GAME AI REPORT

1. Story

In 7017, the world has come to an end. Human race has been pushed to the brink of extinction by an unknown army from outer space. City after city with full of population had been wipe out in a blink of eyes. Human has lost the war! There is no army can stop the assault of the aliens.

However, another war is still raging on, the war for survival. In order to safe keep mankind's future, a group of extraordinary scientists has created two weapons that can turn the tide of this survival war: a special suit that can expel any alien come close to the wearer and a portal that lead people to another dimension, out of the alien's reach. Unfortunately, there is not enough resource to mass produce the suit and the portal, they can only make one suit and one portal at this time. And the Leader, a veteran sergeant, is the first human to ever try on the suit. He will lead the people and protect them until they reach the portal.

2. General Gameplay

Player controls the Leader by using any basic mouse and keyboard movements on a top-down view camera. The main objective of this game is to guide people, "the Sheep", to the safety of the portal.

However, the enemies, "the Hunter", will try to stop the Leader from guiding his people. They roam around the map, search for their preys and try to kill them as soon as possible. The Hunter acts and hunts as a group when they acquire their target. In contrast, the Sheep forms flocks and usually follow the Leader around.

Because of the suit, the Hunter tends to stay away from the Leader as far as possible when they see him. Player can use this to their advantage in order to stop the Hunter from eating or chasing the Sheep.

The map is generated randomly each time the game is fired.

3. Agent Characteristics

The Sheep: they act similar to mindless drones and can only follow the Leader. When they see the Hunter, they will run away from it. If they can not find the Leader, they will run mindlessly. Each Sheep has a panic and courage levels.

Whenever a Sheep see a Hunter coming close to them, its panic level is increased and it will start to run away from the Hunter. However, the increase rate is based on its courage level. If it has high courage level, the increase rate is low and it is hard to fall into a seizure state. Whereas, with low courage level, the Sheep is easy to become crazy and can not follow the Leader.

Seizure state is a special characteristic of the Sheep. In this state, it moves with its slowest speed and does not follow the Leader. If it does not see any Hunters nearby, it will be back to normal shortly.

The Hunter: they roam around the area and search for their prey. They act as a group. If a Hunter has the highest leader level in a group of Hunters around it, it can give out command for the rest. If the Hunter has the lowest leader level, it will receive command from the Leader to hunt or will hunt alone when it sees a prey.

The Hunter's leader level is changed dynamically through the game. If the Hunter can not find any preys or receives command from other Hunters, its leader level will be decreased. If it can give out command or spot a prey, its leader level will be increased.

There is another level to indicate how hard the Hunter will try to hunt for its prey which is named Ferocity. Ferocity level is given out randomly for each Hunter similar to the leader level. If a Hunter has high Ferocity, it will try to hunt and seek for its prey longer than a low Ferocity level Hunter. Sometimes, because its Ferocity is so high, the Hunter can also ignore the expel force of the Leader's suit in order to catch its prey.

The Hunter stops hunting only when it can not find its prey anymore. At this time, it will roam around and pick another prey.

4. Finite State Machine

Our state machine contains three main parts which are Enter, Run and Exit and is managed by a Finite State Machine Logic Editor. Enter is called at the beginning of the state. Run is called in every frame when the state is running. Exit is called when the state is changed by the state machine in order to give place for another different state.

In order to signal for the state machine to change to different states, each state contains several ExplicitStateReference links acting as references to different states in the game object's state machines. That means, in every frame, the state machine will check for the condition to request the next state transition. However, in the next state transition, the state machine does not decide which state the game object should be in hence the implementation of the ExplicitStateReference links.

In each request for the next state transition, a state link is attached to the request in order to guide the state machine to change the game object's state to another state.

- **The finite state machine for the Sheep has four distinct states**

  + Sheep_roaming: This is the starting state (idle) of any Sheep. At the beginning of this state, a set of steering behaviours will be applied to each Sheep such as ArriveBehaviour and SeekBehaviour. Its courage level is also set randomly and its panic level is set at 0.

  In this state, the Sheep roams around the area. For each frame, the state will check if they can see the Leader or not. If they see the Leader, they will follow him using the ArriveBehaviour. If they can not see him anymore, they will seek for him at his latest position they saw by using the SeekBehaviour.

When a Sheep sees a Hunter, its panic level will begin to increase based on its courage level. When the panic level is increased to a specific threshold which is larger than 7, the Sheep's finite state machine will change to another state which is Sheep_running.

