SPO2001
Nord University

README 🧟 🧠

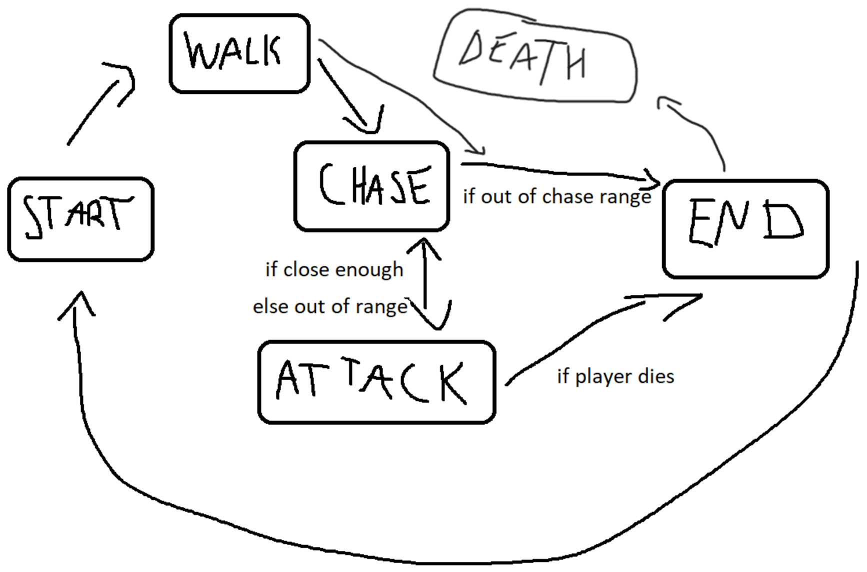


Figure 1: All necessary states for the first idea of our Finite State Machine (FSM)

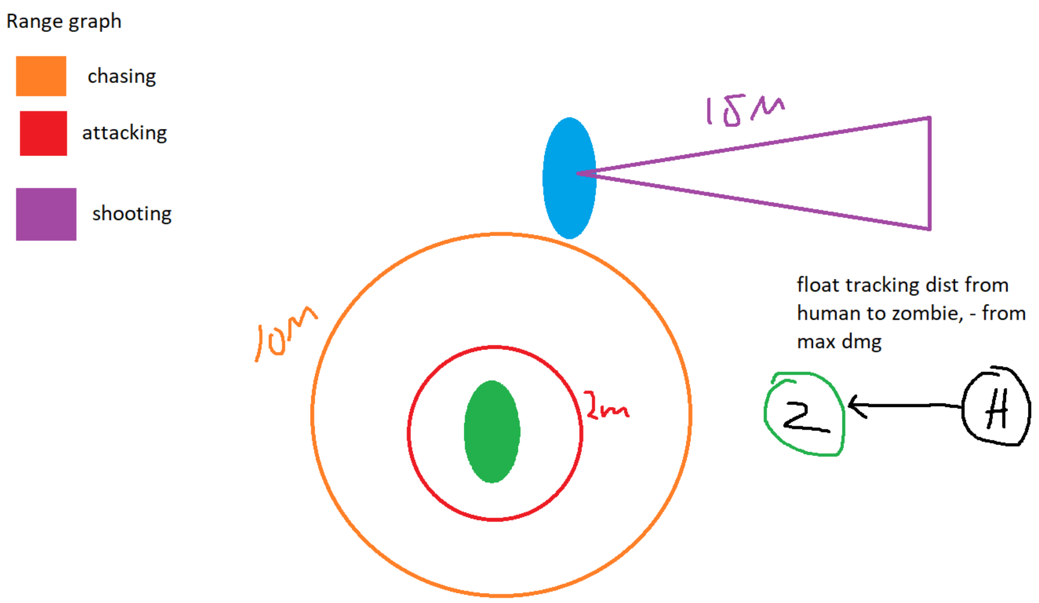
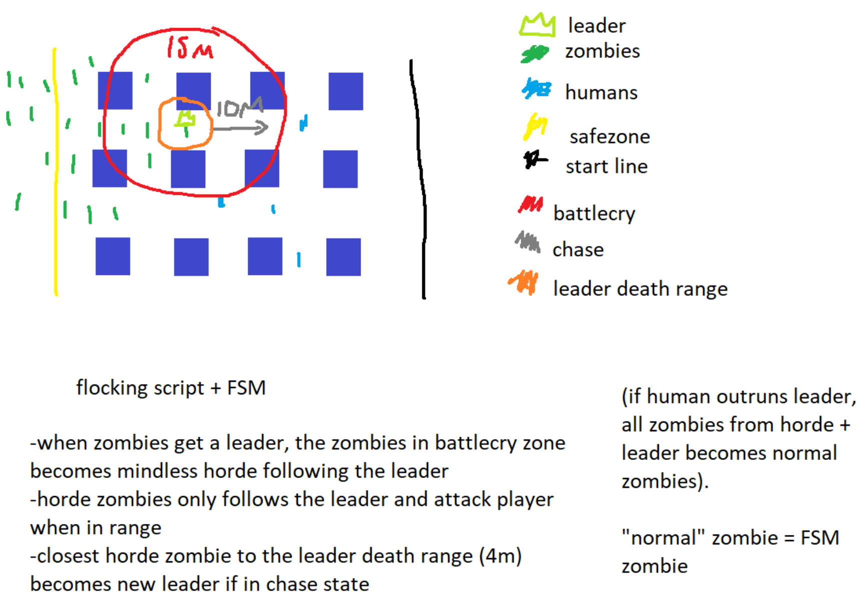


