Lab3 – Moving through your Maze

Preparation

If you haven't looked at the Lab1 solution from Dr. Crawfis, do so here:
https://osu.instructure.com/files/34932348/download?download_frd=1

Many students were placing cameras under or within the character hierarchies. No need for this if the camera has a target. Likewise, there is no need to put the movement script under or within a character hierarchy. Also saw many people using Unity Events and Vector3.forward.

Tasks:

1) Add first-person camera using the new Input System and C# events like Lab1
    a. Rotate with the mouse position or Gamepad right joystick. You should use RotateTowards and play with a maximum rotation speed (use a Scene with only a large plane). See: https://docs.unity3d.com/ScriptReference/Quaternion.RotateTowards.html.
    b. Move with the arrow, WASD or Gamepad left joystick but this time in local space, not world space. Use MoveTowards but allow the speed to be changed using an event. See: https://docs.unity3d.com/ScriptReference/Vector3.MoveTowards.html.
    c. Extra credit: Change the movement speed if the user *"presses"* the Shift key. Change it back when they *"release"* the Shift key.
2) Add box colliders to your walls and make them static.
    a. Note: If you have "blanks" for your walls (i.e., a platform maze) you will need to create invisible walls by creating a Prefab with an Empty GameObject containing a Box Collider component. Experiment with making the box colliders slightly larger than the grid cell size. This should keep the player away from the edge some.
3) Likewise, make sure your floor (floor tiles) has a static box collider.
4) Create a mapping function (Func) or class that allows you to query the Unity "maze-space" (worldspace?) and returns a cell location in the abstract Maze space. It takes a position (Vector3? Vector2?) in "maze-space"and determines which (row,column) the position corresponds to in the abstract maze. Likewise, the class (or another Func) takes a (row,column) and determines a position in the maze-space.
5) Write a MazeQuery class as shown in class. It will provide an IEnumerable<(row,column)> for various queries: dead-ends, straights, etc. Later we may extend this to a path.
6) On Play:
    a. Create a random maze
    b. Add collectables:
        i. And (spawn) coins or power-ups at each dead-end as well as wherever you choose. These should have triggers that respond to the player. At a minimum they should delete.
            1. Extra credit: Play a sound and have them float upward for a brief time (or distance) using a coroutine.
            2. Extra credit: Add a Score and floating score points

    ii. Add a spinning animation script to each coin (can be done in the Prefab).
    iii. Add a bobbing animation script to each coin (can be done in the Prefab).
c. Add a trigger at the exit of your maze
d. When the player hits the exit trigger, notify your EventManger. It should then fire an event that the game is over. A separate script should display another scene or simply some Game Over text (you can have this be inactive and then turn it to active).
e. Place your player at the center of the starting cell and assign the movement script to them.
    i. Extra credit: Create a small path that leads to the start and place your character somewhere along this path. The path can be straight.
f. Extra credit: After Game Over is displayed for a brief period, restart the level. Can be done by loading the Scene again or deleting the maze, collectables and trigger boxes.