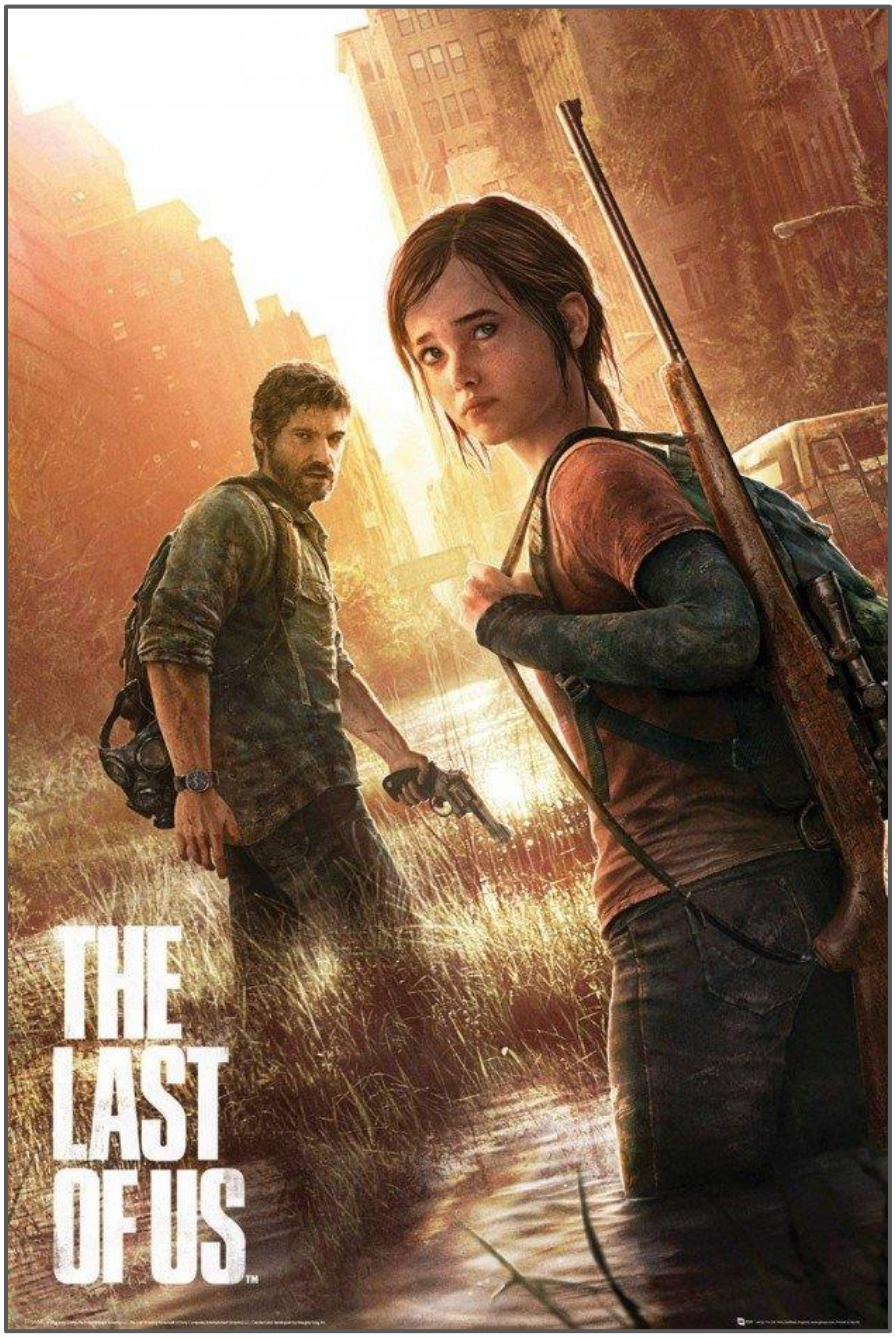

The Last of Us

WRITTEN REPORT ON AI USAGE

Kimberley Kane 15021826 | CU6052 AI for Games | 14/04/2019

Contents

Introduction

In this report I will cover how the artificial intelligence of agents (named 'The Infected') are implemented in a popular survival horror game called The Last of Us. I will outline the investigative research I have taken and demonstrate my understanding and knowledge of the AI used and how my findings will aid the development of my own AI game.