+ Sheep_running: This state is the same as Sheep_roaming state. The Sheep will try to follow the Leader whenever it spots him. However, its speed is increased in order to help it run away from the Hunter. At the beginning of this state, another steering behaviour, which is FleeBehaviour, is set for the Sheep to make it run away from the Hunter.

In each frame, the state will check if the Sheep can see the Hunter or not. If it can still see the Hunter, its panic level will be increased. If it loses sight of the Hunter, its panic level will be decreased and has a high chance to go back to its roaming state.

If its panic level is higher than normal threshold which is larger than 25, its state will be changed to Sheep_gonenut state.

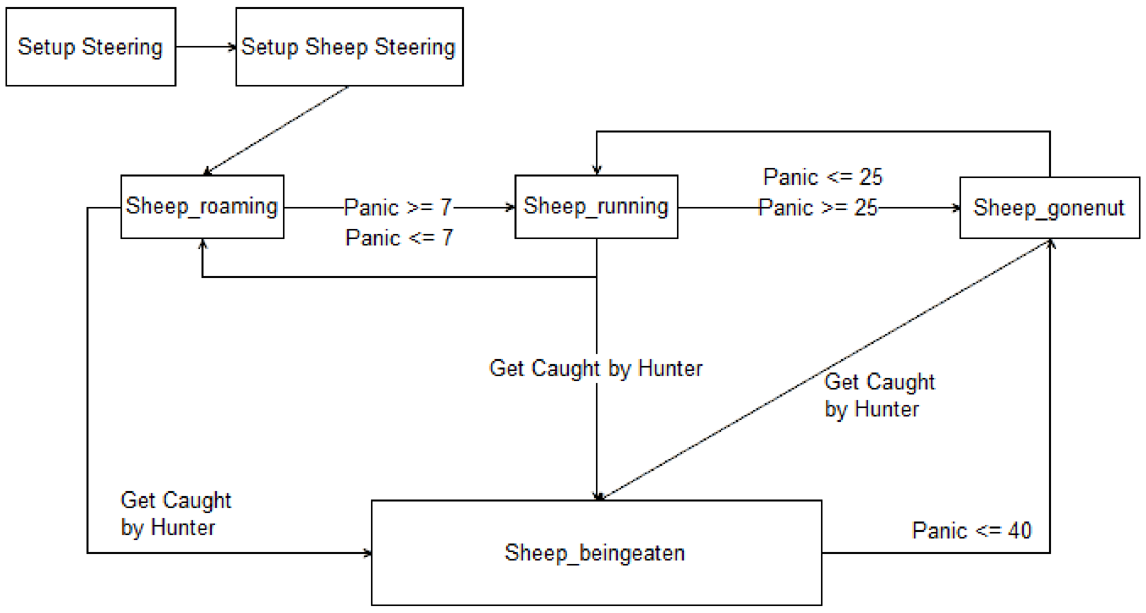
+ Sheep_gonenut: In this state, the Sheep will move with its slowest speed and will be waiting for being caught by the Hunter.

The Sheep can only recover from this state when its panic level is decreased to the lowest. That means, if the Sheep loses sight of the Hunter, its panic level will wait for an amount of time then decrease. The decreasing rate can be improved if the Sheep knows the Leader nearby its area.