Figure 2: A graph representing the ranges for the zombies and the humans



flocking script + FSM

-when zombies get a leader, the zombies in battlecry zone becomes mindless horde following the leader
-horde zombies only follows the leader and attack player when in range
-closest horde zombie to the leader death range (4m) becomes new leader if in chase state

(if human outruns leader, all zombies from horde + leader becomes normal zombies).

"normal" zombie = FSM zombie

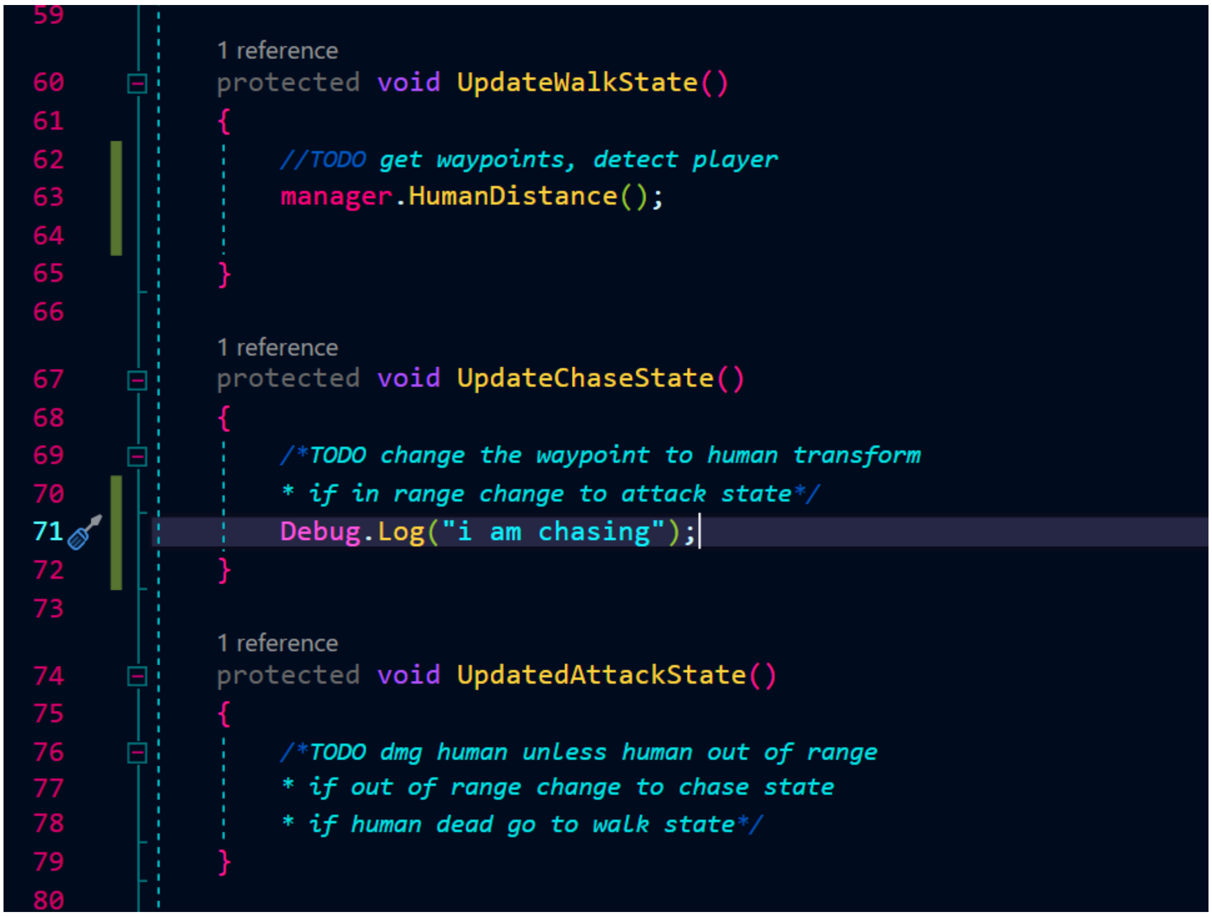
Figure 3: Graph for visualising more in-depth how the leader/horde is meant to work in part 4