The Last of Us: History

The Last of Us is a 3D action-adventure survival horror video game, played in third-person perspective. The game was developed by Naughty Dog and published by Sony Computer Entertainment. The game was released for the PlayStation 3 in June 2013, with the PlayStation 4 remastered version being released in April 2014.

The Last of Us is the fourth best-selling video game for the PlayStation 3 of all time and has won over 240 Game of the Year awards. The game received critical acclaim upon release and has a score of 10/10 on IGN entertainment.

The Last of Us: Storyline

The game is based in the United States in a post-apocalyptic environment. The player must defend themselves against hostile humans and cannibalistic creatures infected by a mutated strain of the Cordyceps fungus.

The player traverses through the environment to advance the story and can collect firearms, improvised weapons and resources to help them on their journey. Through-out the game the player will encounter various enemies and must either fight or use the stealth mechanic to sneak by them.

Through-out most of the game, the player takes control of Joel, a man tasked with escorting a young girl, Ellie, across the United States. Ellie is immune to the Infected and she is civilizations last hope of destroying the infection and mutant creatures permanently. Joel is in search of the Fireflies, a group of survivalists' that wish to produce a vaccine for the infection and need Ellie to do so.

Why The Last of Us?

The Last of Us is one of my favorite games of all time. The main reason I enjoy this game is because of the clever AI they have used. In my opinion, the AI implemented is extremely realistic and unique to other games I have played. As you further progress in the game, I feel you start to develop a closeness towards the characters due to the realism they portray, and you can easily relate to the game in many ways.

The enemy AI is some of the best I have seen; you genuinely feel threatened, as each enemy has their own behavior pattern and they are very unpredictable. The game carries a very real threat in that sense, making the player think tactically about their next move, as one wrong move will kill them.

For my AI project, I wish to create a 2D side-scroller game that closely resembles the AI used in The Last of Us.

Artificial Intelligence in The Last of Us

The AI system used in The Last of Us makes a distinction between the high-level decision logic (skills) that decides what the character should do and the low-level capabilities (behaviors) that implement those decisions. For example, movement is encapsulated in the *move-to* behavior that is invoked by many different skills. This architecture closely resembles that of the Panda Behavior Tree engine used in Unity.