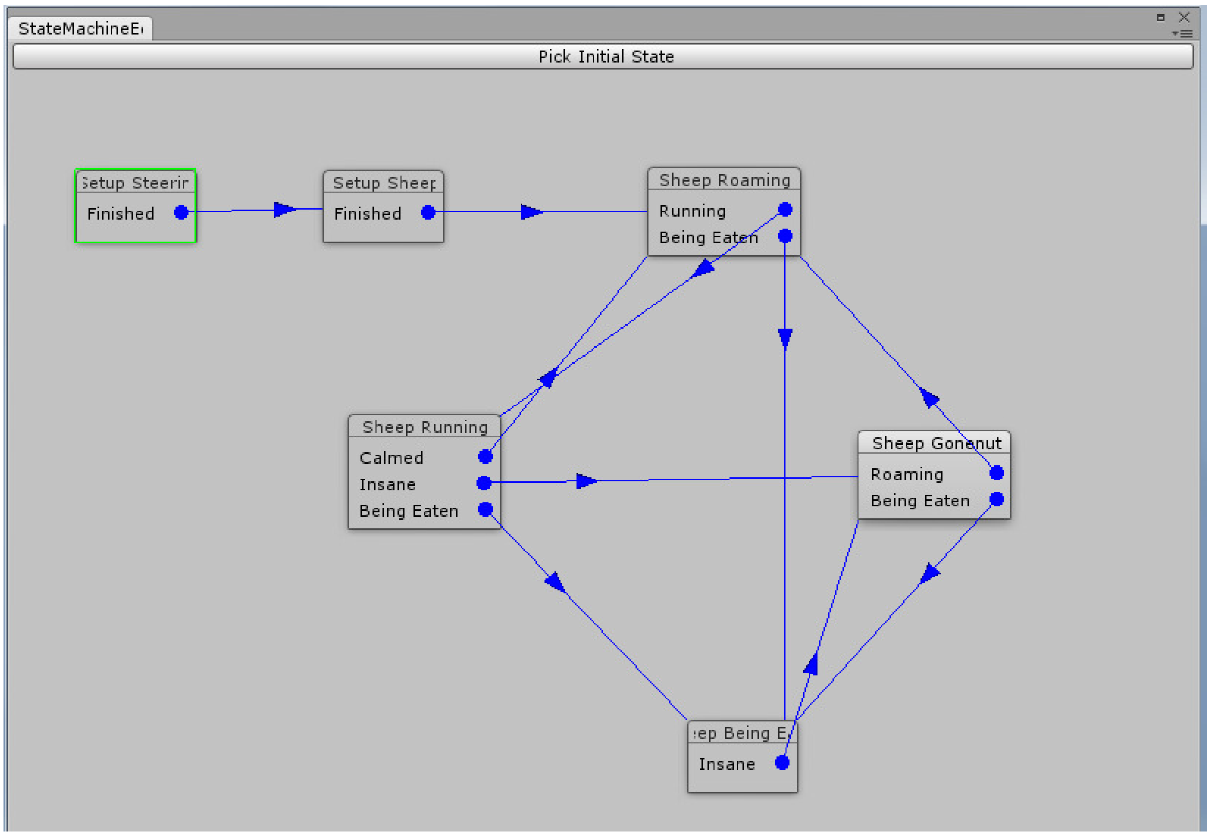
With its panic level decreased to the lowest, the Sheep is back to its roaming state.

+ **Sheep_beingeaten:** This is a special state for each Sheep. This state only occurs when the Sheep gets caught by its Hunter. At the beginning of this state, its panic level is automatically set to 55.

In each frame, the panic level will be increased until it reaches its highest threshold which is 70. At this time, the Sheep has lost its will to live. It will be eaten by the Hunter. However, if the Sheep spots the Leader in its area, its panic level will be decreased until it reaches 40. At that time, the Sheep's state will be changed to Sheep_gonenut state.



Sheep's state machine



Sheep's FSM Logic Editor

- **The finite state machine for the Hunter has three distinct states**

+ **Wolf_roaming**: This is the starting state (idle) of any Hunters, the Hunter roams around and searches for its prey. At the beginning of this state, leader level and Ferocity are generated randomly for each Hunter. Flee steering behaviour is also added to direct the Hunter away from where the Leader is.

In each frame, the state firstly checks if the Hunter has received any commands from other Hunters to hunt for a specific prey or not. If it is yes, the state will be changed to Wolf_hunting state. If it is no, the Hunter will roam freely and seek for its prey. When it spots a prey, it will change to Wolf_hunting state.

However, before the state is changed, it will check if it has the highest leader level in the group of Hunters around it. If it is the leader, it will give out command to hunt for a prey. If it is not the leader, it will hunt alone.

+ **Wolf_hunting:** In this state, the Hunter's running speed will be increased in order to chase down its prey. Seek steering behaviour is also added to help the Hunter seek for its prey when the prey goes out of its sight.

