Journal 3

Allie Arnott

Task 1A & 1B: Finished in class

Done as part of class on week 3, any questions or commentary I had were answered then.

```
//velocity = Vector3.zero;
//if ((Input.GetKey(KeyCode.D)) | (Input.GetKey(KeyCode.RightArrow)))
if (Input.GetKey(KeyCode.RightArrow))
{
    //moving right
    velocity += Vector3.right * acceleration * Time.deltaTime;
    leftRight = true;
}
//else if ((Input.GetKey(KeyCode.A)) | (Input.GetKey(KeyCode.LeftArrow)))
else if (Input.GetKey(KeyCode.LeftArrow))
{
    //moving left
    velocity += Vector3.left * acceleration * Time.deltaTime;
    leftRight = true;
}
else
```

Task 1C: 30 minutes (ran out of time)

Felt like I was on the right path here I think if I just checked the original velocity x and y individually for the positive and negative values then added the new slowed velocity when it wouldn't cause the ship to reverse it would have worked. I know though that if it got to the point where the added velocity WOULD reverse the direction of the ship I would need to just set the velocity to vector3.zero to ensure it wasn't being moved anymore.

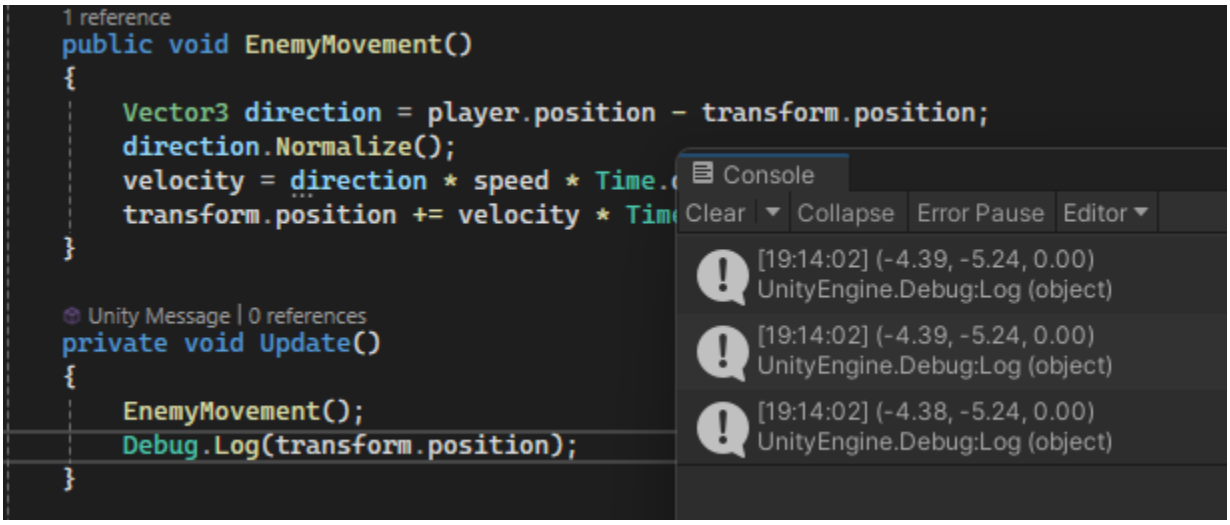
```
    forBack = false;
}
float temp;
temp = velocity.magnitude;
temp = Mathf.Abs(temp);
if (temp > targetSpeed)
{
    velocity = velocity.normalized * targetSpeed;
}
if ((!leftRight) && (!forBack))
{
    Vector3 temp2 = velocity - (velocity.normalized * acceleration * Time.deltaTime);
    if ((velocity.magnitude > 0) && (temp2.magnitude < 0))
    {
        velocity = Vector3.zero;
    }
    else if ((velocity.magnitude < 0) && (temp2.magnitude > 0))
    {
        velocity = Vector3.zero;
    }
    else
    {
        velocity = temp2 * Time.deltaTime;
    }
    //velocity = velocity - (velocity.normalized * acceleration * Time.deltaTime);
}
transform.position += velocity * Time.deltaTime;
//speed = velocity.magnitude.ToString();
```