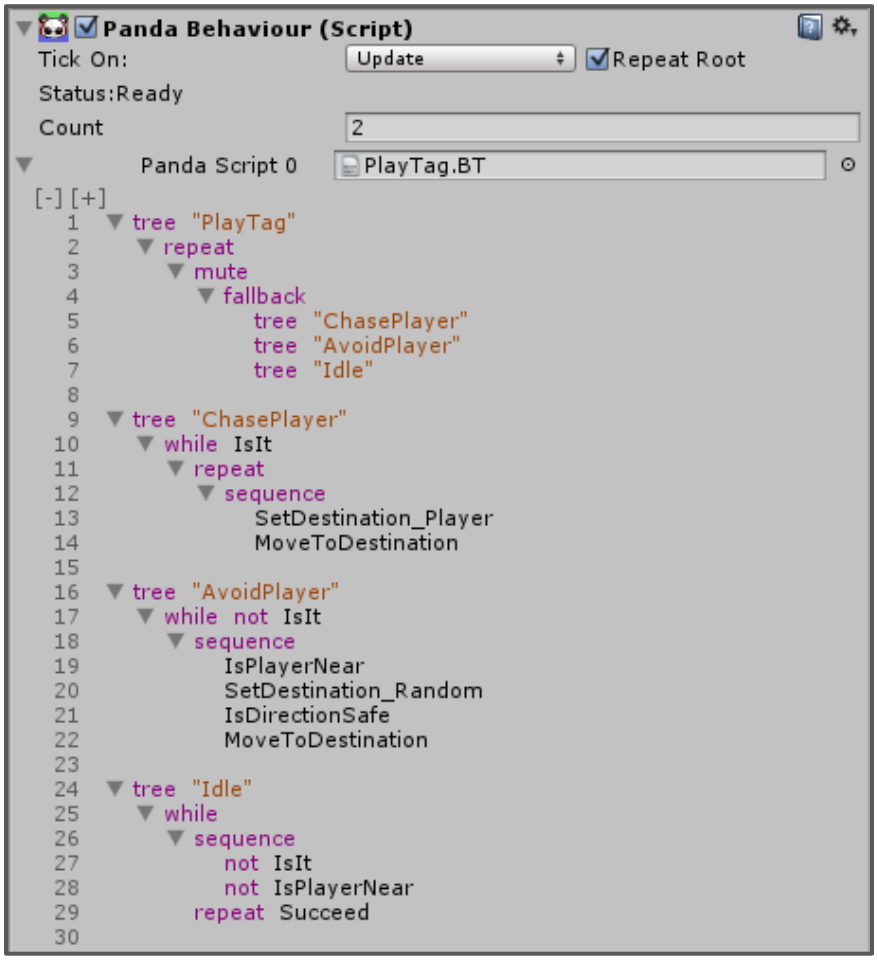

Figure 1.0 shows how the Panda Behaviour Tree is implemented in Unity

Skills make the decisions and decide what to do based on the motivations and capabilities of the character, as well as the current state of the environment. Skills answer questions like *Do I want to attack, hide or flee?* and *What is the best place for me to be?* Once a decision is made, behaviors are invoked to implement it. For example, if movement is required, the skill may invoke the *move-to* behavior and then wait for it to succeed or fail. The *move-to* behavior attempts to reach the destination using whatever capabilities are available to it. It answers questions like *Which route should I take?* and *Which animations should I play?* It generates paths, avoids obstacles, selects animations to traverse the environment and ultimately reports the results to the parent skill.

Both skills and behaviors are implemented as straightforward finite-state machines. Skills are tailored to specific character types, while behaviors are more widely used. Each type of character maintains a prioritized list of skills and these skills run until they are completed or are interrupted by a higher priority skill.

Agent Response to Weapons

The Last of Us equip the player with many weapons that are useful in killing enemies.

Molotov Cocktail's

Molotov Cocktail's can be thrown, and all nearby enemies will run to inspect the noise, only to be engulfed in flames and eventually die. The developers limited the number of characters that could be affected by the flames, so that not all the enemies responding to the noise would die, making the game slightly more challenging.

Figure 2.0 shows the main character, Joel, throwing a Molotov Cocktail at the Infected, who are then engulfed in flames.

Smoke Bombs

Smoke Bombs can also be used to distort enemy vision and hearing. They break line of sight and mask the movement of the player. Once an enemy is attracted by the detonation of a smoke bomb and enter the haze, they are effectively deaf and blind.

Figure 3.0 shows the main character, Joel, crafting a smoke bomb to use against enemies.

Enemies ('The Infected')

What are Infected?

Infected in The Last of Us are humans who have succumbed to the parasitic Cordyceps fungus. The pandemic corrupts their minds and leads them to being disfigured and overly aggressive. The Infected are driven only by the instincts of the fungus and have no trace of personality, compassion or self-preservation. In contrast to the Hunter, the Infected appear chaotic and alien.

Figure 4.0 shows the Infected chasing the main character.

Overall Design

All the Infected character types share a single C++ class and are differentiated only by the set of skills and values of the tuning variables in their data files. For example, the code refers to the *vision type* of the character instead of testing if the character is a Runner or a Clicker. This is a very generic way of coding and allows for easy adding and removing of skills, behaviors and tuning variables, without the need to create new characters every time. Another advantage of this design is that allows for straightforward configuration of the difficulty settings for each character type individually, by simply adjusting the thresholds by which Infected would respond to stimuli.

