GitHub:
https://github.coventry.ac.uk/wangh109/304CR-CW1

Video:
https://web.microsoftstream.com/video/9ab7cc1d-3f95-4477-af74-eb54ebc573f1

Or
https://livecoventryac-my.sharepoint.com/:v:/g/personal/wangh109_uni_coventry_ac_uk/EbZ9iO8L131HI-9xjO7OgwsByeQw5YlmSYe7i3wrn_1lPw?e=jgPiNc

304CR GAME AND AI CW1

HAN

9987188

Week 1 Question

Find and briefly describe a bug related to the AI in Cyberpunk2077.

An interesting bug I found in Cyberpunk2077 is that If there is a vehicle damaged in front of the lane, the following vehicles will remain blocking instead of changing lanes. Over time, the entire lane will accumulate many cars, even if the lane next to it is free of any traffic. I recognize this as a bug related to the driving AI.

What do you think causes this bug?

The main reasons for this problem may be due to two reasons. Whether the wrong pathfinding or the lack of a particular steering behavior while AI drives the car. (Millington & Millington, 2006) proposes two types of driving options. The first approach is to generate more than one racing lines to allow the car to achieve its optimal speed. Computer-controlled cars can drive along the predefined path by Splines which is a mathematical curve.

However, there are some shortcomings:

- Will not be able to avoid if there is a crash ahead
- If the player collides, the car will not be able to deviate from the route
- Car cannot overtake

This could be one of the reasons for this bug.

The second method is to let the AI drive the car, put some inputs into the physics simulator to create more realistic effects. This means that the AI often failed to achieve its desired line, especially when hit by another car can create some other problems. The earliest use of this method was in the game Gran Turismo (Polyphonic Digital, 1997). So as in Cyberpunk 2077, the AI drive car will remain in the racing line waiting for the blockage ahead to be relieved. In brief, artificial intelligence often fails to achieve the route it expects, especially if it is pushed by another vehicle or blocked by an obstacle.

Give your answer in terms of how you think the underlying algorithms are working.

In terms of the underlying algorithms, I think the driving AI will first watch the explicit traffic flow or pedestrian on the road. Then enter into the decision-making state which could use FSM /script/ rule-based system to select its destination and then go through the Markov/Fuzzy SM to generate the desired steering. the next step is to use A* to find the path, and finally to steering itself to the right direction.

Finally, to solve the cyberpunk2077 problem, when cars crashed ahead or vehicles blocked the lane sits on the road, A steering behaviors can be added to the AI: The car

can monitor the traffic in the next lane and then wait for a long straight line before overtaking. This kind of behavior has been applied in many driving games from Gran Turismo to Burnout (Criterion Software, 2001).

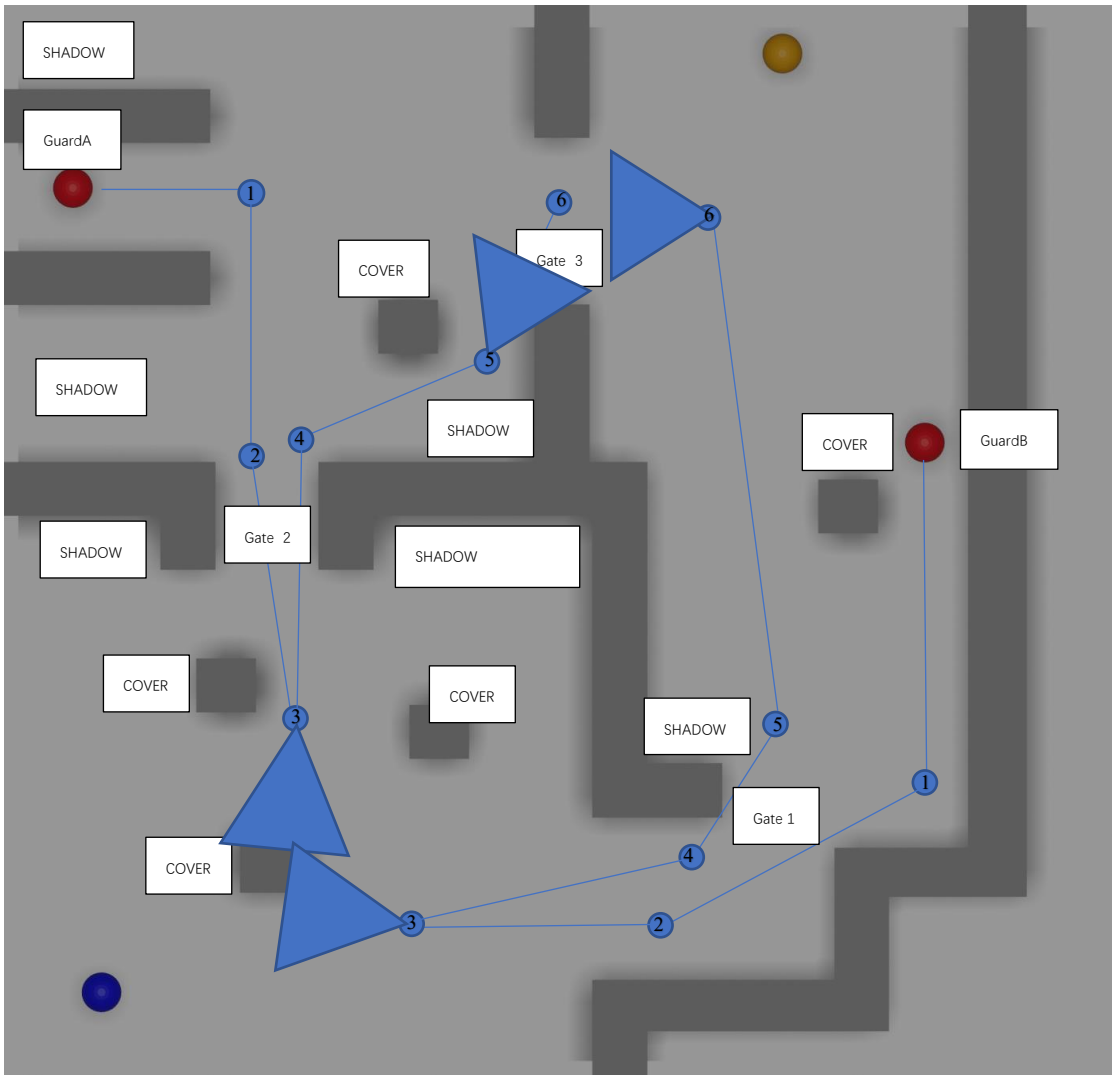
Week 2 Question
Where would you position the waypoints for each guard to create the best gameplay?