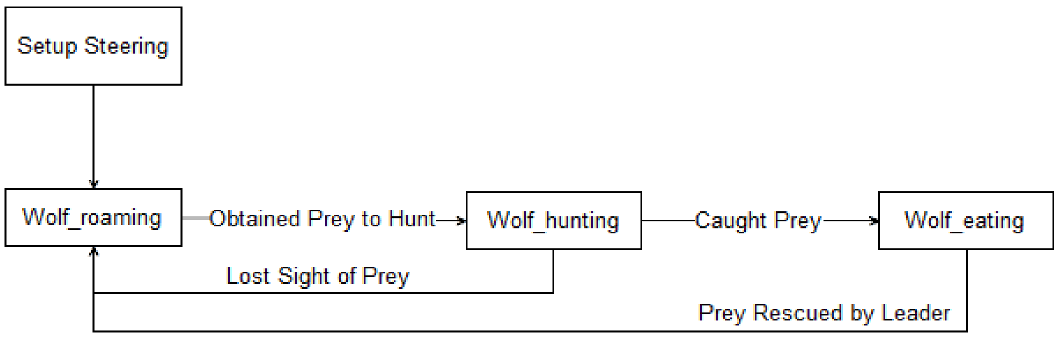
In each frame, the state checks if the Hunter still has vision of its prey or not. If it is yes, the Hunter will continue chasing its prey. If it is no, the Hunter will seek at the prey's last position in an amount of time based on its Ferocity level. If the Hunter still can not find its prey, the state will be changed back to roaming state.

If the Hunter catches its prey, the state will be changed to a special state named Wolf_eating state.

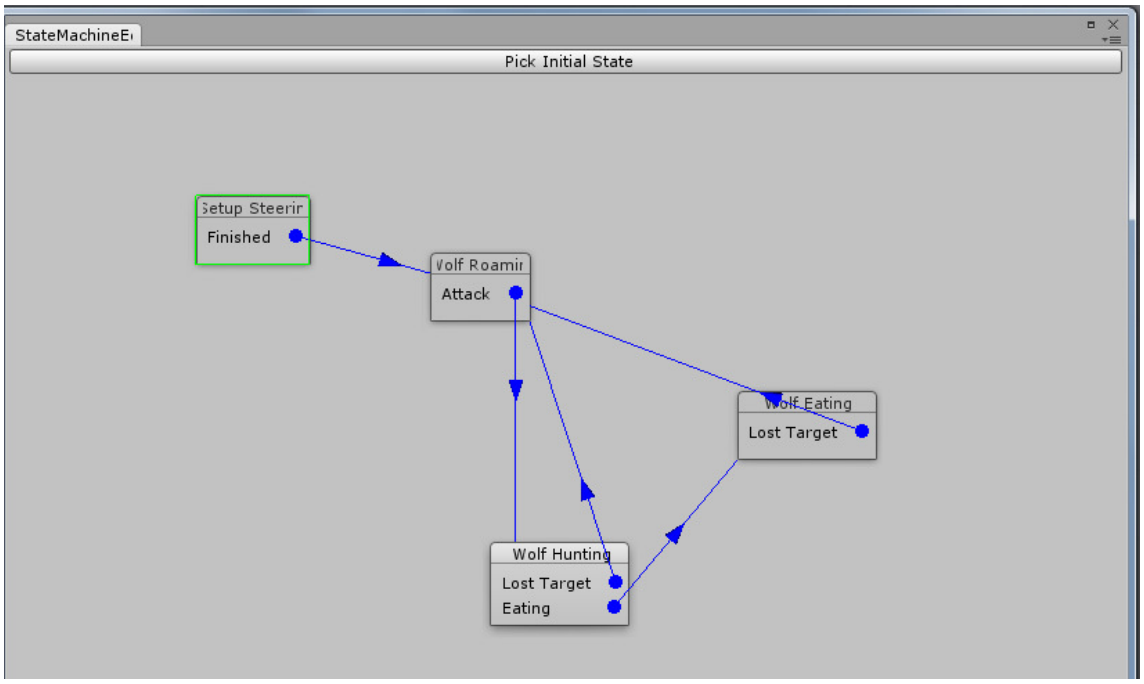
+ **Wolf_eating:** In this state, the Hunter stops moving. It is eating the Sheep.

In each frame, the state will check if the Leader is nearby or not. If yes, the Hunter will move away slowly from its prey and give time for the prey to recover from its panic. If no, the Hunter will continue eating the prey. When the prey's panic level is increased to the maximum of 70, the Hunter finishes eating its prey.

At this time, the state machine will trigger roaming state in order to replace Wolf_eating state.



Hunter's state machine



Hunter's FSM Logic Editor

SetupSteering state

A special state used to setup everything related to steering behaviour for each game object such as separation, randomwalk and boxavoidance. Therefore, the objects will move randomly around the map, stay away from the building and stop getting stuck into each other.

SetupSheepSteering state

A special state used to setup everything related to the sheep and their steering behaviour such as cohesion and alignment in order for them to form a flock around each other or the player.