Senses and Behaviours

```
class Sense {
    //This class contains the tuning variables,
    //responsible for adjusting each sense
public:
    enum VisionType { LIMITED, BLIND };
    enum SpeedType { SLOW, MEDIUM, FAST };
    enum RarityType { COMMON, UNCOMMON, RARE };
    enum CombatType { GROUPED, AMBUSH, MELEE, RANGED, ARMORED };

    Sense(VisionType, SpeedType, RarityType, CombatType)
    {

    }
};


class Behaviour {

    //Assigns a behaviour (call-back function) to a specific skill, can be overridden in derived classes
public:
    void AddBehavior(std::function<void()> behaviourCallback)
    {
        //Add the behaviour to the list of call-backs

    }

    //static container located in this class storing callback functions
    //We should use a map or unordered map here and have a priority order as well, this is just an example.
private:
    static std::vector<std::function<void()> > m_behaviourCallbacks;

};
```

Figure 5.0 shows a simple pseudo-code implementation on the Sense and Behaviour classes.

Skills

```
class Skill {

    //Assigns a behaviour to the skill, a wrapper function
protected:
    void AssignBehaviour(std::function<void()> behaviourCallback)
    {
        m_behaviour.AddBehavior(behaviourCallback);
    }

private:
    //Every skill has a behaviour
    Behaviour m_behaviour;
};

class OnFire : Skill {

    //Assign Behaviour to this Skill upon creation
public:
    OnFire()
    {
        //We need to use std::bind and placemarkers for this to work correctly.
        AssignBehaviour(&OnFire::OnFireDo);
    }

    //Just an example, this could be further encapsulated, maybe have an OnFire behaviour deriving from the Behaviour class as well.

private:
    void OnFireDo()
    {
        //When on fire is active, do the following
        //Update positions
        //Update animations
        //Update audio, etc.
    }
};

class Wander : Skill {
```

Figure 6.0 demonstrates how the Skill class could potentially be implemented.

Infected

```
class Infected {

    bool Create(const std::string& dataFile)
    {
        //Load in the tuning variables and skills from the data file, for the Infected in this specific part of the game

        //File::Load("infected_Area1");
        //Realisitically, we'd use a serialization library such as Cereal for this

        //Store all the senses data for this infected type in the container
        //E.g.
        m_senses.emplace_back(new Sense(Sense::VisionType::BLIND,
                                        Sense::SpeedType::SLOW,
                                        Sense::RarityType::COMMON,
                                        Sense::CombatType::AMBUSH));

        //This would be encased in some kind of loop so we can add all the data to the container in one go

        //Repeat for the skills
        m_skills.emplace_back(new OnFire());
    }

    std::vector<Sense*> m_senses;
    std::vector<Skill*> m_skills;
};
```

Figure 7.0 demonstrates how the Infected class could be implemented.

Interaction

The Infected rely primarily on hearing. Logical sound events are used to allow the player to understand and predict reactions of the Infected. The sounds are broadcast over a radius and any character within that radius can hear it. Logical sounds are also partially occluded by walls and obstacles. Each time a logic sound event is broadcast, rays are cast from each character within the sound radius to the source of the sound to determine the level of occlusion. The logical sound is broadcast to all characters in range that are not completely occluded.

On the flip side, the Infected do not hear as well when they are unaware of the player, which makes it easier for the player to be stealthy. The Infected are difficult to sneak up on. To make the Infected feel and appear more dangerous, the developers scaled the broadcast radius of logical movement sounds with the speed of the player. This allows for the player to approach the Infected quickly from farther away but requires the player to move slowly at melee range. To make the player aware of this, the Infected enter an agitated state when they start to hear a noise, which gives the player a chance to respond before being discovered.

Figure 8.0 shows the player in 'Listen Mode'. This allows them to see through walls and obstacles and hunt out any enemies.

Infected only react to stimuli. For example, if the player throws a bottle, they are drawn to the sound of the bottle landing as opposed to deducing the distance of the character that threw it.

Infected Types

Figure 9.0 shows the different types of Infected and how they appear.

Runner

Runner's are the most common type of Infected. They are fast and often attack in uncoordinated groups. They can see and have advanced hearing.

Stalker

Stalker's are similar to Runner's but hide in dark areas and ambush prey.

Clicker

Clicker's are visibly disfigured by the infection, with masses of fungus distorting their features and leaving them blind. They have developed a type of echolocation to compensate. They are slower than Runner's but have a deadly frenzy attack and ignore melee attacks that don't use weapons.