A graph usually consists of two types of elements: nodes and edges. Nodes are usually described as points or circles, and the lines between points are called edges.(Millington, 2006)

In terms of pathfinding, waypoints are locations in the level where the pathfinding system is used to connect areas in the level. In normal wayfinding, they are used to tell the AI how to get from point A to point B without going through walls or taking a route that doesn't seem reasonable to the player. To increase the fun of the game, we could have some tactical waypoints. Waypoint tactics are about giving waypoints more information so that the AI can make more complex and informed decisions. (Timothy Carbone, 2019)

In this graph, if players want to reach the yellow ball location, which is the location of the treasure, they need to go through at least one or two of the three gates. To increase the difficulty of the game, the route of travel of the two red balls must be around the three gates for movement. This is to ensure that players do not get the treasure too easily. We can see that there are three small square barriers in the area where the player was born. In the remaining two areas there is also one each. These small square areas are effective in blocking the AI's view as the AI can only see the front 90 degrees of the area. This will help players to find some cover points or shadows to hide.

We could design waypoints with attributes to help players avoid the sight of the guards. In this game. We can simply store two tactical qualities: shadow and cover so we could presume the best places for the player to hide in order to avoid the guard.



Guard A waypoints and their thoughts:

1. Node1 is the starting point.
2. The second point is to block gate2 to prevent players from passing.
3. The third point is to give players more challenges, through the use of cover points and shadow can be very good to avoid the sight of the guards.
4. Node 4 and node 6 are to prevent the player pass through gate2 too easy, the guards will turn back to better guard the gate3 to prevent players from getting the treasure.
5. Node 5 is to eliminate dead space to prevent players from staying in the shadow point too long to wait for the guard to pass.

Guard B waypoints and their thoughts:

1. Node1 and node 2 is the starting point to prevent player pass the gate 1.
2. Node3 is to monitor the lower half of the lower-left corner area. This can be done in conjunction with guard1 to maximize the restriction of the player's position so that the player can maximize the use of cover points to hide themselves.
3. Node4 is to strengthen to prevent players from passing gate1, Node5 is to

eliminate the shadow point in the lower-left corner to prevent players from staying in the shadow point for too long.

4. Finally, Node6 is designed to work with Guard1 to guard the waypoint of gate3.

Week 3
Identify an open problem in AI.

A large part of human success stems from our ability to collaborate. Artificial intelligence-powered machines are playing an increasing role in our lives, it will be crucial to enhancing their ability to cooperate and facilitate collaboration to have better efficiency .

In the field of AI , recent advances have been limited to individual agents operating in constrained environments. The ability to stimulate competition and collaboration among AI agents will be critical to enabling collective knowledge and building dynamics for the development of AI. (KDnuggets, 2019)

This field of open problem is called cooperative AI. Their main object of research is the creation of cooperative agents and the build tools to facilitate collaboration between a group of agents and there are still many open problems in the area of cooperation for AI, such as how to reconcile the degree of common versus conflicting interests among agents and most important: how to cooperate among kinds of agents, such as machines, humans, or organizations? (Dafoe et al, 2020)

Explain if/how it relates to Game AI.

A multi-agent system (MAS) is a computer system composed of multiple interacting intelligent agents (J. Hu et al., 2020). All the agents in it have the same goal. The main difficulty in such systems is how to solve the coordination problem: how to ensure that each agent's decisions are beneficial to the team and that they work together to produce the optimal solution?

Multi-agent cooperation has made some progress in the field of games. Jaderberg et al (2019) designs a computer program to playing the capture-the-flag mode in the video game Quake III Arena, in which two multiplayer teams compete to capture the other team's flag. Through thousands of hours of training, these agents have gradually learned the same successful strategies used by human players.

Zemzem & Tagina (2018) proposed and evaluated TM_Qlearning which a multi-subject reinforcement learning algorithm combining traditional Q-learning. This approach is particularly effective in games, especially in unknown and ad hoc settings, because it improves thematic coordination and accelerates learning by proposing several new cooperative selection strategies.

OpenAI (2019) uses a multi-agent system to produce hide-and-seek games, where up to six different strategies emerge as agents confront each other in hide-and-seek. Each new strategy puts a previously non-existent pressure on the agent to move to the next stage. The results showed that agents emerged with surprising behaviors such as Box surfing, Endless running, and Ramp exploitation (hiders and seekers). These acts go beyond even the imagination of the players

This is particularly important for in-game AI, where multiple agents can cooperate with each other through reinforcement learning to choose the optimal strategy for cooperation or against players in unpredictable environments. By combining the effectiveness of TM_Qlearning, (Zemzem & Tagina, 2018) confirmed that this algorithm achieved good effectiveness in a hunting game demonstration.