5. Steering Behaviour

- **How the algorithm works**
  - At its most fundamental level, steering behaviours output a desired velocity as a Vector which is then applied to the object to be steered.
  - All steering behaviours have 2 weights, an internal one and an external one. Every time the desired velocity is requested, beforehand the internal weight is set to `1.0`, which can then be changed by the steering behaviour, making it stronger or weaker as desired. In our project it is normally used as an 'off' switch. The external weight is set to determine the importance of a steering behaviour. In collections of steering behaviours, the weights combined are used to determine the strength of each behaviour.
  - An example of this in action is the wall avoidance behaviour. Its external weight is set to `100.0`, much larger than the default of `1.0`, meaning that in practice it takes precedence over all other behaviours, which is important as to ensure collisions with walls don't take place. However, if an agent is far enough away from a wall, the

    - internal weight is set to 0, removing any effect on other steering behaviours, regardless of the external weight, which remains unchanged.
    - Complex steering behaviours such as flocking are created by combining simpler behaviours such as cohesion, seeking, separation and alignment.
- **Choices you made that are specific to your project**
    - Steering behaviours are added to a generic 'Legs' object, this also calculates how strong each steering behaviours should be depending on their weighting. The Legs object also controls all movement in general, so it was the logical place to deal with steering behaviours.
    - External weights are set by States in the Finite State Machine. This is important as it means that individual states can prioritise different behaviours. For example in the wandering state, a sheep may set its Flee behaviours weight to `0.0`, while when in the running state it may be set to `10.0`.
    - Steering behaviours are categorized as either targetable or a group member behaviour. Targetable behaviours target a certain coordinate, while group member behaviours are passed some nearby objects of a given type which can be used to control group defined behaviour such as flocking. These 2 types have been abstracted in our implementation, making it easier for new behaviours to be created. Examples of targetable behaviours are Seek, Arrive and Pathfind, while Group behaviours include Cohesion, Alignment and Separation.

- **Any deviations from textbook algorithms you made, and why.**
    - The textbook mentions nothing of an internal weight. This was implemented as a way for steering behaviours to tell the legs that any output they make is pointless and should be ignored, or conversely, extremely important and must have priority over other behaviours. Implementing this using only a single weight would have been difficult as the weight as set by the user should be the default, and must be saved and restored when using a single weight value.
    - The textbook when defining path following steering behaviours involves ensuring the agent is a certain distance from the path and follows a point ahead of the object. Since we're using a grid, we decided to simply make sure the agent visits each grid square along the path in order.

6. A* Pathfinding Algorithm

- **How the algorithm works**
    1. Initialise a map to store any A* nodes.
    2. Initialise a map of Booleans as the closed set
    3. Initialise a minimum priority queue for the open set
    4. Add the current position to the open set
    5. While there is still a node in the open set
    6. Mark the node as processed in the closed set and remove it from the open set.
    7. If the current node is the target node, exit.
    8. For every neighbour of the node, skip it if it's in the closed set, otherwise if it is in the open set, update it if the current path to it is shorter than the previous one, and if not in the open set enqueue to the open set (Ordered by F-Score). Set the neighbours parent to the current node.
- I used the Manhattan distance as the heuristic because of the cityscape environment of this game. since there are only effectively 4 possible directions of movement, the Manhattan

distance is the lowest possible distance between two points on the grid. Since we are looking for the smallest distance path, this is an admissible heuristic, as it will never be greater than the actual path length.

- The algorithm piggybacks off the Grid data structure used to generate the city and maze. AStarNodes are little more than references to squares of the Grid with an F-Score and parent attached.
- When I first tested my algorithm it appeared as though nothing was working, it turned out when I was converting Unity Vector3 to grid coordinates that I casted the fraction of the Vectors position relative to the grid to an int prematurely (converting it to 0), meaning when multiplied by the size of the grid it always returned 0. The first segment of the path was never showing up, it was because I was checking if its parent was null before adding it to the path in the path construction phase.
- My closed set is simply an array of bools, one for each possible grid square. This was done since the algorithm is run multiple times every second by nearly every actor, so a constant time lookup was favourable.

7. Contribution

- Kieren David Wallace (s3235426): FSM logic editor, groundwork such as developing senses (Eyes, Ears, Hearable,...), designing the finite state machine interface and making the map, programming player behaviour, fixing bugs for State Machine and Steering Behaviour.

- Andrew Dunn (s3332747): making the map, developing steering behaviours in game and A* pathfinding.

- Bao Luong (s3272297): developing finite states for Sheep and Wolf, organising the report file.