Procedural Generation Assignment #3: Grids and Cellular Automata

[Due 02/17/2020 before the start of class]

The goal of this assignment is to familiarize you with using 2D arrays or "grids" as scratch space during your generation process. This assignment also introduces you to a simple and powerful grid generation technique called "Cellular Automata".

If you're new to using 2D arrays in C#, or need a refresher, here is a quick explanation: https://www.tutorialspoint.com/csharp/csharp_multi_dimensional_arrays.htm

To start the assignment, download the template project from this address: https://github.com/badtetris/ProcGen2019Assignment3

1. Open the scene named "Task1". Modify code in Task1Generator.cs to complete these tasks.

a. **Complete the function named "fillGrid"** so that your grid spawns walls **all along the border of the grid** along with the requested number of **extra walls in random locations**. None of your walls should occupy the same location (**HINT:** you only need to change values in _wallGrid. The spawnWalls function has already been implemented to instantiate walls once _wallGrid has been filled).

b. **Complete the function named "getPlayerSpawnPos"** to provide an **empty location inside the border where the player will be spawned**.

(**NOTE:** the Task1Randomizer script randomizes the values of gridWidth, gridHeight, and numExtraWalls *before* your code runs, so make sure your code works with the values given in these variables. You can press 'R' while running the scene to reload the scene with new values.)

[2 points]

[Tasks 2 and 3 on Next Page]

2. Open the scene named "Task2". Modify code in Task2Generator.cs to complete these tasks.

a. **Complete the functions named "fillGrid" and "getPlayerSpawnPos"** so that you generate walls completely randomly and the player is spawned in an empty location (you can feel free to use the same code you wrote in task 1-b for "getPlayerSpawnPos").

b. **Complete the function named "nextCAValue"** to complete the implementation of a "Cellular Automata". The "nextCAValue" function returns a boolean where true indicates a wall should exist in the given location and false indicates that a wall shouldn't exist in the given location.

   To test your CA implementation, run the game from the Task2 scene and press space to perform a CA step (your player will respawn during each step).

   You are free to implement the "nextCAValue" function however you like, but your goal is to generate interesting levels after a few steps of CA.

For more info on Cellular Automata and hints on how to generate interesting caves, check out this tutorial (**Note**: The tutorial code is written in Java instead of C#): https://gamedevelopment.tutsplus.com/tutorials/generate-random-cave-levels-using-cellular-automata--gamedev-9664

[2 points]

3. Open the scene named "Task3". For the final two points, expand or mod the existing CA generator from Task2 in some way and save your mod to this scene. How you mod it is up to you, but your goal should be to make something neat. Here are some **suggestions**:

- Create a CA generator that generates more than just walls (for instance, water, trees, mountains, etc.). Come up with a custom CA ruleset for generating interesting maps with these tiles.
- Design some mechanics that allow the player to interact with the generated map (for instance, perhaps the player gains the ability to dig through walls). Try to create a fully featured game (this will likely involve creating some new tiles to spawn beyond just the player and the walls).
- Turn the CA steps themselves into game mechanics. For instance, can you think of a way to make a tile "move" by using CA rules to update its position?

[2 points]
[Total: 6 points]