Similarly, in game-based reinforcement domain learning, great progress has been made in the past years on two-player zero-sum games, such as chess and Go (Silver et al., 2018), Starcraft II (Vinyals et al., 2019), (two-player) poker (Noam Brown and Anton Bakhtin, 2020) and Two-player tournaments. (Dafoe et al., 2020)

These studies confirm that in the field of AI collaboration, reinforcement learning will be the future trend to solve this open problem: how to cooperate among kinds of agents, such as machines, humans, or organizations?

Week 4

What are the most common AI approaches, and problems, in a game of your selected genre?

A first-person shooter is a first-person perspective shooter in which the player experiences the action through the three-dimensional view of the character. (Jay Garmon, 2005) This kind of genre game's primary design focus is on combat or advancing to certain waypoints to accomplish objectives set by the developer. In single-player mode, this involves the player using guns or other types of long-range weapons to eliminate those pre-made enemies (AI) targets in the game. (Veldhuis, 2010)

The video game industry has been looking for convenient and practical ways for game non-player characters (AI) to make intelligent decisions quickly and efficiently. This includes navigation features that help characters move through 3D environments, such as the most used A* algorithm. The A * is a computer algorithm that extends Dijkstra's algorithm with some of the features of Breadth First Search (BFS). The A* algorithm is widely used for shortest pathfinding in games. It efficiently draws the shortest path between multiple nodes or points on a graph. (Thaddeus Abiy, 2020) In Left 4 Dead, an A* algorithm pathfinding algorithm is used to move an AI to a certain location. (Booth, 2009).

Although the A* algorithm is one of the most versatile algorithms in the industry, it is not foolproof. It cannot handle things like co-movement, moving obstacles, map changes, danger zones, formation, turn radius, object size, animation, path smoothing or many other topics by itself and developers need to solve these problems by combining other algorithms like genetic algorithms, reinforcement learning, etc. (Red Blob Games, 2014)

NPCs in first-person shooter games often use Finite State Machine(FSM) to manage their behavior. (Hoorn, Togelius,& Schmidhuber, 2009). An FSM is a computational model which consisting of more than one states. Only one state will be allowed to active at the one time, so the AI need to switch from one state to another in order to acts different operation. Although finite state machines have been around for a long time, they are still quite common and useful in modern games.

Finite state machines also have a few problems. If we need to develop AI with multiple states, then we have to consider multiple possible states of AI behavior. If the number of such states may reach 100 or more. This causes a dramatic increase in code complexity, making it very difficult to add new programs and the system becomes unmanageable or even has to reinvent machine. Their logic is also limited, and all you can do is edit the transitions from one state to another. It occurs when a state machine reaches a state and cannot exist because no valid input would trigger such activity.

This often leads to behavior with unknown results, which can cause the system to fail. (Vorkflowengine,2020)

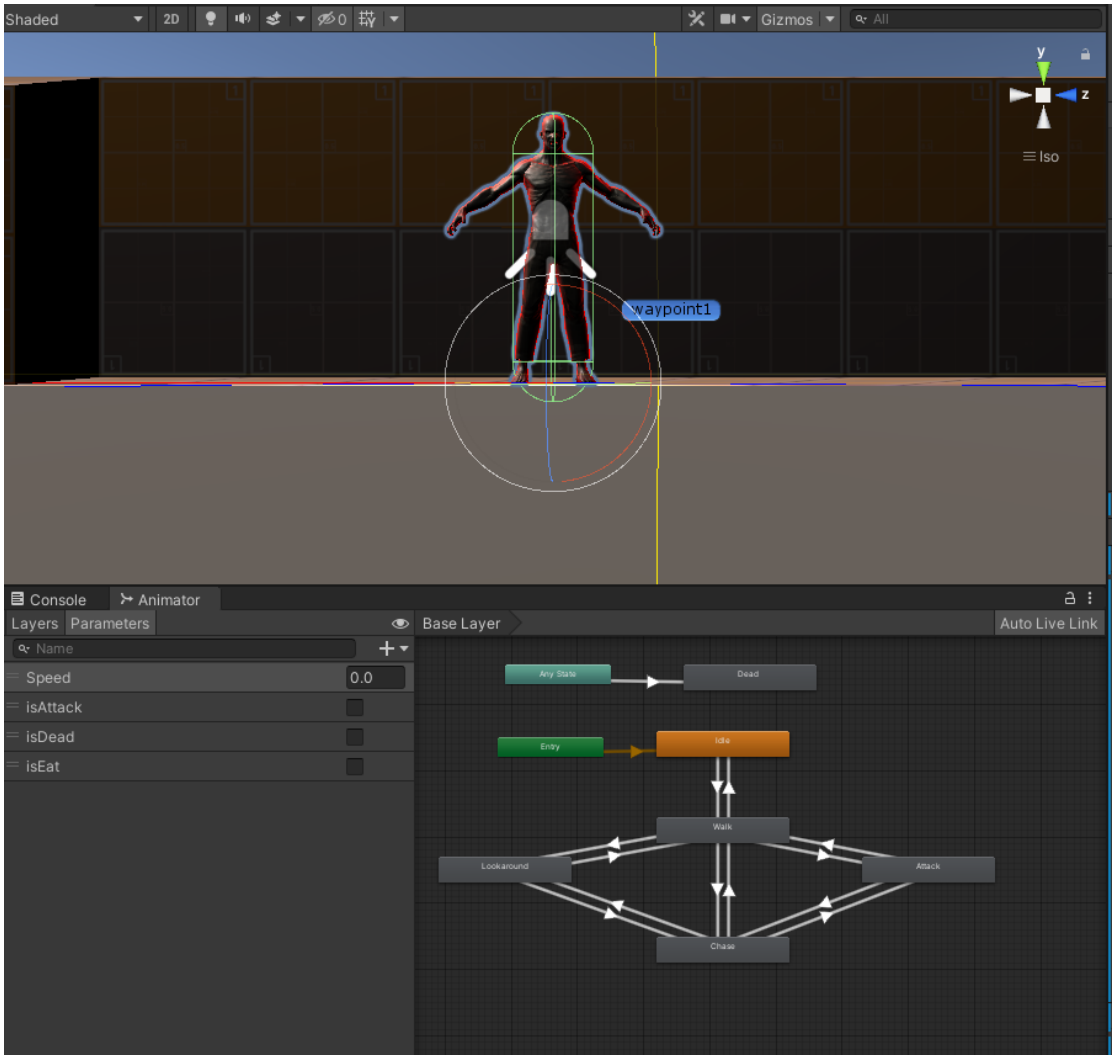
Therefore, the idea of organizing tasks in a tree-like structure (also called behavior trees) has emerged in some games. In contrast to FSM, behavior trees provide a framework for designing more understandable and accessible AI in many cases. Moreover, well-organized trees make visual debugging easier in practice. (Jakob Rasmussen, 2016)

