**Lab 1 – Simple Obstacle Course (Instructor Manual)**

**Pre-Lab:**

* Check that everyone has Unity and VS Code set up properly. Work with students that encounter any issues
* The pseudo-code answer should look something like:

IF NO\_OBSTACLE

MOVE\_FORWARD

ELSE IF HIT\_OBSTACLE

IF LEFT\_SIDE\_NOT\_OPEN & RIGHT\_SIDE\_NOT\_OPEN

STOP

ELSE

IF LEFT\_SIDE\_OPEN

MOVE\_LEFT

ELSE IF RIGHT\_SIDE\_OPEN

MOVE\_RIGHT

**Lab:**

Part 1:

* Load in Part1 scene (just a blank world)
* Introduce students to the Unity interface and the basic rules
  + Main sections in the Unity workspace:
    - Scene (at the center): contains environments and objects of your program. This is where you will design what each of your “level” looks like
    - Game (tab next to scene): where you can see what your program looks like when it is running. This tab can be dragged to be viewed side-by-side with Scene
    - Hierarchy (on the left): where you can find the structure of the currently loaded scenes, including names of all objects in the scene
    - Project/Console tabs (at the bottom):
      * Project: where you can find the structure of the project. Can be used to find available assets in the project to utilize during the designing process
      * Console: where information about the program is printed (such as error/warning) when it is running. Used during the debugging process
  + Unity follows basic laws of physics e.g. gravity, friction
  + GameObject: the data type for objects in the program that can interact with the environment and/or be interacted with. For instance, the player, the ground, the wall, and any object added to the scene are all GameObject.
  + Components: properties that can be added to a GameObject to specify its nature and behavior. These can be pre-defined and customizable (e.g. Rigidbody, Material), or user-defined through C# scripts. The majority of the code we write will be these scripts that are added to specific objects in the scene to establish the desired behavior
* Show how to create the ground and player object
  + Create the cubes
  + Adjust dimensions and positions
  + Assign pre-made component for ground and player
* Put player box above ground and start play simulation
  + QUESTION: why does the box not fall as expected?
  + Answer: the box is not recognized as an object with mass → need to add Rigidbody component to player object
* Show how to add a C# script to the player object to control movement
  + Explain how C# script is just a customized component
  + Add P1PlayerMovement script to Player object
  + Explain the Start() and Update() functions
* TODO: play around with the different properties for the player object (e.g. size, color, etc.) and get the Player object to continuously move forward in a straight line when the simulation starts
  + Give the hint of the rb variable referring to the player itself and built-in method AddForce() for Rigidbody for applying forces to objects
  + If the function is correct but the box looks like it is being kicked → hint at Slippery component (need to add to Player and Ground objects to get it to move smoothly)
* BONUS: figure out how to get the camera to move with the player
  + Assign FollowPlayer script to the Main Camera and adjust the Z value under the FollowPlayer component of the Main Camera

Part 2:

* Load in Part2 scene (the simple obstacle course)
* Navigate to the P2PlayerMovement script
* Explain the task: finish the code relating to the movement logic for the player object so it can navigate past all the obstacles
  + The logic for maneuvering is whenever the player object collides with an obstacle, it will adjust its internal states based on the side that is open (if any) to indicate what action it should take. The Update() function would then check the inner state and produce the desired behavior
* TODO: implement the following functions:
  + Update(): code for the actual movement based on the boolean inner states
  + OnCollisonEnter(): only the relevant section relating to the logic for adjusting the boolean inner states based on the obstacle’s information
* BONUS: implement ChangeColor() function to change the color of player object based on the input
  + E.g. if ChangeColor(“red”) is called, the box should turn red
  + If successful, try calling function at different parts of the decision-making process to visualize what the code is doing

**Post-Lab:**

1. What type of logic was used to determine whether the player object should dodge?

→ OR logic

1. Let’s think about this simulation using some of the tools from physical symbol systems. What symbols does this machine use to represent its own movements (think about our *boolean* variables)? What do these symbols designate in the simulated world?

→