Preparation:

We started by discussing the general framework of the zombie behaviour, with all 4 parts in mind. With this we visualised the core needs for the sensory system of the AI and made figures 1-3. We set up the Unity project with the given package. Made some small adjustments to the prefabs (visually) to adhere to our liking.

Part 1:

We started with making the BaseState.cs script which holds the main functions for our FSM. The FSM.cs script is the “brain” of the zombies. In this script we have the different states that the zombie can enter, walk, chase, attack and death. We realized that death was not needed for Part 1 but decided to keep it since it will be utilized later. So what we wanted to happen is that when the game starts the zombie enters the walk state, and when it gets close enough it enters the chase state and then if close to the human it attacks. We realized that this didn’t want to work the way we intended.



```
1 reference
protected void UpdateWalkState()
{
    //TODO get waypoints, detect player
    manager.HumanDistance();

}

1 reference
protected void UpdateChaseState()
{
    /*TODO change the waypoint to human transform
    * if in range change to attack state*/
    Debug.Log("i am chasing");
}

1 reference
protected void UpdatedAttackState()
{
    /*TODO dmg human unless human out of range
    * if out of range change to chase state
    * if human dead go to walk state*/
}
```

Figure 4: First iteration of FSM script

So Charlie did a little work on his own, researching and then coming to the conclusion that mixing the FSM with NavMesh would work better. So then we got started on making the new script, also had to change on the prefab adding the NavMesh Agent component. This way we got all the variables for making it move, and the area for the zombie to move on. The new scripts all stem from a State.cs script. This script contains all the enums for the different States and the enums for the Events that handles where in the state the zombie is. There are also other methods in there that we utilize in the different states. The ZombieController.cs script is the FSM handler, it is a small script that keeps the human transform and it is where we call the method Process() from the State.cs script. The Process() method is what keeps track of the progress in the Event stages, which are Enter, Update and Exit. All the states are their own scripts for easier readability and to not have a 1000 line script. For holding all the waypoints which we use for the zombie to wander around is the Path.cs script which is a singleton script. This makes sure that we only have one instance of the waypoints in the scene. The ZombieIdleState.cs is the first state the zombie enters, this transitions into either the Chase state or Walk state, this depends on if the zombie can see the human or not, so by exiting the Idle state and entering the other state the Process() continues. In the ZombieWalkState.cs we calculate the closest waypoint for when we go from chasing/attacking to walking again when we enter the state. When it is in update we walk from waypoint to the next and if the zombie can see the human it goes to nextState which is the Chase state. In ZombieChaseState.cs it follows the human until two cases, one when the zombie is close enough to attack the human and it then enters the Attack state or if the zombie no longer can see the human then it goes into Walk state. Last is ZombieAttackState.cs, first we stop the zombie so it can attack, it won't walk and attack. It will turn towards the human to attack it. If the zombie is out of range then it goes into the Idle state, and the reason for doing this instead of going into Chase or Walk is that from Idle it can easily go into those states if the conditions are met. But if it is in Chase or Walk it will take a longer time to go through the code to reset.