Zombie ~~Shooter~~ Stealth

The name Zombie Shooter is confusing, Although the tutorial doesn't show any shooting capabilities. So, I decided to move the game in the direction of a third-person stealth game which the final goal is to reach the treasure.

In the game, the FSM will be used to control the behavior of the zombies as well as the animation. This greatly extends the single behavior in the tutorial.

```
void FSMController()
{
    //Call the corresponding state handling function according to the current state of the zombie
    switch (currentState)
    {
        case ZoombieState.Walk:
            Walkstate();
            break;
        case ZoombieState.Chase:
            Chasestate();
            break;
        case ZoombieState.Attack:
            Attackstate();
            break;
        case ZoombieState.Dead:
            Deadstate();
            break;
        case ZoombieState.Patrol:
            Patrolling();
            break;
    }
}
```



The zombie attack in the tutorial is just an animation effect. In the zombie stealth game, the zombies will have the ability to kill the player by attack state.

```
void Attackstate() {

    //If the player vision is lost in the attack state, the zombie will automatically enter patrol mode
    if (nearbyPlayer == null)
    {
        currentState = ZoombieState.Patrol;
        agent.ResetPath();
        anim.SetBool("isAttack", false);
        return;
    }

    if (Vector3.Distance(nearbyPlayer.position, transform.position) > 3f) //If the player is in view but cannot reach the attack distance, the zombie enters chase mode
    {
        currentState = ZoombieState.Chase;
        agent.ResetPath();
        anim.SetBool("isAttack", false);
        return;
    }



    if (nearbyPlayer != null)
    {
        //Calculate the angle between the front of the zombie and the player, only the player in front of the zombie can attack
        Vector3 direction = nearbyPlayer.position - transform.position;
        float degree = Vector3.Angle(direction, transform.forward);
        if (degree < 90f / 2 && degree > -90f / 2 && !PlayerAI.isDead)
        {
            anim.SetBool("isAttack", true);
            PlayerAI.HP -= 0.05f;

        }
        else
        {
            //If the player is not in front of the zombie, the zombie needs to turn before it can attack
            anim.SetBool("isAttack", false);
            transform.LookAt(nearbyPlayer);
        }
    }

    Berserk();
```

In the tutorial, the zombies will keep tracking the player until the player dies. This puts the zombies into God mode. A good way to avoid this is to add a sensing feature for the zombies. At first, I wanted to use the collider's trigger to detect if it collided with the player object. (Unity Technologies, 2019) But I found that this method causes huge overhead when there are too many objects around. It affects the game speed. Then I adopted my teacher's suggestion to use distance to determine whether the player is in the perceptual range.

```
public void FindPlayer()
{
    nearbyPlayer = null;

    float distance = Vector3.Distance(player.transform.position, transform.position);

    if (player != null)
    {
        //Collider[] surround = new Collider[20];  //collider buffer size
        //int count = Physics.OverlapSphereNonAlloc(transform.position, MaxRadius, surround); //  Returns the amount of colliders stored into the results buffer

        //for (int i = 0; i < count+1; i++)
        //  {
        //if (surround[i] != null)
        //{

        if (distance < HearingRange)   //if the player is in the range
        {
```

When the player enters the zombie's perception range, we need to define a zombie's field of view angle. This will be calculated by using the angle function of vector3. (Unity Technologies, 2019). We calculate the angle between the line of sight directly in front of the zombie and the line between the player to calculate whether the player is within the zombie's line of sight.

```
    //Debug.Log("player is in the range");
    Vector3 directionBetwen = (player.transform.position - transform.position).normalized;
    directionBetwen.y *= 0;  // height is not a factor of the angle

    float angle = Vector3.Angle(transform.forward + Vector3.up * 0.5f, directionBetwen);  // find angle between zombie direction line and player line

    if (angle <= MaxAngle)  //if player is in the sight angle
```

Finally, we generate a ray to shoot at the player and specify his maximum distance which is the maxRadius. This is to let the player hide behind walls without being discovered by zombies.

```
    if (angle <= MaxAngle)  //if player is in the sight angle
    {
        Ray ray = new Ray(transform.position + Vector3.up * 0.5f, player.transform.position - transform.position);
        RaycastHit hit;

        if (Physics.Raycast(ray, out hit, MaxRadius))
        {
            if (hit.transform == player.transform)  //If ray does hit the player without hitting any obstcles
            {
                PlayerInSight = true;
                nearbyPlayer = player.transform;
            }
            else
            {
                PlayerInSight = false;
                nearbyPlayer = null;
            }
```

If the Raycasthit transform is equal to the player's transform we store the player's transfer into the nearby object. This way the zombie has a target. It is worth noting that here I adjusted the ray upward by 0.8 so that it can effectively simulate the ray emitted from the zombie's eye position, instead of hitting the floor from the ground.

```
    {
        Ray ray = new Ray(transform.position + Vector3.up * 0.8f, player.transform.position - transform.position); //Here the ray added a vector up 0.8 value is to avoid the zombie's ray from
        RaycastHit hit;
```

After the perception function, we can define a patrol behavior to make the zombie walk with a purpose instead of just having the function of rushing to the player as in the tutorial.