Bloater

Bloater's are highly disfigured, blind, slow and heavily armored. They throw growths from their bodies that burst into disorienting clouds of dust that slow down the player.

Characteristics of Infected Character Types

The table below shows the overall senses of each character type.

Type	**Runner**	**Stalker**	**Clicker**	**Bloater**
Speed	Fast	Fast	Medium	Slow
Vision	Limited	Limited	Blind	Blind
Rarity	Common	Uncommon	Uncommon	Rare
Combat	Attack in groups	Ambush in dark areas	Melee frenzy, limited melee vulnerability	Armored, ranged attack, invulnerable to melee

Infected Skills

Prioritized Skills for Infected Character Types

The table below contains the skills used by each type of Infected, sorted by priority.

Runner	**Stalker**	**Clicker**	**Bloater**
On-fire	On-fire	On-fire	On-fire
Chase	Ambush	Chase	Throw
Search	Sleep	Search	Chase
Follow	Wander	Follow	Search
Sleep		Sleep	Follow
Wander		Wander	Sleep
			Wander

On-Fire Skill

The on-fire skill works by invoking the *infected-canvass* behavior briefly in a small area, with a custom set of animations. This causes the character to flail and dart around chaotically in an arbitrary environment without running into walls.

This effect can be achieved in Unity by using an Animation Curve to alter the characters position, whilst playing a custom animation.

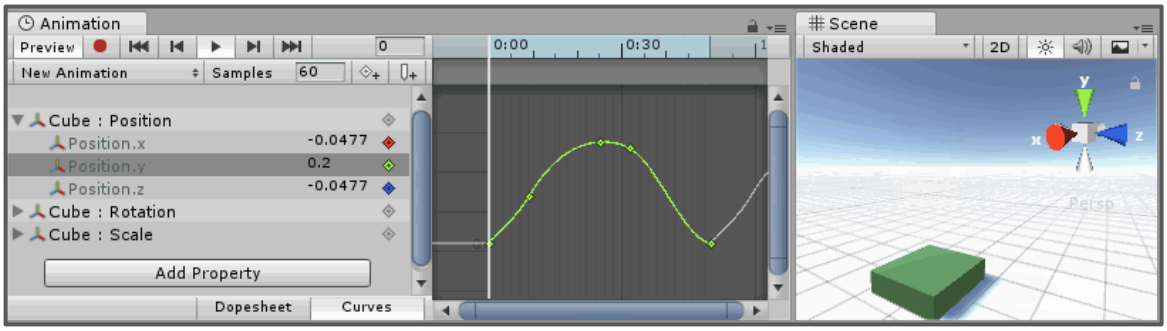

Figure 10.0 demonstrates how the On-Fire skill can be achieved in Unity, using the Animation tool.

Chase Skill

When an entity is first encountered, the Infected turns toward the stimulus and screams to alter the player that they have been discovered. The chase skill does this by invoking the *surprise* behavior that selects from a set of animations that orient the character in the appropriate direction. Next, the chase skill invokes the *move-to* behaviour, which provides an interface to the navigation system and is responsible for moving the character to a location or entity. As the entity moves through the environment, the *move-to* behaviour continuously updates the path, leaving the parent skill to simply monitor for success or failure.

As there is no navigation mesh available for 2D games in Unity, we must find another, simpler way of mimicking this skill. We must first let the player know they have been discovered, either via sound or

some other trigger. We would then orient the enemy towards the target (player) and then chase the target. I would use the Arrive steering behaviour to accomplish this, making sure the enemy arrives *next to* the target.

Figure 11.0 shows the logic behind my implementation of the Chase skill.

If we wanted to make it more realistic, we could make the enemy pause or move around a small area briefly to give the appearance that the enemy is searching for the target (even though we are still aware of the target's presence).

