## **VINCENT BENNETT**

## Project 4 Check-off

## **Due: END OF DAY TODAY (Tuesday, November 20) at 11:59 pm**

**Place an X in the box that describes your level of progress on each of the following tasks below.**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Done! ☺ | Almost finished | Some progress | A little progress | Have not started |
| Humans and Zombies use separation to keep geometries from intersecting |  |  |  | X |  |
| Zombie finds and pursues closest human | X |  |  |  |  |
| Zombie wanders when all humans are caught |  |  |  | X |  |
| Humans wander when not threatened | X |  |  |  |  |
| Human evades the closest zombie (in range) | X |  |  |  |  |
| Humans and Zombies avoid obstacles |  |  | X |  |  |
| Debug lines are drawn for human and zombie future positions | X |  |  |  |  |
| 3D models used for Humans and Zombies |  |  |  |  | X |
| Weights and conditionals balanced |  |  |  |  | X |
| Above and beyond |  |  |  |  | X |

**Add code snippets and screenshots here:**

protected Vector3 ObstacleAvoidance(GameObject obstacle)

{

Vector3 vecToCenter = obstacle.transform.position - position;

float dotForward = Vector3.Dot(vecToCenter, transform.forward);

float dotRight = Vector3.Dot(vecToCenter, transform.right);

float radiiSum = obstacle.GetComponent<Obstacle>().radius + radius;

if(dotForward < 0)

{

return Vector3.zero;

}

if(vecToCenter.magnitude > safeDistance)

{

return Vector3.zero;

}

if(radiiSum < Mathf.Abs(dotRight))

{

return Vector3.zero;

}

Vector3 desiredVelocity;

if(dotRight < 0) // Left

{

desiredVelocity = transform.right \* maxSpeed;

}

else // Right

{

desiredVelocity = -transform.right \* maxSpeed;

}

Debug.DrawLine(transform.position, obstacle.transform.position, Color.green);

Vector3 steeringForce = desiredVelocity - velocity;

return steeringForce;

}

public Vector3 Flee(Vector3 targetPos)

{

// Step 1: Find DV

Vector3 desiredVelocity = vehiclePosition - targetPos;

// Step 2: Scale vel to max speed

desiredVelocity.Normalize();

desiredVelocity = desiredVelocity \* maxSpeed;

// Step 3: Calculate seeking steering force

Vector3 seekingForce = desiredVelocity - velocity;

// Step 4: Return force

return seekingForce;

}

public Vector3 Seek(Vector3 targetPos)

{

// Step 1: Find DV

Vector3 desiredVelocity = targetPos - vehiclePosition;

// Step 2: Scale vel to max speed

desiredVelocity.Normalize();

desiredVelocity = desiredVelocity \* maxSpeed;

// Step 3: Calculate seeking steering force

Vector3 seekingForce = desiredVelocity - velocity;

// Step 4: Return force

return seekingForce;

}