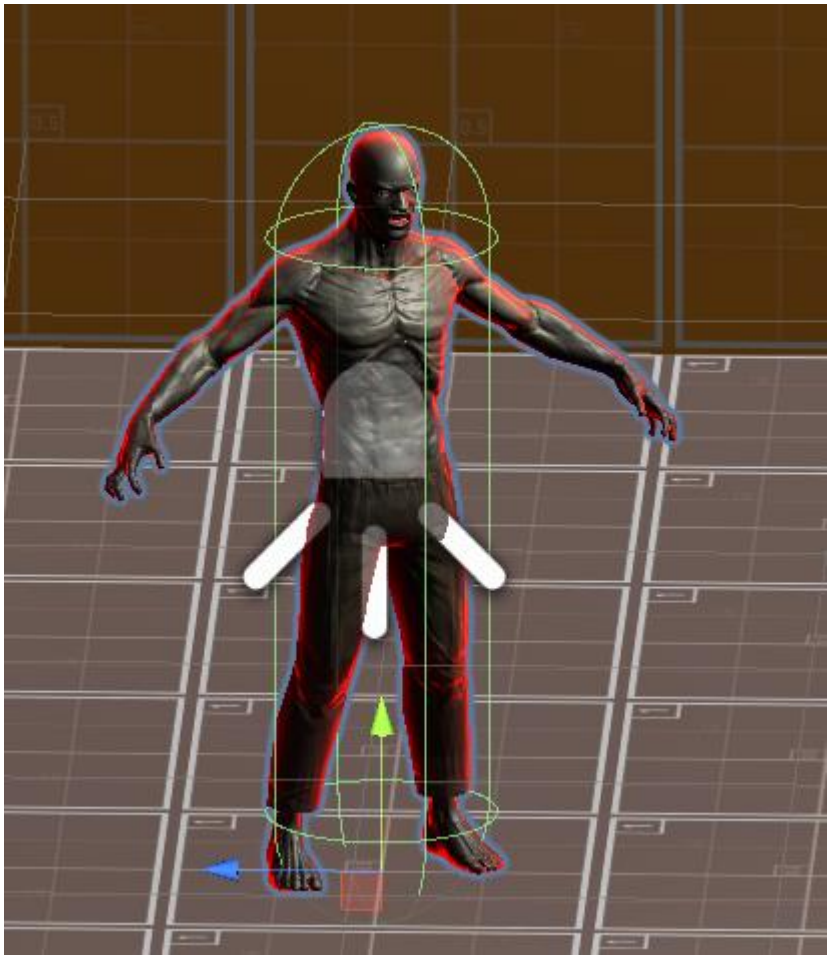
```
void Patrolling()
{
    FindPlayer();

    if (!PlayerInSight) //if the zombie can't see the player then patrolling
    {
        anim.SetFloat("Speed", 1.1f);
        if (agent.remainingDistance < 0.5f)
        {
            anim.SetFloat("Speed", 0.5f);

            stoptimer += Time.deltaTime;   //calculate the stop time

            if (stoptimer > stopTime) //If the stop time is reached
            {
                index++;
                Debug.Log(index);
                index %= 6;   //Get the subscript of the point array using the remainder method cuz we have 6 waypoints so we take the remainder of the six
                agent.destination = wayPoints[index].position;  //zombie walk to the next waypoint
                stoptimer = 0;
            }
        }
    }
    else
    {
        currentState = ZoombieState.Chase;  //if can see the player, chase the player
    }
```

I borrowed the map from the second week's problem on Aula and took the opportunity to implement the waypoints I was considering. Using these waypoints could increase the gameplay of the game rather than the tutorial.

Waypoint alone is not enough. Once the player is familiar with the logic of zombie patrol, the game will become tasteless. So, in the treasure area I add zombie random wandering mechanism. Players will not be able to predict the behavior of zombies, thus adding more fun to the game.

```
void Walkstate()
{
    anim.SetFloat("Speed", Speed);
      //choose a random direction to move
        Vector3 randomRange = new Vector3((Random.value - 0.5f) * 2 * 15.0f, 0, (Random.value - 0.5f) * 2 * 15.0f);
        Vector3 nextDestination = transform.position + randomRange;
        agent.destination = nextDestination;


    Normal();

    if (PlayerInSight)
    {
        currentState = ZoombieState.Chase;
        agent.ResetPath();

        if (Vector3.Distance(player.transform.position, transform.position) < 3f)
        {
            currentState = ZoombieState.Attack;
            agent.ResetPath();
            anim.SetBool("isAttack", true);
            return;

        }
        return;
    }
```