Search Skill

This skill is active when an Infected loses track of the player during a chase. The search is not exhaustive, they eventually lose interest and resume wandering around. A navigation mesh is again used for this skill. This navigation mesh contains 'hiding places', which are polygons on the navigation mesh that are not visible from other parts of the navigation mesh. When the search begins, search points are generated to reveal hiding places. A search point is added at the center of the polygon containing the last known location of the target. A breadth-first search is performed, which visits neighboring polygons until line of sight to the search point is broken. A new search point is added at the center of the parent polygon (i.e., the last polygon with line of sight to the previous search point). This process continues until the entire search area is visited or a fixed number of search points have been added. The result is a graph of points with visibility to each other and to all the hiding places around them. When an Infected reaches a search point, it briefly investigates before moving on.

In Unity, a simple way of achieving a similar effect would be to develop a waypoint system. The enemy would explore each waypoint ('hiding places') either randomly or in order, changing direction every so often to increase unpredictability. The waypoint system could have a specific cut off point and once the player is a certain distance away from this cut off point the enemy resumes wandering and stops searching.

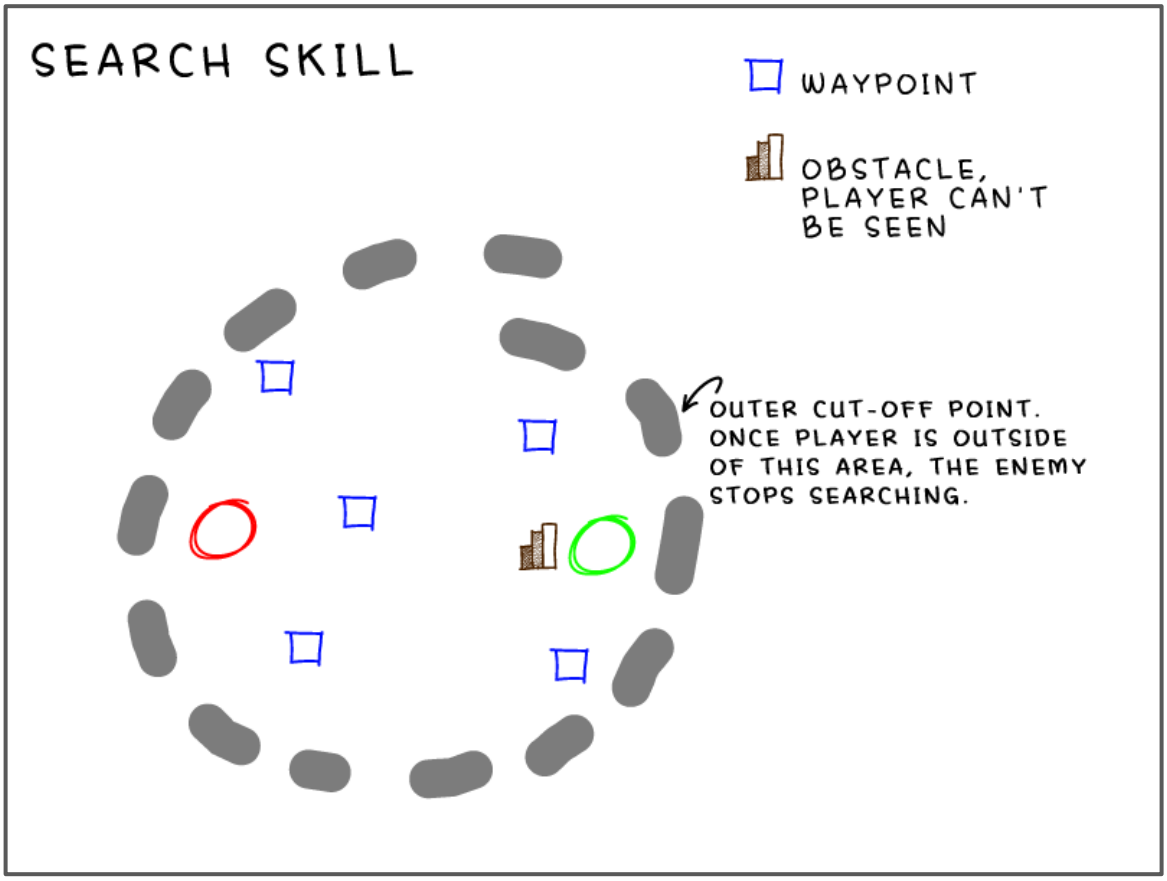

Figure 12.0 the logic behind my waypoint system, a simple way of re-creating the Search skill.