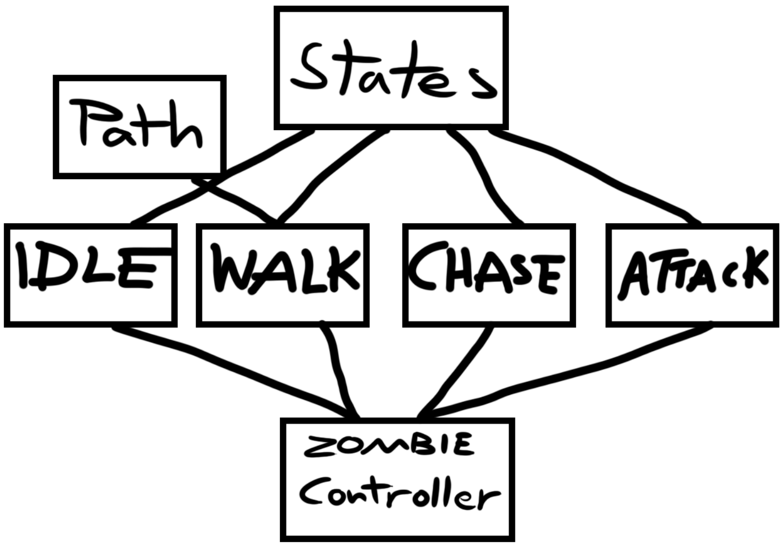
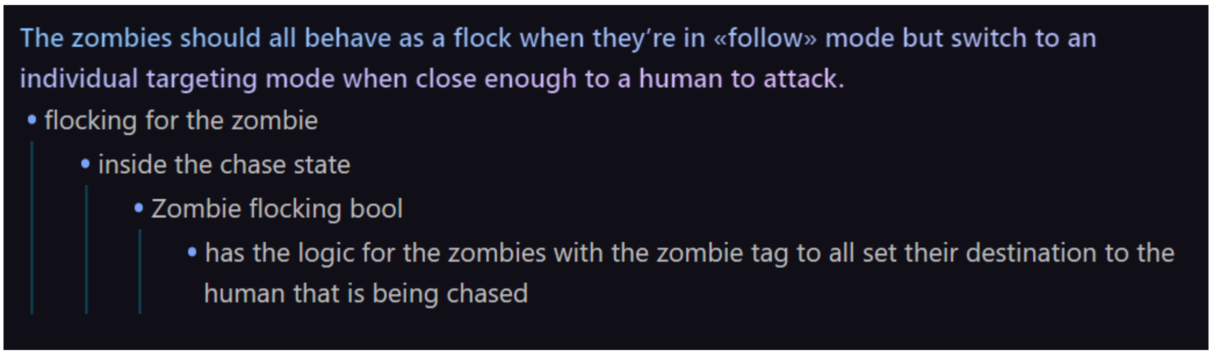


Figure 5: new Finite State Machine

Part 2:

With everything set up from *Part 1* we now started on looking on how to expand on the scripts we already have with making variants of the ones needed. We also set up the NavMesh, making the plane walkable and the buildings and walls non-walkable so that the human and zombies wouldn't walk through them. We also made sure that the zombies won't be able to pass through the safezone by making the collider bigger and added so that if the zombie enters by OnTriggerEnter, the gameobject gets destroyed. The human spawns anywhere along the start line, this is chosen by a random range from one side to the other, and sets immediately out to get to the safe zone. We use the NavMeshAgent method called SetDestination, we just declare the safezone as a variable by setting the NavMesh agent's destination to right behind the safe zone. If at any point the player clicks with the mouse on the screen, it sends a raycast and the point where the mouse clicks becomes the new destination for the navmesh agent, but when they get within 1 meter of it, they go back to getting to the safe zone. If the zombies get close enough it will attack the human giving you a canvas popup saying “All Humans Killed”. If the human gets to the safezone a canvas popup will say “All Humans Survived”. This since we only have one human and it was easy to add.

Part 3:


The zombies should all behave as a flock when they’re in «follow» mode but switch to an individual targeting mode when close enough to a human to attack.