Task 2A:

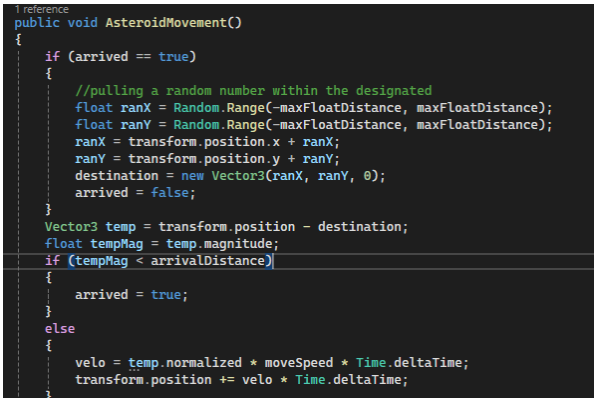
Using a velocity float, public and set by the dev, calculate the vector needed to move the enemy towards the player with the enemy.transform and player.transform by subtracting the one from the other, and getting the result's normalized vector. Multiply that vector by the enemy's velocity float and add it to the enemy's transform each turn so they move towards the player.

Task 2B: 15 minutes

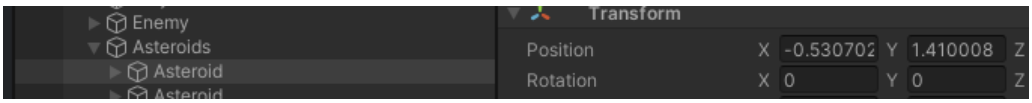
Simple enough to implement, since I assume that the enemy will always be moving towards the player at a standard speed, I did not apply any acceleration or deceleration to them. Note: when testing at least a speed of 100 is recommended.



Task 3: 23 minutes



Gosh they move adorably slow, I had to just crank up the speed to 55 to really see anything. Anyway, at first I had tried random.value * maxfloatdistance for both, and I think that would have worked with a bit more thought? But the asteroids seemed to be stuck, so since I know how random.range works I went back and implemented that instead.



Task 4: 1 hour (ran out of time)

Apologies but around this time I am developing a headache. I am 90% sure I am on the right track, by having a moving variable position to track how far along the draw line is between the stars and having the program iterate through all the previous stars in the list to draw their lines, it should work. But something somewhere isn't working right, and I think it might be how I have the drawPos working with it's movement but I am out of time, and this headache is winning.

```
1 reference
public void DrawConstellation()
{
    // if the current star is the last in the list, then reset to the first in the list
    if (starCurrent >= starTransforms.Count)
    {
        starCurrent = 0;
        starNext = 1;
    }
    else
    {
        //Debug.Log(starTransforms[starCurrent].position + " " + drawPos);
     //otherwise check how close the end of the line is to the next star
     Vector3 temp = starTransforms[starNext].position - drawPos;
        //Debug.Log(drawPos + " " + starTransforms[starNext].position);
        float tempMag = temp.magnitude;
        //if its within a certain distance, simply draw the line to the next star and move the list up by 1
        if (temp.magnitude < 0.01f )
        {
            drawPos = starTransforms[starNext].position;
            starCurrent += 1;
            starNext += 1;
        }
        //otherwise, add the distance that would have been moved in the time since the last frame times however long we want it to take
        //then draw the line
        else
        {
            Vector3 temp2 = starTransforms[starNext].position - drawPos;
            temp2 = temp2 * drawingTime * Time.deltaTime;
            drawPos += temp2 * Time.deltaTime;
        }
        Debug.DrawLine(starTransforms[starCurrent].position, drawPos);
    }
    if (starCurrent == 0)
    {
        //do nothing
    }
```

Above is calculating if the line has gotten to the next star, and trying to draw it.

```
    else
    {
        //draw the lines between all prior stars in the list
        for (int i = 0; i < starCurrent; i++)
        {
            Debug.DrawLine(starTransforms[i - 1].position, starTransforms[i].position);
        }
    }
    Debug.DrawLine(starTransforms[starCurrent].position, drawPos);
}
```

This should be trying to draw the previous lines in the constellation.

I just realized I completely forgot lerp existed. I am an idiot.