**CAP 4503 – Assignment 1**

Jessica Bourgade Connor Boyle Sarah Myhre Wendelyn Sanabria

**Summary of Code**

All of the following code files are attached as components to the ‘Player’ GameObject.

MoveAgent.cs

This file merely contains the functionality for the Player object to be controlled by the user. If the user hits the ‘vertical’ keys then the Player moves forward or backward respectively. If the user hits the side keys then the Player rotates in that direction and changes its heading respectively.

WallSensor.cs

The wall sensor sets up a range at the start of the game and assigns it as the distance for its three rays’ sensors. On every update 3 vectors are created to define the direction of the 3 wall sensor rays: left diagonal, forward, and right diagonal. After each vector is created it is passed into a function ‘check\_and\_draw’ to create the sensor. In this function, ‘Physics.Raycast’ is called with the given vector and returns the first object it hits, if it at all. If it hits an object and that object is tagged as a ‘Wall’ then the print data is changed to give the details. Afterwards a ‘Debug.DrawRay’ is called to draw a ray that ends when it hits the wall or reaches its range. These drawn rays are only visible in the debug window, not the game window. After update is finished, the GUI text describing the sensor data is printed to the bottom left hand corner of the screen.

AdjAgentSensor.cs

The adjacent agent sensor sets up its range at the start of the game. It also sets the scale of its child Sphere GameObject to the same value so that it visually represents the range of the adjacent agent sensor. On every update the current positon is acquired passed into a function ‘findAdjObjects’ to create the sensor. In this function, ‘Physics.OverlapSphere’ is called with the current position and pre-defined sensor range. It returns a list of all the collider objects within the range of the position.

These colliders are iterated through. If a collider is an ‘Enemy’ object then the distance from the Player to the collider is calculated. Then the angle between the Player’s heading and the collider is calculated. The object’s local position in comparison to the Player is then looked at to determine what side of the Player the object is on. If the local position’s x value is less than zero then the object is on the Player’s left side and the angle value is negated. Both angle and distance values are added to the print string. After the update is finished, the GUI text describing the objects in range is printed to the upper right hand corner of the screen.

PieSensor.cs

The pie sensor sets up its range at the start of the game. On every update the counters u, d , l, and r are reset back to zero. These counters represent the number of objects in each ‘pie slice’ of the sensor. ‘Physics.OverlapSphere’ is called with the current position and pre-defined sensor rage. It returns a list of all collider objects within the range of the position. These colliders are iterated through. If a collider is an ‘Enemy’ then the vector between the Player and the object is calculated. The angle between the two is calculated using the vector and the Player’s heading.

Depending on the angle, the counter for the corresponding pie slice is incremented. An angle less than 45o and greater than 135o are in the front and back slices respectively. If an object is on the Player’s side then the object’s local x position I relation to the Player is used to determine which side. A smaller x value means that the object is on the player’s right side. The counter values are added to the print string. After update is finished, the GUI text describing the sensor data is printed to the upper left hand corner of the screen. The object’s header and position are also printed.

L**essons Learned**

The group collectively learned about detecting objects around a controlled agent. We learned that any object that we wanted detected needs to have a collider or else the raycast won’t see it. We learned that to detect a group of objects within a set radius of a point to use Physics.OverlapSphere.

Angles in Unity can be difficult to use. Angles between objects calculated with Vector3.Angle will only ever return a value less than 180 so further detection methods were needed to determine if the angle was actually greater than 180 degrees. With no angles greater than 180 degrees, it was not possible to determine whether an object was to the left or right of the Player object. To accomplish this, we got the object’s local position in terms of the Player by parenting it.

We learned how to set up a Github repository and how to sync our files between our various projects. This included committing only the edited files because committing an entire unity project would cause merge conflicts in all the binary project files. We also learned the importance of setting which type of Unity physics we will use early on since some of us used the 2D physics commands and some used the 3D ones. This resulted in some problems with the objects being detected since the type of command requires that the objects in the game used 2D or 3D colliders respectively.