# Consolidation Projects

## Overview summary

The Consolidation Projects are:

- opportunities to score points on Fundamentals
- selected from one of the Topics given below
- open-ended in terms of how you approach them and what features you want to add

## Requirements

In order for your project to be graded, it must satisfy the following requirements:

- Your project must be submitted as a Git repository.
  - This can be a Git repo 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.
- Your project must have **sufficient** commits.
  - Three commits at the very minimum.
  - The commits should make sense and should correspond with meaningful progress.
  - 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.
- Your project must include a `README` document, either `README.md`, `README.txt`, or simply `README` (no file extension).
- 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 your `README` does not explain what your code can and cannot do, your project will not be graded.
  - Make sure you update your `README` to accurately reflect your submitted program!

## Grading

- You will **score points** on your grade if you complete Fundamental Items.
- The list of Fundamental Items will be posted elsewhere.
- You can only score points on an Item if:
  - The code that satisfies the item is important to your program (no extraneous code).
  - The code that satisfies the item accomplishes what you mean it to (in other words, it matches the `README` description).
- For example, one Fundamental Item is "correctly imported and used Standard Library module." Just a simple `import` statement by itself is not enough to score that point; you need to actually use something from that `import` in your program.

## 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 **do** need to test your program and update your `README` to accurately describe what your program does.
- For example, if you say in your `README` that your program saves scores to a file, but that feature does not actually work, or doesn't work like you describe it, you will not be able to get points for the corresponding code.

## Word Guessing Game

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

- This game may have one or more players.
- The object of the game is to guess a secret word from a bank of words.
- Players take turns.
- On each turn, the player gets to guess a possible letter that might be in the word.
- After guessing a letter, the program tells the player how many occurrences of that letter are in the secret word.
- The game should NOT tell the player the **position** of letters in the word.
- Optionally on that same turn, the player can also try to guess the word.
- The program tracks how many letter guesses and how many word guesses each player makes.
- A player's final score is the number of turns (letter guesses) they made before guessing the correct word. Lower scores are better.
- Each player only has three word guesses, and they lose the game if they get their third guess wrong.

### Ideas for additional options and features of the Word Guessing Game

- How big is the bank of words? Could the bank of words have a theme or a hint of some kind?
- Will the game tell the player how long the word is? Or are all words in the game a certain length? Or is word length completely unknown to the player?
- Will the game display a list of all the letters guessed so far, or will it force the player to try to remember?
- Can the game record the scores by writing results to a list?
- Does the game "remember" and avoid any words it used before with a player (effectively changing the word bank each game)?

## “Tuple Out” Dice Game

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

- This game may have one or more players.
- 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.
  - 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.

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

- 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?
- Can you implement an “AI” player strategy that a player could play against?
- 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?

## Card Memory Game

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

- This game may have one or more players.
- The object of the game is to score the most points by recalling the most information each turn.
- The basic version of this game uses a standard deck of cards.
  - 52 cards, 13 cards in each of four suits (hearts, diamonds, clubs, and spades).
  - The value of a card is called the “rank”, and it represents a number from 2 to 10, plus an Ace and the three “face” cards (King, Queen, Jack).
- Each turn, a player is dealt cards one at a time, displaying each card for a certain amount of time.
- After the cards have been dealt, the player is told: Suits or Ranks
  - The choice of Suits or Ranks is random, and is not known to the player until after the cards are dealt.
  - If “Suits”, the player must list the suits (heart, club, diamond, or spade) of the cards that were dealt.
  - If “Ranks”, the player must list the ranks (number or king/queen/jack/ace) of the cards that were dealt.
- Points are scored for each correct value.
- The game may continue to a certain point value (e.g., first to 20 wins) or across a certain number of turns (e.g., high score after 5 turns).

### Ideas for additional options and features of the Card Memory Game

- There are LOTS of parameters to play around with, including:
  - How many cards are dealt per turn? Is it constant or variable?
  - How long is each card shown before going to the next one?
  - Does the recall have to be in the same order as the cards were dealt?
  - Are “Suits” and “Ranks” responses worth the same number of points?
- Does the program save scores or performance?
- How are cards represented and/or displayed?
- What about different kinds of card decks?