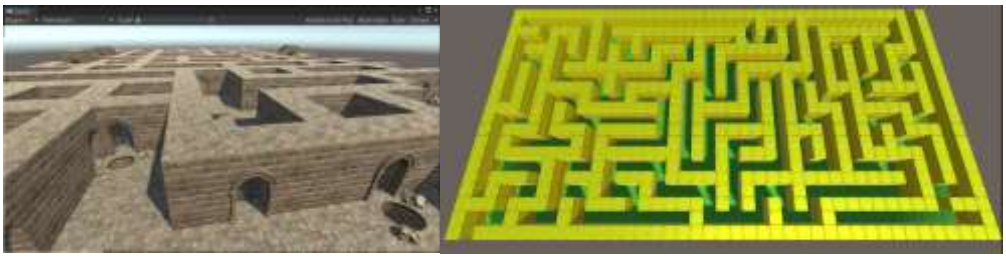
Lab #2

***CSE3541/5541* (*Autumn 2024*)**

Due Date: Tuesday, September 24 by 11:59pm

Guidelines

- The grader will NOT overcome compiler or debugging errors for you.
- All make-ups for lab must be accompanied by a documented and verifiable excuse well before the deadline. Given the severity of the emergency please inform me as soon as possible.
- Lab submissions will NOT be accepted via email to me or the grader.
- Work on the lab on your own. **No group work**!



Objectives

The purpose of this lab is to learn more about the use of geometry, transformations, and hierarchies (and parenting) in the context of maze generation.

Collaboration and Piazza

Please read the course policy on Academic Misconduct in the syllabus. You may discuss the lab with your classmates ONLY at a high level. You must formulate your own solution (all code must be authored by you alone) without any help from others or third party sources (e.g., the internet).

Do not post any part of your solution or spoilers on Piazza.

In your lab report (see below), you need to list with whom you have discussed the lab.

Preliminaries

You have been provided with three Unity packages on Carmen: CSE3541Lab2RecursiveBacktracking, CSE3541Lab2Template, and ParentingExample. The Unity package CSE3541Lab2RecursiveBacktracking contains a solution that generates a maze using the Recursive Backtracking algorithm. You will use the Unity package CSE3541Lab2Template to insert your solution to use a different maze generation algorithm. The Unity package ParentingExample is just an example to possibly help you with other parts of the lab.

Task I (Git and Understanding the code base)

Task I-A: Use a **local** Git repository (not Github) running only on your computer to develop your solution. Remember that Git is the NOT the same as Github (do NOT use Github). Write your solution INCREMENTALLY and use Git to maintain different versions up to your final solution for submission.

TASKI-B: Run the code in the Unity package **CSE3541Lab2RecursiveBacktracking**. You can change the grid dimensions by clicking on the 'LevelGeneration' GameObject in the Hiearchy tab in the Unity Editor and changing the serialized fields 'Number of Rows' and 'Number of Columns'. Don't make these values too high because the algorithm uses recursion and will lead to stack overflow.

Study and understand the scripts in this Unity package. The scripts that you will be working with for the maze generation portion are 'Maze.cs', 'GridTraversal.cs', 'IGridGraph.cs', and 'GridGraph.cs'. The start of program execution is in the Awake() method in the script 'LevelGeneration.cs'. The method 'DepthFirst' returns edges (dual graph) that indicate pathways in the maze. Each edge corresponds to no tile at a grid location (row, column).

Task II (Implement a maze generation algorithm)

Choose one of the following maze generation algorithms: **Prim's algorithm** or **Kruskal's algorithm**. You will implement one of these two algorithms in the Unity package **CSE3541Lab2Template** (the same code as in **CSE3541Lab2RecursiveBacktracking** but without the recursive backtracking code), which you submit as your final solution. Do not delete the code provided in the package. Doing so will result in no credit for the lab. You will insert your solution in places indicated by comments that appear in the template:

Script 'Maze.cs': Finish the line of code under the comment 'Replace ??? with your code'. You can either have the variable **grid** initialized to an object that is an instance of class GridGraphRandomizedNeighborsDecorator or an instance of class GridGraph.

Script 'GridTraversal.cs': In the class GridTraversal, insert your instance variables and your implementation of either Prim's or Kruskal's algorithm into the method 'GenerateMaze'.

DO NOT change the method signature of the method 'GenerateMaze'. Doing so will result in no credit for the lab. Make any helper methods private.

Scripts 'IGridGraph.cs' and 'GridGraph.cs': This class hierarchy contains the method called 'Neighbors', which returns an IEnumerable type. The method returns the neighbors of a cell in the 2D grid of cells. If this method is not appropriate for your solution, then define your own method with a different name in both of these scripts (see the provided comments). This step is optional if you will use the provided method called 'Neighbors'.

