

# University of Portsmouth

## Computer Games Technology

### CT6GAMAI

### AI Report

### UP784120

### Word count:1937

#### Introduction:

This AI project was developed having in mind the idea of a stealth game. One of the most important stealth game is represented by Assassin's Creed Series so some features are inspired from this game series.

It is considered that the heart of stealth games is represented by a simple game loop between two cognitive sense, sneaking and fleeing. (Wong, 2014, para 2) So, in order for this two to happen the AI will need to be able to move around the map, react to player's input and change it's behavior based on different events that happen in the game. The artefact will not only have a hostile AI, that will have the role of attacking the player, but crowds will also be implemented. Crowds will have the role of creating a more real feeling to the artefact.

Rasmussen argues that the state of the art in AI games calls for AI technologies that can support complex, deep and immersive game worlds. He gives example of the Assassin's Creed series where the number of the non-player characters is growing and the demands of life-like behaviors are increasing. (Rasmussen, 2016, para 9)

Both crowds and the hostile AI will require a state management system that will change their actions based on some conditions. Also their movement around the map will be done by using a Pathfinding algorithm. Crowds will also require Steering Behaviors in order to make their movement look more natural.

The hostile AI will be limited in terms of visual and auditory perception. It will be able to see the player only if he is in a certain range and in the FOV and it will react to sounds made by the player.

## Analysis:

### Finite State Machines:

I did choose to use Finite State Machines in order to organise the AI's behaviors because the AI has to react to different events and inputs based on a certain context. On the other hand, a goal driven approach might have been worked well for an agent that need to satisfy a particular goal that is formed from multiple subgoals. For very simple agents who do not perform lots of tasks FSM are considered to be suffice because the goal driven approach might be a little heavy-handed(Owens , 2014, para 15).

The FSM was developed using Scriptable Objects and the polymorphism concept of object oriented programming. Scriptable Objects are serializable Unity classes that allow the developer to store a large amount of shared data independent from script instances. The advantages of scriptable object is that it makes it easier to manage changes and debugging (Unity, 2018,para 2). Going back to polymorphism, this concepts gives the ability to overload functions in order to change the behavior or the state of the AI. ("Polymorphism in C++", n.d)

There are some important rules that need to be followed while developing FSM: the set of state should be fixed, the machine can only be in one state at a time, a sequence of inputs or events is ent to the machine and each state has a set of transitions, each associated with an input and pointing to a state. ("State", n.d)

The FSM has two base classes, Action and Decision. The Action class is used to create new classes that represent AI's action, while the Decision class is used to create new classes that represents AI's decision based on which it changes its state. Furthermore, States are formed from transitions that contain a true and a false state based on a certain decision and also in has a number of actions attached. Each State, Action, and Decision was saved as scriptable object.

The Hostile AI in the artefact has five main states:

- Patrol : AI patrols between preset waypoints
- Chase Noise: AI Chases to the sound source
- Chase Player : AI chases the player if he is in the view range
- Get Ammo: the AI will go to the closest Ammo pack if the ammo level is low
- Get Health: the AI will go to the closest Health pack if the health level is low

The Crowds AI have only one state: following the leader, the leader being able to patrol between set waypoints as the hostile AI.

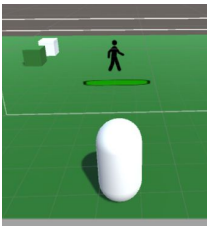
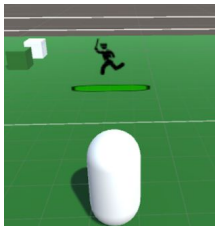
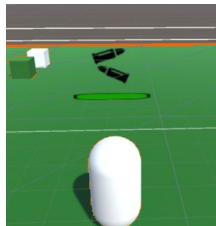
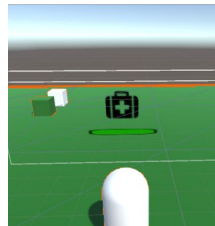
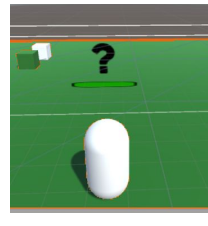
Figure 1 shows the AI with the Patrol icon on	Figure 2 shows the AI with the Chase icon on	Figure 3 shows the AI with the Ammo reload icon on	Figure 4 shows the AI with the Health Regeneration icon on	Figure 5 shows the AI with the Chase Noise icon on
				