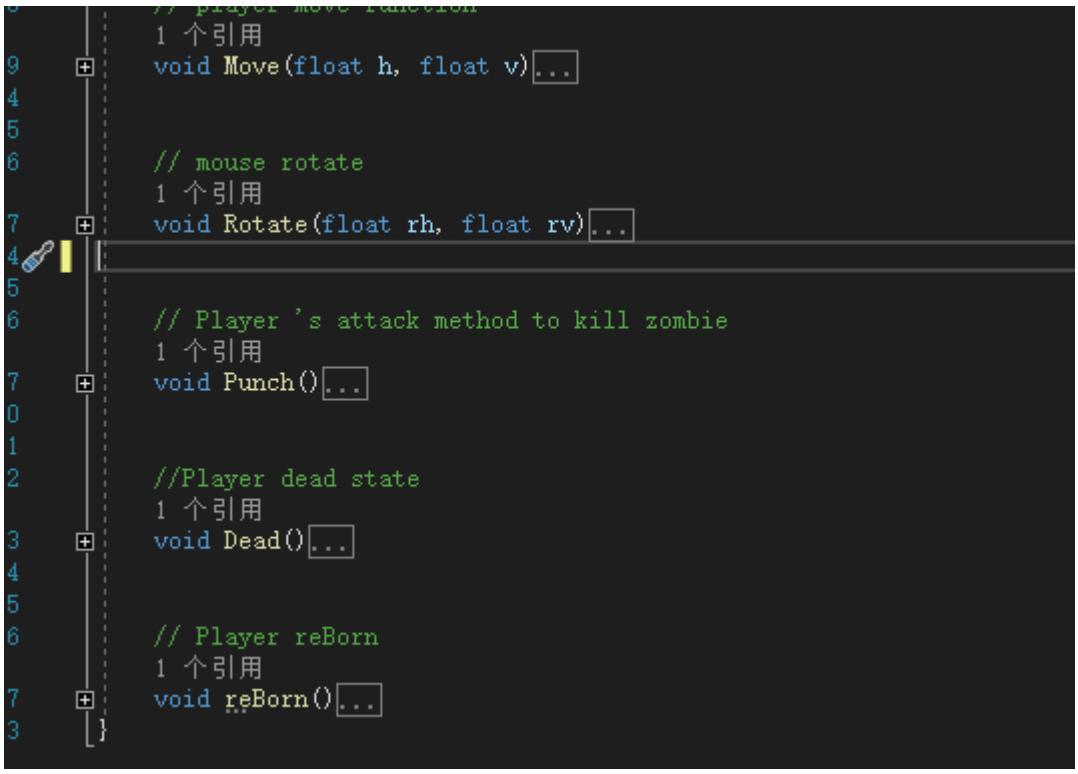
In addition to this, I used shaders to create visual sensory differences to the different states of the zombies. The zombie will turn red when attacking and chasing, which I call Berserk mode.



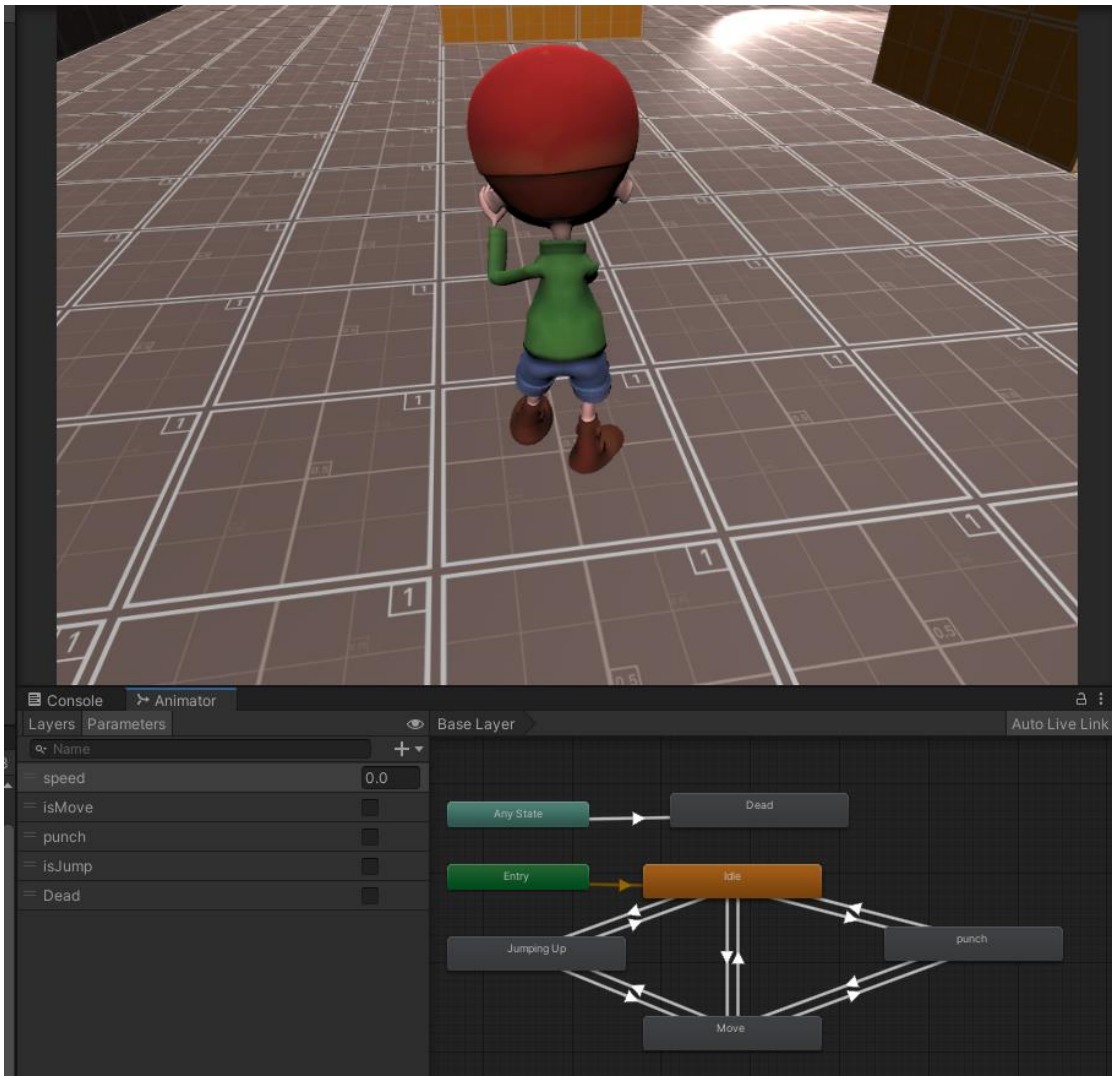
In the tutorial, zombies have no life value. This will make the player very frustrated. I have added the combat feature. When a zombie has more than 0 life while that the player presses the attack button at a distance where he can attack, the zombie will bleed out and eventually die.

```
void TakeDamage()
{
    if (player != null)
    {
        if (HP > 0) //Determine if a zombie is dead by HP
        {
            if (Vector3.Distance(player.transform.position, transform.position) < 3f && PlayerAI.anim.GetBool("punch"))   //If the player is close to the zombie && If the player presses attack
            {
                //Minus HP value
                HP -= 3;

                //We can't use nearbyplayer here because the player is not in the visual perception range of the zombie,however we can use the lookat function to make the player re-enter the nearbyplayer object
                transform.LookAt(player.transform.position);

                //Zombies enter chase mode, in chase mode will be based on the distance to decide to attack or chase
                currentState = ZoombieState.Chase;


            }
        }
        else
        {
            currentState = ZoombieState.Dead; //When the life value of the zombie is less than 0, the zombie is judged to be dead
        }
    }
```