Task III (Scene rotation)

Using the **New Input System** (as you did in Lab #1), Implement the following feature: pressing 0 (zero) starts continuous rotation of everything in the scene about the y-axis. Pressing 0 (zero) again stops this rotation. For transformations, look up the Unity documentation on the following methods: transform.Translate(), transform.Rotate(), and transform.GetChild(). Look into using parenting to define the scene hierarchically. Refer to the provided Unity package ParentingExample.

Important notes: You **MUST** use the New Input System. You must rotate the maze geometry and NOT just rotate the camera around the maze.

Task IV (Different prefabs)

Use different prefabs for the floors and walls. Some free assets with some nice blocks.

> https://assetstore.unity.com/packages/3d/environments/fantasy/flat-cube-environment-195664
>
> https://assetstore.unity.com/packages/3d/environments/forest-low-poly-toon-battle-arena-tower-defense-pack-100080
>
> https://assetstore.unity.com/packages/3d/props/polygon-starter-pack-low-poly-3d-art-by-synty-156819
>
> https://assetstore.unity.com/packages/3d/environments/cube-world-brick-blocks-proto-series-153993
>
> https://assetstore.unity.com/packages/3d/environments/dungeons/dungeon-low-poly-toon-battle-arena-tower-defense-pack-109791

In addition, you can also import a package or two to use as floors/platforms, walls, lava, etc. to create a nice looking game level.

Task V (Game level)

Generate a maze, i.e. a game level, and move the camera to get several good images, or have several cameras and switch between them using input (I am not asking you to implement camera switching as you did in Lab #1 here). Include the best images in your lab report.

Task VI (Lab report)

Write a detailed report that **states what you have implemented**. This is a report, so please do not just provide short answers.

- **Include a short introduction that introduces the reader to the problem (as you would in any engineering lab report)**
- **Include a few screen shots of your final produc**t
- **Include the source code that YOU inserted for your solution (not all of the source code) Provide a high level description of your source code as well as good documentation in the source code.**

Include in your report (in addition to what you implemented) the following:

1. State with whom you have had any discussions about the Lab
2. State all of what you have implemented including which maze generation algorithm (Prim's or Kruskal's). You will state extra credit in question 5.
3. Write **pseudocode** of your algorithm (between plain English and code) that succinctly describes your implementation of the algorithm. An example of pseudocode for the Recursive Backtracking algorithm may be (also see lecture notes):

    Depth First (start row, start column)
    a. Create 2D visited array of Booleans
    b. Call Recursive function

    Recursive Function (cell)

    a. If (cell is not visited)
        Mark as visited
        For each neighbor, called n
            If n is not visited
                Call Recursive Function (n)
                Return edge
4. What is the asymptotic run time of your algorithm in the worse case, i.e. in Big-Oh notation?
5. What changes would you suggest to the design of code base, e.g. class hierarchy? Provide at least a couple of good suggestions that show you have thought about this.
6. If you did extra credit, state what you implemented and explain with some detail what you accomplished and provide images.
7. How hard or easy did you find this lab? Explain.

Extra Credit

- (5 points) Implement either Eller's algorithm (see http://www.neocomputer.org/projects/eller.html or whatever description you find) or the other maze generation algorithm in this lab (i.e., if you implemented Prim's then you can implement Kruskal's or if you implemented Kruskal's then you can implement Prim's).
- (3 points) Implement procedural clutter or object placement. For example, implement an algorithm to automatically places torches (or actual in-engine point light sources) within the maze.
- (2 points) Add various heights to your maze.
- (2 points) Create a maze that is raised above an animated lava bed or animated water.

Grading

- Maze generation algorithm. Either Prim's or Kruskal's. (50 points)
    - -10 points: Did not implement randomized versions of the algorithm
    - -20 points: Not a perfect maze
    - -10 points: Not an interesting maze, i.e. long and straight paths, not "windy", not biased
- Scene rotation (10 points)
- Used different prefabs (10 points)
- Lab report (20 points)
- Your inserted code is well written, readable, and documented (8 points)
- Submitted in the correct format and following file name convention (2 points)

Submission

In addition to your report (**submit this as a PDF file**), submit a **Unity package**.

Within Unity, right-click the scene file and choose Export Package … This ensures that only the files used in the scene are exported (as well as all scripts as it cannot tell sometimes). Save the package as YourlastnameLab2. This should result in a file called YourLastNameLab2.unitypackage being created in the folder you specified (hint, use Documents or some other space). This is the file you should turn in for grading.
Under the Lab2 assignment on Carmen, submit both the PDF file and the unitypackage file using File Upload option (not the Buckeye Box option).

You can test to determine if you have made the package correctly by creating a new Unity project. Open the package file from a file browser, import all of the files in it, open the scene within Unity, and click play. Don't forget to the install the New Input System package.