Figure 1. Patrol [Primary Source]	Figure 2. Chase [Primary Source]	Figure 3. Ammo reload [Primary Source]	Figure 4. Health Regeneration [Primary Source]	Figure 5. Chase Noise [Primary Source]

Figure 6 shows AI's states

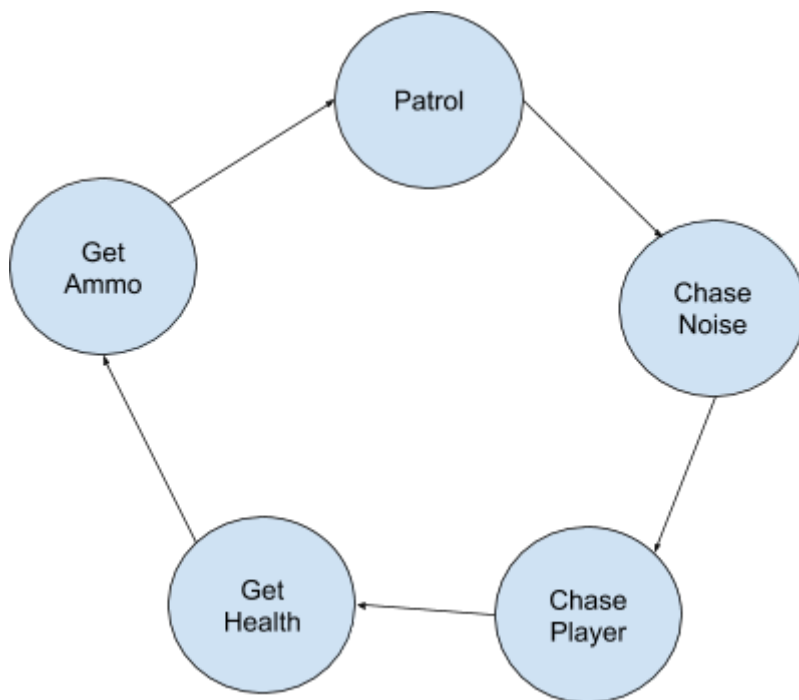


Figure 6. AI State [Primary Source]

If the respective state decision is true, then the AI will stay in the state else it will move to the next one and will try to find the next state.

## Pathfinding A\*:

A\*, Dijkstra and Breadth First Search are one of the most known pathfinding algorithms. Breadth First Search explores the grid equally in all directions while Dijkstra explores the possible path taking into consideration the cost of each move. A\* is a combination between Breadth First Search and Dijkstra. Instead of exploring in all directions or exploring all possible paths, A\* is a more optimized method for a single destination and also prioritizes paths that lead closer to the final position. ("Introduction to A\*", 2014)

The A\* was integrated mainly by using Sebastian Lague tutorials but it needed to be adapted in order to work with Steering Behaviors. (A\* Pathfinding, 2014) In this tutorial was also suggested to use a heap sort algorithm in order to improve the performance of the algorithm. An interesting thing that was presented was that sometimes the AI might move in a straight direction so there is no need to save all the waypoints. A better approach would be to save the waypoints only when the AI changes its direction.

The most important steering behavior that needed to be combined with the A\* algorithm is represented by the Path Follow behavior because this allows the player to move around the map. In the artefact the, AI has some predefined waypoints that were called global waypoints. When the AI gets close to a global waypoint it generates a path to the next global waypoint. This path that is created between two global waypoints was called the local path and it is formed from local waypoints.

In the images below the local waypoints (red) and global waypoints (black) can be observed.

Figure 7 shows the AI patrolling between waypoints. Red points are local waypoints while green points are global waypoints.