On the player side, I want to give the player AI more freedom, so remove the navmesh agent function makes the player can use the up, down, left and right buttons to move, the player will have the ability to kill zombies by Left mouse click. More information about the player values setting to improve the gameplay can be found in the code comments.

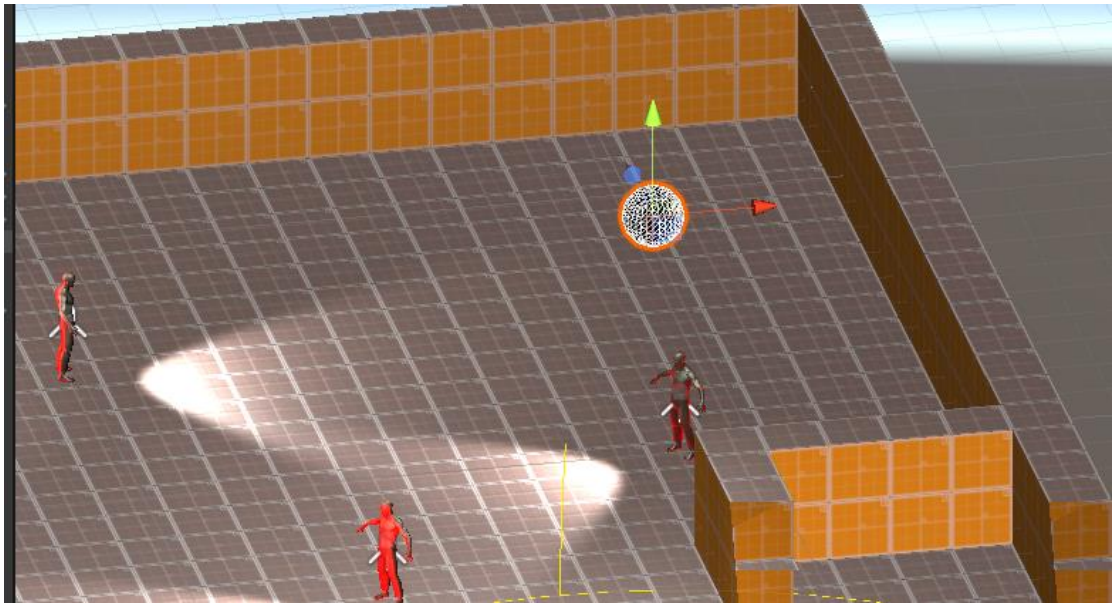


Both zombies and player have well-established animation controller to refine their behavior.



Finally, I set up treasure points and a restart screen to increase the gameplay by more friendly to the player.

Treasure goal:

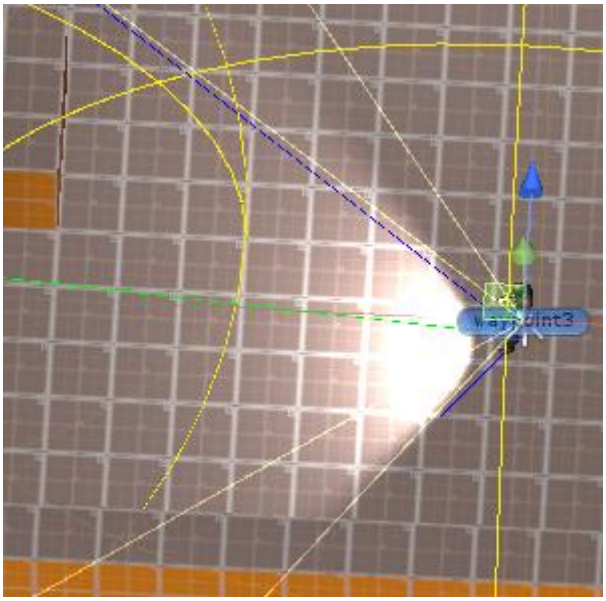


Game End canvas:

In addition to enhancing gameplay on the designed waypoints. In the zombie base value also has some reflection on the gameplay. Such as the HP value of the zombie is set at 10 because the player does 3 damage per attack. This is to ensure that the zombie is not so easy to be killed to set a better game difficulty. The zombie's speed is set to a value relative to the player. It will go up to 2 when the zombie is chasing the player to make sure the zombie can catch up with the player

```
public float Speed = 0.9f;
public float HP = 10;                   //The HP value of the zombie is set at 10 because the player does 3 damage per attack.
                                        //This is to ensure that the zombie is not killed too easily and that killing the zombie is not too time consuming.
                                        //So three attacks to kill a zombie would be a good value to improve game play.
//zombie sensor
public float HearingRange = 10f;        //Considering the size of the map and the perception of the zombie, this value is set to 10.
```

The zombies in the game have a 90-degree angle view to better simulate reality and to comply with some industry standards. This is to try to mimic the feeling of nature by simulating the human eye. In order to make players have a better game experience, the zombie body specially set up a point light to have the same angle with the zombie field of view to implicitly inform the player of the zombie vision range.



At this point, a simple zombie stealth game is complete. By extending the zombie's perception function and strategic patrol points, it gives the player a great gaming experience as well as increases the gameplay, In the future, perhaps adding a sneak function to the player to more strongly enhanced gameplay.

References

Criterion Software. (2001). Burnout [computer software] https://www.ea.com/games/burnout?isLocalized=true

Dafoe, A., Hughes, E., Bachrach, Y., Collins, T., McKee, K. R., Leibo, J. Z., Larson, K., & Graepel, T. (2020). Open problems in cooperative AI. *arXiv Preprint arXiv:2012.08630,*

J. Hu, H. Niu, J. Carrasco, B. Lennox, & F. Arvin. (2020). *Voronoi-based multi-robot autonomous exploration in unknown environments via deep reinforcement learning.* https://doi.org/10.1109/TVT.2020.3034800

Jaderberg, M., Czarnecki, W. M., Dunning, I., Marris, L., Lever, G., Castañeda, A. G., Beattie, C., Rabinowitz, N. C., Morcos, A. S., Ruderman, A., Sonnerat, N., Green, T., Deason, L., Leibo, J. Z., Silver, D., Hassabis, D., Kavukcuoglu, K., & Graepel, T. (2019). Human-level performance in 3D multiplayer games with population-based reinforcement learning. *Science, 364*(6443), 859-865. https://doi.org/10.1126/science.aau6249

Jakob Rasmussen. (2016). *Are behavior trees a thing of the past?* https://www.gamasutra.com/blogs/JakobRasmussen/20160427/271188/Are_Behavior_Trees_a_Thing_of_the_Past.php

Jay Garmon. (2005). *Geek trivia: First shots fired.* https://www.techrepublic.com/article/geek-trivia-first-shots-fired/

KDnuggets. (2019,). The emergence of cooperative and competitive AI agents. https://www.kdnuggets.com/the-emergence-of-cooperative-and-competitive-ai-agents.html/

Millington, I., & Millington, I. (2006). *Artificial intelligence for games*. CRC Press LLC.

Noam Brown and Anton Bakhtin. (2020, Dec). ReBeL: A general game-playing AI bot that excels at poker and more. https://ai.facebook.com/blog/rebel-a-general-game-playing-ai-bot-that-excels-at-poker-and-more/

OpenAI. (2019). *Emergent tool use from multi-agent interaction.* https://openai.com/blog/emergent-tool-use/

Red Blob Games. (2014). *Red blob games: Introduction to A*.* https://www.redblobgames.com/pathfinding/a-star/introduction.html

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., & Hassabis, D. (2018). A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science, 362*(6419), 1140-1144. https://doi.org/10.1126/science.aar6404

Technologies, U.*Unity - scripting API: Vector3.* https://docs.unity3d.com/ScriptReference/Vector3.html

Technologies, U. (2019). *Unity - scripting API: MeshCollider.* https://docs.unity3d.com/ScriptReference/MeshCollider.html

Thaddeus Abiy. (2020). *A* search | brilliant math & science wiki.* https://brilliant.org/wiki/a-star-search/

Timothy Carbone. (2019, -12-09T22:39:05+00:00). Artificial intelligence in games – waypoint tactics. https://timpcarbone.com/2019/12/09/artificial-intelligence-in-games-waypoint-tactics/

Unity Technologies. (2019a). *Unity - scripting API: Collider.OnTriggerEnter(collider).* https://docs.unity3d.com/ScriptReference/Collider.OnTriggerEnter.html

Unity Technologies. (2019b). *Unity - scripting API: Vector3.angle.* https://docs.unity3d.com/ScriptReference/Vector3.Angle.html

Veldhuis, M. O. (2010). (2010). Artificial intelligence techniques used in first-person shooter and real-time strategy games. Paper presented at the *Human Media Seminar: Designing Entertainment Interaction, , 2011*

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., Oh, J., Horgan, D., Kroiss, M., Danihelka, I., Huang, A., Sifre, L., Cai, T., Agapiou, J. P., Jaderberg, M., . . . Silver, D. (2019). Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature, 575*(7782), 350-354. https://doi.org/10.1038/s41586-019-1724-z

Vorkflowengine. (2020). *Why developers never use state machines.* https://workflowengine.io/blog/why-developers-never-use-state-machines/

Zemzem, W., & Tagina, M. (2018). Cooperative multi-agent systems using distributed reinforcement learning techniques. *Procedia Computer Science, 126*, 517-526. https://doi.org/10.1016/j.procs.2018.07.286