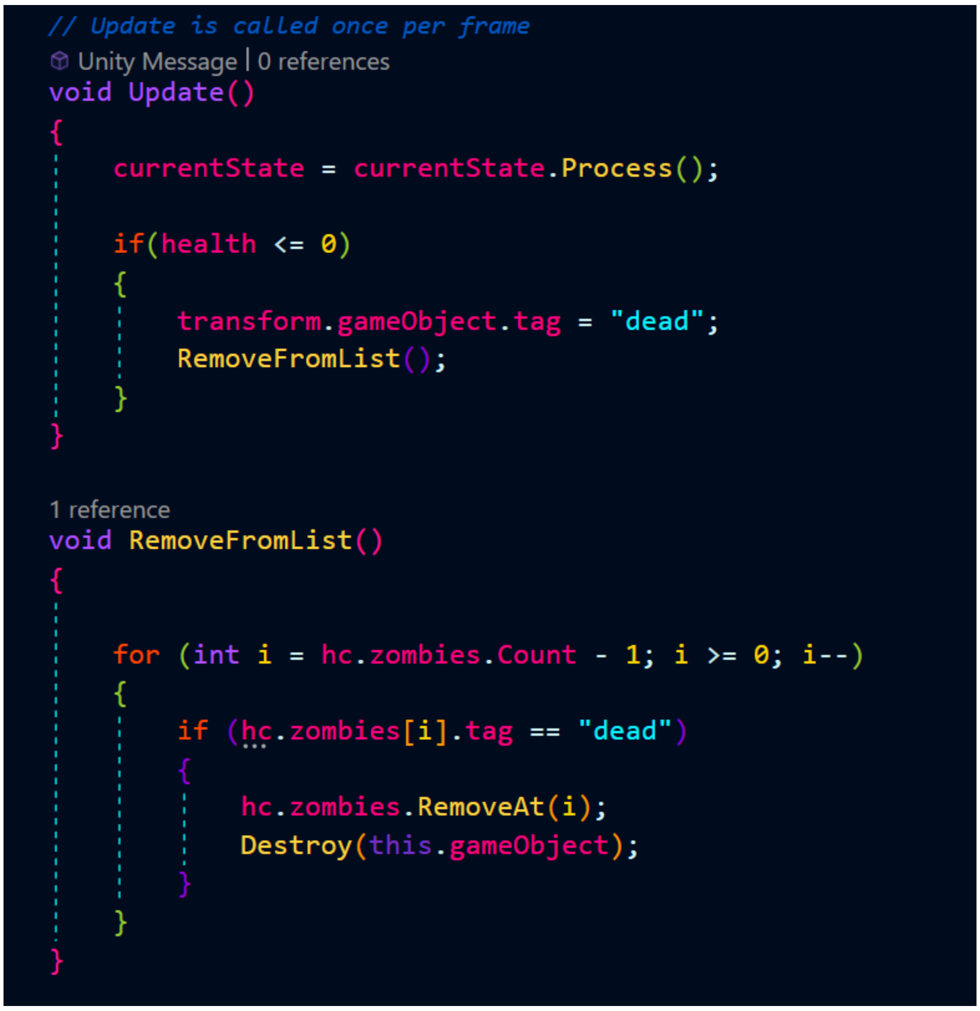
- flocking for the zombie
  - inside the chase state
    - Zombie flocking bool
      - has the logic for the zombies with the zombie tag to all set their destination to the human that is being chased

Figure 6: notes for how to do the flocking

Zombie flocking: “The zombies should all behave as a flock when in <<Follow>> mode”. We interpreted the instructions so that the zombies that detect the human and are following it (i.e are in the ‘Chase’ state) behave as a flock, and those that are out of view of the human wander around as usual. Basically when one zombie starts chasing the player it goes into the Flock state everyone that is closeby will also automatically enter the Flock state until it is close enough to go into the Attack state. Then it will execute that on its own.

Next we started working on the weapon part. One big trouble we had was getting the gun to aim at the nearest zombie, because we started out trying to make the gun itself aim at it. After some mind twists and iterations we made it work by having it on the Human Controller script, since the human has to find the nearest zombie to shoot anyway, it made sense to make it aim there as well. It goes through the list of transforms we have of the zombies, and then calculate from that the closest and “best” to shoot at zombie. We call the WeaponHandler method from the Weapon_Part3.cs script. This is where the shooting happens and later will add the correct damage.

