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

**Pre-Lab:**

1. Have Unity and Visual Studio Code installed on your computer
2. Have Visual Studio Code properly linked to Unity
   * For reference: <https://code.visualstudio.com/docs/other/unity>
3. Watch this video to get acquainted with Unity: <https://youtu.be/IlKaB1etrik>
4. Imagine a robot moving in a tunnel with some obstacles in its path that it needs to navigate around. The obstacles can leave either the left or right side open (or neither!). To navigate past the obstacles, whenever it hits an obstacle, the robot needs to determine whether it needs to move to the right or left to get past the obstacle, or to stop if neither option is available. Write the pseudo-code for the robot’s movement in the tunnel. It’s OK to leave this pseudo-code fairly high-level and informal, but it should touch on the broader issues that need to be tackled for this navigation task.

**Lab:**

Part 1:

* Load in Part1 scene (just a blank world) and explore the basics of the Unity interface
* Unity Interface Basics:
  + 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
* TODO: Finish designing a simple block moving continuously in a straight line on a flat platform and show it to your TA when you are done
  + First, create the ground as a flat cube and add GroundMat as a component to the ground object
  + Create a basic cube as the player object and add the PlayerMat and Rigidbody component to it
    - Why is the Rigidbody component necessary?
  + Figure out how to change colors of the ground and player
  + Add some more 3D objects on the ground to serve as obstacles
  + Add the P1PlayerMovement script as a component to the Player object. This is where you can specify the logic for the player’s movement
    - In the Inspector screen for the Player object, find the added script and specify the rb variable to refer to the Player object. This is so that the code can refer to the player itself when specifying movement
  + Open the P1PlayerMovement script and implement the Update() function so that the player object automatically moves forward.
    - Hint: use the AddForce() method for Rigidbody objects
  + Try running the program. Is the player moving smoothly? If not, why is that?
    - Hint: look at the given assets
* BONUS: figure out how to get the camera to follow the player object
  + Hint: look at the given assets

Part 2:

* Load in Part2 scene (the simple obstacle course)
* Open the P2PlayerMovement script
* TODO: Finish implementing the Update() and OnCollisionEnter() functions so that the player object can navigate through the obstacle course
  + Follow the specifications given in the file
  + Hint: think back to the pseudo-code logic from the pre-lab if you are stuck
* BONUS: Implement the ChangeColor() function to change the player object’s color based on the color input
  + E.g. calling ChangeColor(“red”) should change the box to red
  + If you can get it to work, try inserting it in different functions in the P2PlayerMovement script and see how the color reflects the logic in your code

**Post-Lab:**

Submit your P1PlayerMovement and P2PlayerMovement files as well as the answers for the following questions on Canvas:

1. What type of logic was used to determine whether the player object should dodge?
2. 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?
3. Which part of this lab did you find most challenging?
4. What do you think could be improved for this lab? (not for marks, but feedback is greatly appreciated!)