Antipattern points by changing the specific "bad habit" behavior in your Final Project.

Topics

Your project should be an honest attempt to implement **ONE** of the following games.

- You don't necessarily need to get every rule of the game 100% correct.
- But you cannot choose to simply implement a different game. The game you implement should match as closely as possible with the rules of the game, described below.
- You **do** need to test your program and update your `README` to accurately describe what your program **actually** does.
- For example, if you say in your `README` that your program saves scores to a file, but that feature does not actually work as described, you will not be able to get points for the corresponding code.
- For each game, I give some ideas for some optional features that you might consider including, which may give you ways to apply different sets of Patterns.
- Remember, you score points not by the number of features, but by demonstrating **Patterns** that help you implement features.

“Tuple Out” Dice Game

For this project, you will implement a simulation of a simple dice game with the following rules:

- The object of the game is to score the most points, or to be the first to reach a certain score.
- Players take turns rolling dice to score points, as described below.
- Each turn, the active player rolls three dice:
  - If all three dice are rolled with the same number, the player has “tupled out”, and ends their turn with zero points. (For example, rolling three “4”s at the same time.)
  - If **two** dice have the same value, they are “fixed”, and they cannot be re-rolled.
  - During their turn, the player can re-roll any dice that are not “fixed”, as often as they would like, until they decide to stop, or until they “tuple out” (get three of the same number).
- When a player decides to stop, they score points equal to the total of the three dice, and then their turn ends.
- If a player “tuples out”, their turn ends and they score 0 points for that turn.

Examples of play

1. A player starts out by rolling a 1, 2, and 2 on three dice.
   - The 2’s are **fixed**, because there’s two of them, so they can’t re-roll those.
   - But they want to reroll the 1, since it’s so low.
   - They re-roll it to a 3, but they want to re-roll it again, to try to get higher.
   - When they re-roll the second time, it turns up as a 2, and since the other two dice are also 2’s, they have “tupled out”. They get zero points for the round.
2. A player starts by rolling a 1, 2, and 4.
   - They decide to roll all three again, and they get a 1 and two 5’s. This means the two 5’s are **fixed**, and they could choose to re-roll the 1.
   - However, they decide to stop here, because the 5’s are high, and they don’t want to risk “tupling out” by re-rolling the 1 and getting a third 5.
   - So they stop their turn with 11 points (5 + 5 + 1).

Ideas for additional options and features of the “Tuple Out” Dice Game

- How many players do you want to include? If it’s one player, are they playing to beat a “high score” that is recorded across games, or do they play against a “computer player”?
- When is the game over? I recommend either playing until one player reaches a score of 50, or playing for five total turns. But you could play around with some different options.
- How and when will you display the running scores, so that players know what the current scores are?
- Can the game record the scores for each game, including who won?
- Can the game record something like “high score” records over many games, or a running tally of how many games a particular player has won?
- If you implement a “computer player”, could you implement different strategies?
- Would the game be better or more interesting if the dice were changed, including the number of dice or the number of values on each die?
- What about adding rules for additional special scoring?

“Decktionary Battle”

This project is an implementation of a simple 2-player card game.

- This game uses a standard deck of playing cards, with the Kings removed
  - This results in a deck with 48 cards.
- The game starts with the deck face down (shuffled), and eight cards are dealt to each player.
- The players keep their cards “in hand” (meaning they can see them), and private (meaning the other player can’t see them).
- One player starts by playing a card. This is called “leading”.
- Whatever suit (hearts, diamonds, clubs, or spades) the player leads with, the other player must follow, if possible.
- If the second player cannot play a card in the same suit, they can play any card they wish.
- The highest-value card that is in the “lead” suit wins that round, and that player earns a point.
- The player who wins the point gets to lead in the next round.
- After every round, one of the cards in the deck is removed and shown to both players. This has no effect on scoring or points, other than giving players information about what cards the other player might have.
- After the players have played all eight of their cards, they are each dealt eight more cards.
- **Ending the game**:
  - If the game ends 16-0, the player with zero points has “shot the moon”, and immediately scores 17 points, making them the winner.
  - The player with the most points at the end of the game wins.
  - If one player has 9 or more points in the middle of the game, and the other player has at least 1 point, the game can be ended early instead of playing through the entire deck, since there is no way the other player can still win.

Examples of play

1. Player 1 leads by playing a 5 of Hearts.
   - Player 2 has only one card with Hearts, a 3 of Hearts.
   - Player 2 **must** play the 3 of Hearts.
   - Player 1 wins the point, since a 5 beats a 3.
   - The players turn a card over from the deck to reveal a 7 of Hearts. Player 2 wishes they had had that card! Oh well.
   - Player 1 leads the next round, and play continues.
2. Player 1 leads by playing a 5 of Hearts.
   - Player 2 has a 9 of Hearts and a 3 of Hearts.
   - They decide to play the 9, which beats Player 1’s 5 of Hearts.
   - The players turn a card over from the deck, revealing a Queen of Spades. Interesting, now we know that no one will get that Queen!
   - Player 2 wins the point, and leads the next round.
3. Player 1 leads by playing a 5 of Hearts.
   - Player 2 does not have **any** Hearts.

- Player 2 decides to play a 9 of Clubs.
- Player 1 wins the point, because even though 9 is higher than 5, the 9 did not follow suit, so it cannot win the point.
- The players turn over a card from the deck, revealing a 2 of Clubs. No one cares.
- Player 1 leads the next round.

Ideas for additional options and features of the "Decktionary Battle" game

- Is this game one player "against the computer", or is it meant to be played by two human players?
  - If one player, how will you implement the computer's choices in the game?
  - If two players, how will you handle keeping the players' hands hidden to each other?
- What about options to play longer or shorter games?
- Are the results of games saved? How?
- Any rules variations you'd want to try out?