Next part was making the bullets deal damage, we made a method inside the ZombieController_Part3.cs script called TakeDmg(int dmgAmount). This gets called all the places we want to tweak the dmg the zombie will take. We also had some troubles with killing the zombies. Since we have a list of Transforms that gets iterated through during play we can't just remove it. But after looking into how to do a reverse for loop to make it iterate through it and remove the empty ones the script seemed to be working again. This happens in the RemoveFromList() method.



```
// Update is called once per frame
Unity Message | 0 references
void Update()
{
    currentState = currentState.Process();

    if(health <= 0)
    {
        transform.gameObject.tag = "dead";
        RemoveFromList();
    }
}

1 reference
void RemoveFromList()
{

    for (int i = hc.zombies.Count - 1; i >= 0; i--)
    {
        if (hc.zombies[i].tag == "dead")
        {
            hc.zombies.RemoveAt(i);
            Destroy(this.gameObject);
        }
    }
}
```

Figure 7: Removing from list

Next we made the ammo prefab, it is a cube gameobject with a script called Ammo.cs that holds how much ammo to be picked up. In the HumanController_Part3.cs we have so that when the human collides with the ammo it gets picked up. In the ZombieController_Part3 we have the random chance for the ammo pouch to drop. We also have a maxAmmo so that if the human picks up ammo and goes over the limit, it's set to the max instead., and an UI element to display how much ammo is left. If the ammo goes down to zero the weapon is unable to fire a bullet.

Lastly, we implemented a way to check the distance to the zombie before shooting, and making the damage dependable on the distance, + adding a small bonus that had a random size to it as well.

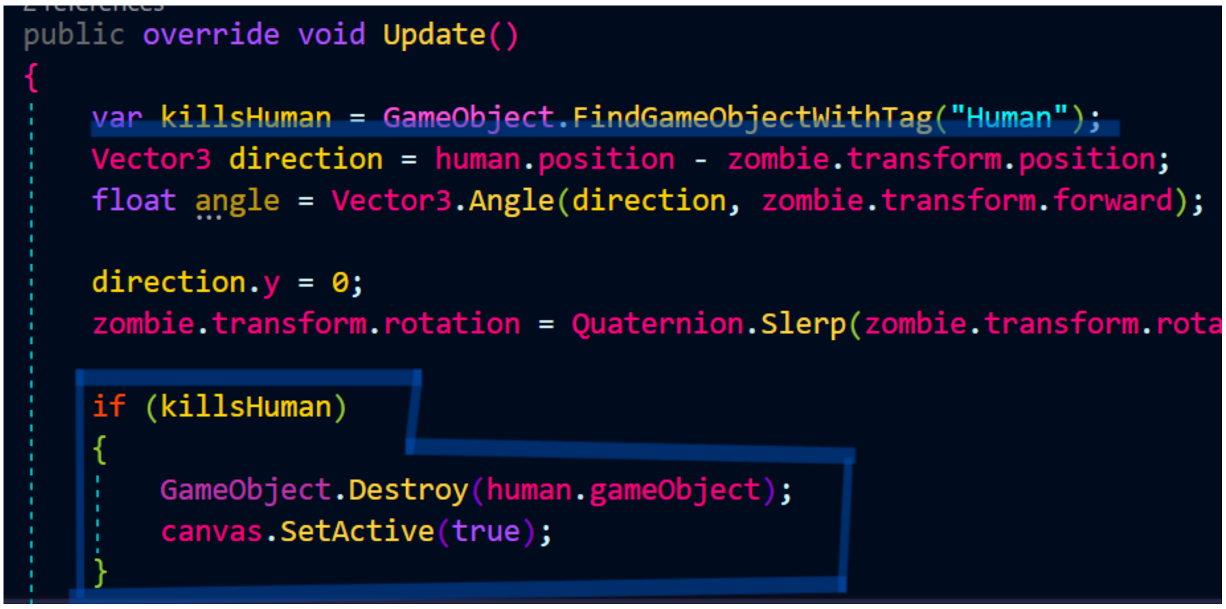
Part 4:

We looked at what we already had and figured out what we needed for finishing part 4. We figured that the way we were doing flocking was pretty much what was required for part 4, but the Human controller needed to be configured for having multiple humans, and to select one to give it new waypoints. The weapon and ammunition drop system needed to be created, and the win/lose conditions altered.

