# Project & Scene Setup

1. Create a new Unity scene named StackLab.
2. Change Edit → Project Settings → Default Solver Iterations to 10
3. Add a Main Camera and position so the bottom area (ground) is always visible.
4. Add a ground GameObject with a box collider. Create and assign a new tag named 'Ground'.

## UI

1. Add a Canvas (Screen Space - Overlay).
2. Add a Button named SpawnButton and label it 'Drop Cube'.
3. Add a TMP CounterText showing '0'.
4. Add a TMP AIMessageText initially blank and hidden.

## Prefabs

1. Make a CubePrefab with:

* MeshRenderer
* BoxCollider
* Rigidbody (mass 1)

## Game Logic Implementation

1. Create the Pool System:
2. In your Scripts folder, add a new C# script called Pool.cs.
3. Implement a generic object pool class to efficiently reuse cube and dot objects. It should:

* Instantiate a set number of objects from a prefab.
* Deactivate objects when not in use and reactivate them when needed.
* Provide Get() and Release() methods for retrieving and returning objects.

1. Create the StackCube Script:
2. Add a new script called StackCube.cs.
3. This script should contain an integer id field to identify each cube.
4. Attach this script to your CubePrefab.
5. Create the Candidate Data Script:
6. Add a new script called Candidate.cs.
7. In this file, define two structs (not classes):
8. Candidate: Holds Vector3 position and Quaternion rotation for cube placement.
9. CandidateScore: Stores results of a simulation, including score, stability, and maxDisplacement.
10. Implement the StackManager:
11. Make a script called StackManager.cs and attach it to an empty GameObject
12. In StackManager, reference the following in the Inspector:
    1. CubePrefab (with StackCube script attached)
    2. DotPrefab (optional, for visualizing spawn spots)
    3. UI elements: SpawnButton, CounterText, AIMessageText
    4. Main Camera
13. Define public settings for pool sizes, candidate sampling, and camera offsets.
14. In Awake(): Initialize pools, setup UI, assign camera, etc.
15. Implement SpawnButton logic:
    1. *Generate/simulate candidates\**,
    2. drop cube,
    3. increment counter,
    4. visualize with dots.
16. Track active cubes in a list.
17. Compute tallest stack: Group cubes, find tallest stack, update UI.
18. Adjust camera smoothly to keep stack visible.

14. Create the simulation:

1. Make a Separate Physics World programmatically
2. Copy the Real Stack into the Simulation.

* For each cube in the main scene, take a cube from the simulation pool and copy its transform and angular velocity

1. Create a list of potential candidate structs. (e.g., -10°, -5°, 0°, 5°, 10°).
2. Loop through each candidate and run a separate physics simulation for it.
   1. Reset the simulated stack to its original state from Step 2.
   2. Add a new simulated cube using the candidate's position and rotation.
   3. Manually step the simulation forward for about 1.5 seconds by repeatedly calling simPhysics.Simulate(timeStep).
   4. Measure the final stability using displacement and tilt
   5. Calculate a score based on this stability. (score = 100 - (displacement \* weight) - (tilt \* weight).

5. Pick the Best Candidate.

6.Place the best Cube in the Real Game.