Follow Skill

The Infected do not communicate with each other. Consequently, if an Infected senses the target and starts to chase, any nearby Infected that are not directly nearby are oblivious. The Follow skill was implemented to allow one character to follow another that is chasing something. The following character does not receive any information about the target being chased; it just has the compulsion to follow along. Conceptually, the Infected seek the opportunity to claim the unknown prey for themselves. As a result, when an Infected is alerted and starts chasing the target, it may pick up others along the way, but as the player expects, the alerted character will be the first to arrive.

We can achieve a similar effect in Unity by using a Finite State Machine (FSM). Fortunately, Unity provides us with this through the Animator tool. If an enemy is in the Follow state, all close-by enemies will precede to follow the target, without knowing what it is.

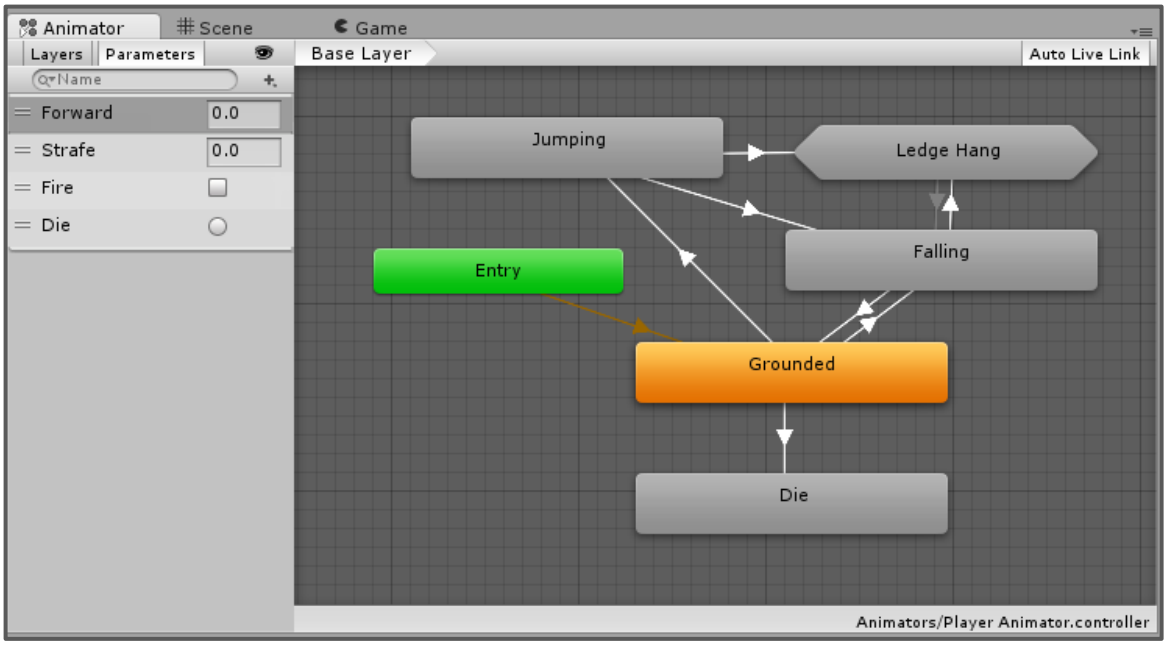

Figure 13.0 shows the Animator interface in Unity. The states would be altered to our needs.

Sleep Skill

When the Sleep skill is in use, the Infected's senses are drastically reduced, giving the player opportunity to avoid, distract or dispatch the character more easily. The player must exercise caution when Infected are using this skill. If the player moves too quickly or too near a sleeping Infected, it will stir briefly before settling back into slumber. If the player makes a loud noise from a great enough distance, the Infected will wake up and search a small area, before going back to sleep. If the player continues to disturb the character or is sufficiently loud, it will wake enraged and Chase the disturbance. When the chase ends the Infected will wander through the environment instead of going back to sleep.