We struggled a lot with the selection of humans. Since what we were doing before was having it happen in update so that all the humans would move to the same spot at the same time. But we came to the conclusion that it was hard to implement to our already existing code so we made new scripts. One for each of the humans called Human[insert their number].cs. These all get called in a script called SelectingHuman.cs this is where we have each human assigned to one Button UI. We had to use a lot of bools to make only one human respond at the time, without the bools all the humans would respond to whichever button was pressed. Since we are using the same scripts as in Part 3 we had to go back and add to the human there. So far it seems to be working fine.

The randomization for the weapon was probably our easiest task in Part 4. In the AmmoWeaponSpawner.cs script we have the bounds selected with the implemented struct in Unity called Bounds. We set the spawning area in the inspector and then when we instantiate the weapon and ammo it gets a random position chosen by the random range from min to max. We have a timer that spawns them at random intervals set between our chosen minTime and maxTime. And we use a for loop to make sure we can just spawn them endlessly. The weapon instantiated has a script on it called WeaponRandomizer.cs, this script has the randomization for fire rate, shooting distance, ammo and damage. In start we set all of these to be of different Random.Range values. Then in our RandomizerPlz() method we set the values from where they are on different scripts to be of the random value made here. We also make sure that the human can’t have more ammo than the max ammo.

Lastly is the win condition for the zombies, right now we have it like this. In the ZombieAttackState_Part3.cs we have the Update method where all the important stuff happens. It finds the human with the tag “Human” and kills it, and then the canvas is SetActive(true). But that doesn't work for having more humans than one. So when one gets killed the canvas gets set immediately.



```
public override void Update()
{
    var killsHuman = GameObject.FindGameObjectWithTag("Human");
    Vector3 direction = human.position - zombie.transform.position;
    float angle = Vector3.Angle(direction, zombie.transform.forward);

    direction.y = 0;
    zombie.transform.rotation = Quaternion.Slerp(zombie.transform.rota

    if (killsHuman)
    {
        GameObject.Destroy(human.gameObject);
        canvas.SetActive(true);
    }
```

Figure 8: Killing human(s)

We tried different approaches, using ints, bools, different scripts, checking for null, checking the length, etc etc. But in the end we ran out of time and got left with what we knew worked. You are able to kill all the humans, but at the first death the zombies win canvas and get SetActive(true). The reason for it not working is that when the human gets destroyed the script on it also gets destroyed which means we end up with the MissingReferenceException. We tried making arrays and lists, and putting it on other scripts, other objects etc, but nothing seemed to work as intended.

Cited Sources:

"Brain Meal" asset by X-Factory from the Unity asset store, a brain model to represent the juicy organ that the zombies so desire:
https://assetstore.unity.com/packages/3d/props/brain-meal-89596

Kyaw, S. A., Peters, C., & Swe, N. T. (2013). *Unity 4.x Game AI Programming*. Packt Publishing. (chapter 2)

Menshov, A. (2021, July 20). *Simple state machine example in C#?* Stackoverflow.
https://stackoverflow.com/questions/5923767/simple-state-machine-example-in-c

Microsoft. (n.d). *List<T> Class*. Microsoft.
https://learn.microsoft.com/en-us/dotnet/api/system.collections.generic.list-1?view=net-7.0

Pêcheux, M. (2021, July 25). *Make a basic finite state machine [Unity/C# tutorial]* [Video]. YouTube. https://www.youtube.com/watch?v=-VkezxxjsSE

"Sci-Fi Gun Light" asset by Factory Of Models, a gun asset we used for our humans:
https://assetstore.unity.com/packages/3d/props/guns/sci-fi-gun-light-87916

Unity Technologies. (2022, December 09). *Bounds*. Unity Documentation.
https://docs.unity3d.com/ScriptReference/Bounds.html

Unity Technologies. (2022, December 09). *Move To Destination*. Unity Documentation.
https://docs.unity3d.com/Manual/nav-MoveToDestination.html

Unity Technologies. (2022, December 09). *Building a NavMesh*. Unity Documentation.
https://docs.unity3d.com/Manual/nav-BuildingNavMesh.html

de Byl, P. (2021, March 05). *Finite State Machines*. Unity Learn. Retrieved (2022, December 12).
https://learn.unity.com/project/finite-state-machines-1

Safwan. (2014, November 28). *How to spawn enemy at random intervals?* Stack Overflow. Retrieved (2022, December 15).
https://stackoverflow.com/questions/27187555/how-to-spawn-enemy-at-random-intervals