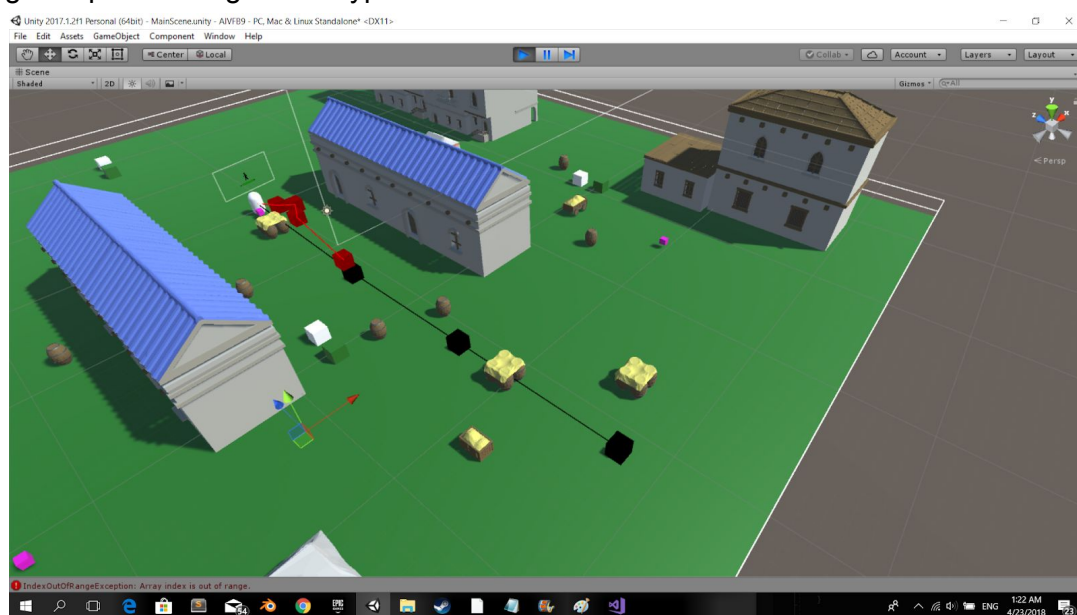


Figure 7. AI Patrol Waypoints [Primary Source]

Figure 8 shows the AI patrolling between waypoints. Red points are local waypoints while green points are global waypoints.

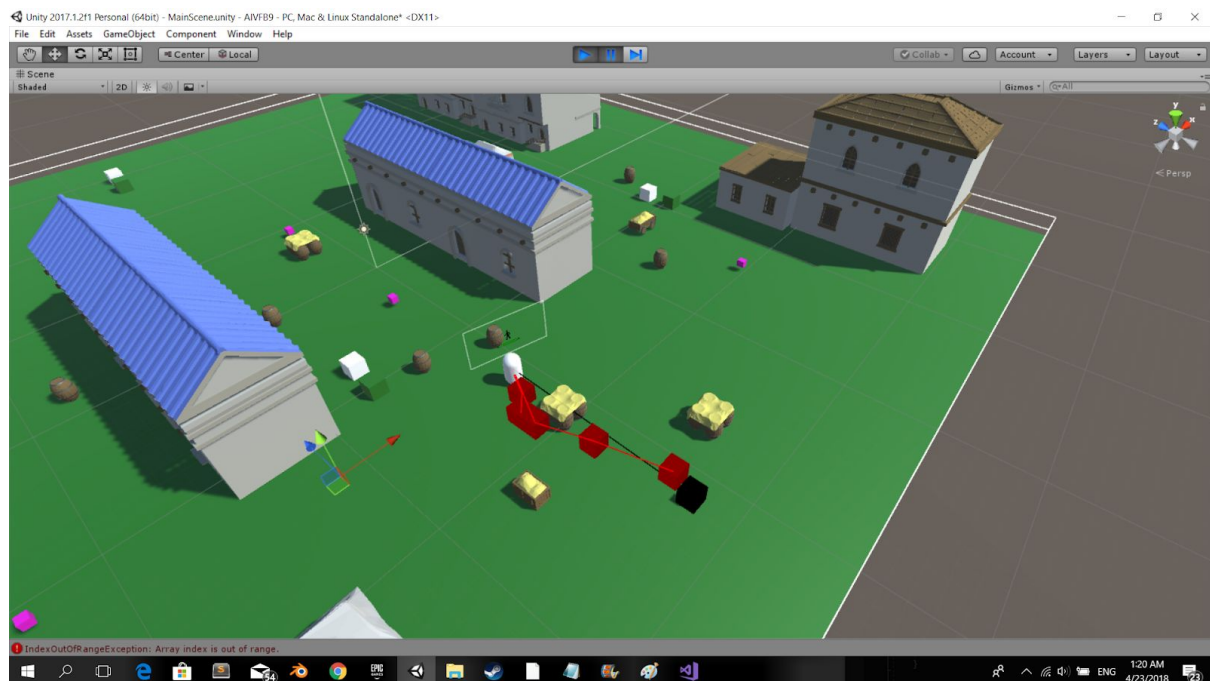


Figure 8. AI Patrol Waypoints [Primary Source]