Using a Finite State Machine (see Figure 13.0) we could achieve the above simply. The character would need two states – Sleep and Wander. Upon game startup, their initial state is selected at random, some Infected will sleep, and some will be wandering around. The sleep state will contain a variable which stores the distance between the character and the player. Using this distance variable, we can determine how close the player is to Infected and if the player get's too close, the Infected will react slightly. In terms of a 2D game, this could be shown by simply changing the animation of the Infected, to give the impression they are stirring slightly. We would also have a bounding box surrounding the Infected, which acts as a type of trigger. If the player enters this bounding box at a certain speed, the Infected stir once again. The same principle could be used to allow Infected to detect loud noises within earshot.

Wander Skill

This is the lowest priority skill for all Infected types. It is the fallback that is used when nothing else is valid to run. When Infected are not agitated, they will move throughout the environment either randomly or in predictable patterns. Fixed routes are accomplished by laying out a spline. Random wandering selects a random polygon on the navigation mesh as a destination. As the character moves, it keeps track of the polygons it visits. On arrival, it randomly selects a polygon that hasn't been visited and continues to wander. This has the effect of covering a large area in an unpredictable way.

In terms of creating a spline, there are numerous free assets that can achieve this in Unity. For example, the Easy Spline Path 2D asset. This would be useful if you were making a top-down game. Otherwise, a similar effect can be accomplished by setting fixed waypoints in your level (see Figure 12.0). For random wandering, we could generate the waypoints randomly at level startup. This also increases re-playability value, as every time you start a new game the characters change position.

Ambush Skill

This is the only Infected-specific skill that uses cover and is only used by the Stalkers. Cover locations are selected using a system that evaluates points in the environment based on how suitable they are to ambush the player or to retreat after an attack. The Stalkers will lay in wait until the player wanders too close, then attack and run back into cover.

A similar effect can be achieved in Unity by having invisible 'safe-zones' for the Stalkers. This could be as simple as having an array of safe locations the characters can run to after attacking the player. For example, behind a wall or a specific object.

Throw Skill

The Bloater is the only character with a projectile attack. The Bloater rips and throws the fungal growth from its own body ahead of the moving player. The fungal growths produce a cloud of particles that briefly slow player movement.

In Unity this can be achieved by attaching an invisible bounding box around the particles. If this bounding box contacts the player, then the player's speed is reduced. The player could enter a 'Slow' state, meaning their speed is reduced for a set period. Whilst in the Slow state, any further contact with the particles would be ignored.

Conclusion

In this report I demonstrated how the artificial intelligence is implemented in The Last of Us. I covered how this could be achieved in Unity and discussed which steps I would take and which assets I would use to re-create this type of AI in my own game. I demonstrated my understanding of how the artificial intelligence is implemented, by providing diagrams and explaining how I would convert the 3D artificial intelligence into 2D artificial intelligence. I have concluded that The Last of Us uses many interesting technique's that are not unfamiliar to me, such as Finite State Machines, dependency injection and behaviour trees and I have covered how I would implement these either through code or using Unity.

References

Begue, Eric, Panda Behaviour Tree, Feb 19th 2019, Panda BT

Bevilacqua, Fernando, Understanding Steering Behaviors: Flee and Arrival, Oct 26th 2012, Game Development - Flee and Arrival

Botta, Mark, Infected AI in The Last of Us, April 2015, Game AI Pro 2

DC_Assets_Uy, Easy Spline Path 2D, Feb 1st 2019, Unity Asset Store - Easy Spline Path 2D

Jb-dev, Steering Behaviors: Seeking and Arriving, May 10th 2018, GameDev - Seeking and Arriving

Unity, Animator Scripting – Unity Official Tutorials, Feb 2nd 2014, Unity - Animator Scripting

Unity, The Animator Controller Asset, April 15th 2019, Unity - Animator

Won, Daniel, Waypoint Pro 2D, Jan 5th 2016, Unity Asset Store - Waypoint Pro 2D

Zotov, Alexander, How to Create Simple Waypoint System for 2D Unity Game? Simple Tutorial, Nov 24th 2017, YouTube - Create Waypoint System