Discrete Structures: Counting Worlds Capstone

Chapter 5 Project – Design Your Own Universe

Overview

In Chapter 5 you are asked to *design a small universe* (game, system, or scenario) and show that you can *count* what lives inside it.

Your universe could be:

- a small card game or draft system,
- a character builder with races, classes, and loadouts,
- a squad / team builder with roles and items,
- a resource or skill point distribution system,
- or any other world where you make structured choices.

The core idea: you tell a story, then you expose the combinatorics behind it.

Pair programming: You will work in pairs. Both partners are responsible for understanding the math and the code, and both names go on all deliverables.

Project Requirements (Mathematics)

Your universe must include at least:

1. **One permutation situation** (order matters).
   Example: ordering players in a queue, arranging cards, turn order, seat order.
2. **One combination situation** (order does not matter).
   Example: choosing a team of heroes, selecting a hand of cards, picking a loadout.
3. **One stars-and-bars style distribution**.
   Example: distributing points among stats, skill points among abilities, resources among locations, scoops among flavors.
4. **At least one probability question** that you compute from counts.
   Example: probability of drawing a certain type of hand, probability that a random build satisfies a constraint, etc.

For each of these, you should:

- State the problem clearly in words.
- Translate it into symbols (for example, "choose k from n", or "number of integer solutions to $x_1 + \ldots + x_k = n$").
- Name the relevant tool (product rule, sum rule, permutation, combination, stars-and-bars).
- Show the formula and at least one worked example with numbers.

Project Requirements (Python)

Write at least one short Python script (more is fine) that interacts with your universe.

Minimum expectations:

- Use formulas (`math.factorial`, `math.comb`, or your own functions) to compute at least one important count in your universe.
- For a small version of your universe:
  - either *enumerate* all possibilities (for example, all teams of size 3),
  - or run a *Monte Carlo simulation* to estimate a probability.
- Compare the Python result to your theoretical count or probability and comment on whether they agree (and why they might differ for small simulations).

Optional AI / ML extension (extra credit / enrichment):

- Define a simple score for builds, teams, or states in your universe.
- Use search or sampling to find "good" builds (for example, random search, hill-climbing, or simple Monte Carlo rollouts).
- If you are curious, you may treat builds as feature vectors and try a small model that predicts which builds will be strong, but this is not required.

Deliverables Checklist

Each pair submits the following (one submission per pair):

☐ **Universe description** (1–2 pages). Story, rules, what choices players or users make, and any constraints.

☐ **Math writeup** (1–2 pages). For each required situation (permutation, combination, stars-and-bars, probability): clearly labeled problems, formulas, and worked examples.

☐ **Python code**. At least one script with comments, using formulas and either enumeration or simulation. Include a short text summary of what you learned from running the code.

☐ **Reflection paragraph** (about half a page). Answer prompts like:

- What surprised you about the size of your universe?
- Where did the counting get messy, and how did you handle that?
- How might someone use counting or simple AI tools to balance or explore your universe?

Pair Programming Guidelines

During your work sessions, you should intentionally practice pair programming.

Driver Has hands on the keyboard. Types the code, edits the LaTeX, and narrates what they are doing.

Navigator Watches for bugs, asks "why" questions, and thinks ahead:

- Does this match the math model?
- Are we naming variables clearly?
- Are we testing the interesting cases?

Good practice:

- Switch roles regularly (every 10–15 minutes, or at natural breakpoints).
- Both partners must be able to explain every part of the code and the math.
- If you disagree, pause and explain your reasoning in words before changing code.

I will be looking for evidence that you both contributed and that you can both talk about the universe, the counting, and the code.

Grading Rubric (Guide)

This project will be graded using roughly the following criteria (example point weights in parentheses, adjust as needed):

Criterion	**Strong Performance**	**Needs Improvement**
Universe design (0–8)	Universe is clear, coherent, and interesting. Rules are well specified and easy to imagine playing.	Universe is vague, inconsistent, or very small; rules are hard to follow or incomplete.
Counting correctness (0–10)	Permutation, combination, and stars-and-bars problems are clearly stated; correct formulas and computations with explanations.	Frequent mistakes in identifying or applying formulas; explanations missing or unclear.
Use of multiple tools (0–6)	All required structures (permutation, combination, stars-and-bars, probability) appear naturally in the universe and are well motivated.	One or more required structures missing, forced, or not clearly connected to the story.
Python component (0–8)	Code is clean, commented, runs successfully, and clearly connects to the math; comparison between theory and experiment is explained.	Code is incomplete, hard to read, does not run, or has a weak connection to the math; little or no analysis of results.
Communication and reflection (0–8)	Writing is organized and readable. Reflection shows genuine insight into counting, complexity, or possible AI uses. Both partners can explain the work.	Writing is disorganized or very brief. Reflection is superficial. It is unclear whether both partners understand the project.

Total suggested: 40 points. I may also award small bonus credit for creative, well-executed AI or simulation extensions.