The Seek steering behavior is used to move the AI between these waypoints because the movement needs to look continuous. When the AI reaches the final global waypoints it resets and moves to the first global waypoint.

Also, similar with the path following behavior I decided to create a path to target behavior that will move the AI to a certain point. The main difference between this behavior and the previous one is that the AI doesn't need to loop between waypoints. Furthermore when the AI is close enough to the target the Arrive behavior is used instead of the Seek behavior in order to make the movement looks more natural.

## Steering behaviors:

I did choose to use steering behaviors in my artefact because they have the role of making autonomous characters move in a realistic way, by using forces that are combined and produce a life-like feeling. ("Understanding Steering Behaviors", n.d) So, I considered that steering behaviors would work great for crowds.

All the steering behaviors were implemented but object avoidance. However, in the artefact, only seek, arrive, cohesion, separation, alignment and wall avoidance behaviors are used by characters. The steering behaviors were mainly used on crowds in order to get a realistic group movement.

One of the most important aspect of the steering behaviors is represented by the ability of combining behaviors together. There are three , Weight Truncated Sum, Prioritized Dithering, Weight Truncated Running Sum with Prioritization. In the artefact the Weight

Truncated Running Sum with Prioritization was used because it gives a good compromise between speed and accuracy. (Buckland, 2005, p.121)

Figure 9 shows the Weight Truncated Running Sum with Prioritization code.

```
public Vector3 Calculate()
{
    //VelocitySum = Vector3.zero;
    Vector3 force;
    VelocitySum = Vector3.zero;

    if (IsSeparationOn)
    {
        force = Separation() * separationWeight;
        if (!accumulatedForce(ref VelocitySum, force))
            return VelocitySum;
    }

    if (isWallAvoid)
    {
        force = Wall() * wallAvoidanceWeight;
        if (!accumulatedForce(ref VelocitySum, force))
            return VelocitySum;
    }

    if (IsAlignmentOn)
    {
        force = Alignment() * alignmentWeight;
        if (!accumulatedForce(ref VelocitySum, force))
            return VelocitySum;
    }

    if (isCohesionOn)
    {
        force = Cohesion() * cohesionWeight;
        if (!accumulatedForce(ref VelocitySum, force))
            return VelocitySum;
    }

    if (IsSeekOn)
```

Figure 9. Weight Truncated Running Sum with Prioritization [Primary Source]

An important aspect when applying the Cohesion, Separation and Alignment on the AI is that when the AI iterates through tagged vehicles it needs to exclude itself because otherwise it will apply a force against itself.

The cohesion behavior is considered to be useful when trying to keep agents together. (Buckland, 2005, p.117) In the artefact it was used to test the distance between the leader and the rest of the crowd. For example if the distance is too big, the leader would stop and wait for the other agents.

Figure 10 shows a crowd AI.

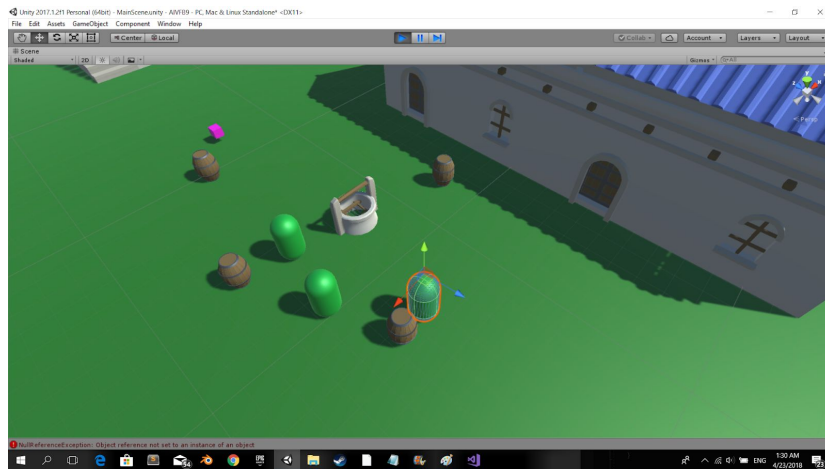


Figure 10. Crowd AI [Primary Source]

Figure 11 shows a crowd AI.

