

# Final Project Report

- Class: DS 5100
- Student Name:
- Student Net ID:
- This URL: a URL to the notebook source of this document

## Instructions

Follow the instructions in the Final Project instructions and put your work in this notebook.

Total points for each subsection under **Deliverables** and **Scenarios** are given in parentheses.

Breakdowns of points within subsections are specified within subsection instructions as bulleted lists.

This project is worth **50 points**.

## Deliverables

### The Monte Carlo Module (10)

- URL included, appropriately named (1).
- Includes all three specified classes (3).
- Includes at least all 12 specified methods (6; .5 each).

Put the URL to your GitHub repo here.

Repo URL:

Paste a copy of your module here.

NOTE: Paste as text, not as code. Use triple backticks to wrap your code blocks.

*# A code block with your classes.*

### Unitest Module (2)

Paste a copy of your test module below.

NOTE: Paste as text, not as code. Use triple backticks to wrap your code blocks.

- All methods have at least one test method (1).
- Each method employs one of Unittest's Assert methods (1).

*# A code block with your test code.*

### Unittest Results (3)

Put a copy of the results of running your tests from the command line here.

Again, paste as text using triple backticks.

- All 12 specified methods return OK (3; .25 each).

### Import (1)

Import your module here. This import should refer to the code in your package directory.

- Module successfully imported (1).

```
# e.g. import montecarlo.montecarlo
```

### Help Docs (4)

Show your docstring documentation by applying `help()` to your imported module.

- All methods have a docstring (3; .25 each).
- All classes have a docstring (1; .33 each).

```
# help(montecarlo)
```

### README.md File (3)

Provide link to the README.md file of your project's repo.

- Metadata section or info present (1).
- Synopsis section showing how each class is called (1). (All must be included.)
- API section listing all classes and methods (1). (All must be included.)

URL:

### Successful installation (2)

Put a screenshot or paste a copy of a terminal session where you successfully install your module with pip.

If pasting text, use a preformatted text block to show the results.

- Installed with pip (1).
- Successfully installed message appears (1).

## Scenarios

Use code blocks to perform the tasks for each scenario.

Be sure the outputs are visible before submitting.

### Scenario 1: A 2-headed Coin (9)

Task 1. Create a fair coin (with faces  $H$  and  $T$ ) and one unfair coin in which one of the faces has a weight of 5 and the others 1.

- Fair coin created (1).
- Unfair coin created with weight as specified (1).

Task 2. Play a game of 1000 flips with two fair dice.

- Play method called correctly and without error (1).

Task 3. Play another game (using a new Game object) of 1000 flips, this time using two unfair dice and one fair die. For the second unfair die, you can use the same die object twice in the list of dice you pass to the Game object.

- New game object created (1).
- Play method called correctly and without error (1).

Task 4. For each game, use an Analyzer object to determine the raw frequency of jackpots — i.e. getting either all  $H$ s or all  $T$ s.

- Analyzer objects instantiated for both games (1).
- Raw frequencies reported for both (1).

Task 5. For each analyzer, compute relative frequency as the number of jackpots over the total number of rolls.

- Both relative frequencies computed (1).

Task 6. Show your results, comparing the two relative frequencies, in a simple bar chart.

- Bar chart plotted and correct (1).

## Scenario 2: A 6-sided Die (9)

Task 1. Create three dice, each with six sides having the faces 1 through 6.

- Three die objects created (1).

Task 2. Convert one of the dice to an unfair one by weighting the face 6 five times more than the other weights (i.e. it has weight of 5 and the others a weight of 1 each).

- Unfair die created with proper call to weight change method (1).

Task 3. Convert another of the dice to be unfair by weighting the face 1 five times more than the others.

- Unfair die created with proper call to weight change method (1).

Task 4. Play a game of 10000 rolls with 5 fair dice.

- Game class properly instantiated (1).
- Play method called properly (1).

Task 5. Play another game of 10000 rolls, this time with 2 unfair dice, one as defined in steps #2 and #3 respectively, and 3 fair dice.

- Game class properly instantiated (1).
- Play method called properly (1).

Task 6. For each game, use an Analyzer object to determine the relative frequency of jackpots and show your results, comparing the two relative frequencies, in a simple bar chart.

- Jackpot methods called (1).
- Graph produced (1).

## Scenario 3: Letters of the Alphabet (7)

Task 1. Create a "die" of letters from *A* to *Z* with weights based on their frequency of usage as found in the data file `english_letters.txt`. Use the frequencies (i.e. raw counts) as weights.

- Die correctly instantiated with source file data (1).
- Weights properly applied using weight setting method (1).

Task 2. Play a game involving 4 of these dice with 1000 rolls.

- Game play method properly called (1).

Task 3. Determine how many permutations in your results are actual English words, based on the vocabulary found in `scrabble_words.txt`.

- Use permutation method (1).
- Get count as difference between permutations and vocabulary (1).

Task 4. Repeat steps #2 and #3, this time with 5 dice. How many actual words does this produce? Which produces more?

- Successfully repeats steps (1).
- Identifies parameter with most found words (1).

## Submission

When finished completing the above tasks, save this file to your local repo (and within your project), and then push it to your GitHub repo.

Then convert this file to a PDF and submit it to GradeScope according to the assignment instructions in Canvas.