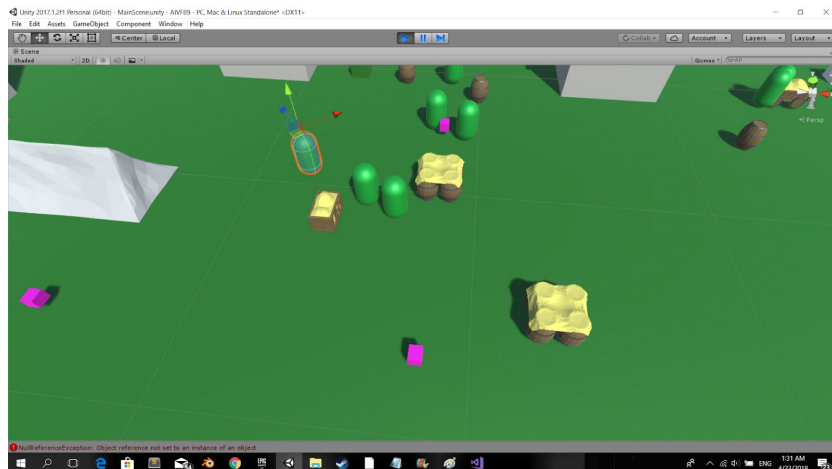


Figure 11. Crowd AI [Primary Source]

## Desirability :

Desirability was implemented in order to make the AI look less repetitive. It is used to decide if the AI wants to reload its weapon or refill its health. In order to integrate desirability I made use of the two formulas that were presented during the lectures.

Figure 12 shows the Desirability code

```
//calculate health regeneration desirability
float healthDesirability()
{
    float value = 0;
    value = (1 - currentHealth / 100) / Vector3.Distance(transform.position, closestToPlayer(healthPacks).transform.position);
    return value*20;
}

//calculate ammo reload desirability
float ammoDesirability()
{
    float value = 0;
    value = ((currentHealth / 100) * (1 - currentAmmo / 10)) / Vector3.Distance(transform.position, closestToPlayer(ammoPacks).transform.position);
    return value*10;
}
```

Figure 12. Desirability code [Primary Source]



As a result of this, for example, if the AI is very close to a health pack he will decide to regenerate its health even if the health level is not very low.

In the artefact, desirability creates some form of emergent behavior, but in this case it creates an emergent behavior that is wanted and it is controlled.

## Other AI features:

### Visual sense:

The AI is able to spot the player only if the player is in the FOV. This feature was implemented by calculating the dot product between AI's position and the player's position. Then if the player is in front of the player the AI checks if the player is in the FOV limits.

### Auditive sense:

The AI is able to hear any noises made by the player and will move towards the noise source.

In order to implement this feature I also made use of the A\* algorithm because there can be situations where the real distance between the AI and the player is very small but there might be a wall between them, so the AI shouldn't hear the noise. By using the A\* the AI calculates will calculate the actual path length between the sound source and the AI.

Figure 13 shows an example of pathfinding used to calculate distance to noise.

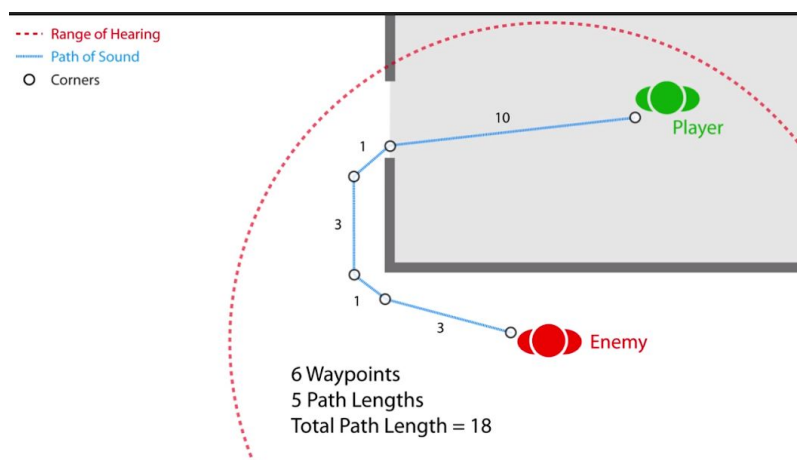


Figure 13. Stealth Game tutorial - 403 - Enemy Sight and Hearing - Unity Official Tutorials, 2013

## Reflection and Conclusion:

Reflecting on the pathfinding and steering behaviors combination, I believe that my solution is not the optimal one when working with crowds. The first idea was to have one crowd



leader followed by multiple leaders. The crowd leader will use the pathfinding algorithm to move around the map while the agents would only use steering behaviors to stay close to the leader. Unfortunately, even if the Weight Truncated Running Sum with Prioritization was used there were moments when too much force was applied on agents and they were stuck and had an unnatural behavior. The alternative alternative solution was to use the pathfinding algorithm on the agents as well so they would avoid all the obstacles and agents. Unfortunately this method was using a lot of resources because the world grid on which pathfinding is calculated, needed to be updated every frame in order to keep track of agents' new positions.

Also reflecting on the scriptable objects approach with FSM i believe that it wasn't the best one because scriptable objects can be used only in Unity. So probably a more general solution such as a stack-based approach would have been better.

In conclusion, I believe that most of the requirements have been implementer. However i believe that the pathfinding code should be optimized because it currently uses lots of resources and this happens because the path is calculated in real time. Also there can be some adjustments made on the Steering Behaviors in terms of weights.

Regarding the code, I tried to make it as modular as possible but I consider that it could have been improved. For example in the Steering Behaviors files I have two similar pathToTarget functions, one used for the Hostile AI and the other one used for crowds. This happened because I have two different NPC files, one used for crowds and the other one used for the Hostile AI. This happened mainly I didn't consider integrating crowds when I started the project. I am aware that I should have made use of OOP principles and use inheritance to create different type of AIs.

#### References:

Buckland, M. (2005). *Programming Game AI by Example*. Retrieved from: <https://unisalesianogames.files.wordpress.com/2011/08/programming-game-ai-by-example-mat-buckland2.pdf>

Lague, S. (2014). *A\* Pathfinding* [Video File]. Retrieved from: <https://www.youtube.com/watch?v=-L-WgKMFuhE&t=3s>

n.d. (2014). *Introduction to A\**. Retrieved from [:https://www.redblobgames.com/pathfinding/a-star/introduction.html](https://www.redblobgames.com/pathfinding/a-star/introduction.html)

n.d. (n.d). *Polymorphism in C++*. Retrieved from: <https://www.geeksforgeeks.org/polymorphism-in-c/>

n.d. (n.d). *State*. Retrieved from: <http://gameprogrammingpatterns.com/state.html>

Owens, B. (2014). *Goal Oriented Action Planning for a Smarter AI*. Retrieved from: <https://gamedevelopment.tutsplus.com/tutorials/goal-oriented-action-planning-for-a-smarter-ai--cms-20793>

Rasmussen, J. (2016). *Are Behavior Trees a Thing of the Past?*. Retrieved from: [https://www.gamasutra.com/blogs/JakobRasmussen/20160427/271188/Are\\_Behavior\\_Trees\\_a\\_Thing\\_of\\_the\\_Past.php](https://www.gamasutra.com/blogs/JakobRasmussen/20160427/271188/Are_Behavior_Trees_a_Thing_of_the_Past.php)

Unity. (2013). *Stealth game tutorial - 403 - Enemy Sight and Hearing - Unity Official Tutorials* [Video File]. Retrieved from: <https://www.youtube.com/watch?v=mBGUY7EUxXQ>

Unity. (n.d). *3 cool ways to architect your game with Scriptable Objects*. Retrieved from: <https://unity3d.com/how-to/architect-with-scriptable-objects>

Wong, K. (2014). *Stealth Game Design*. Retrieved from: [https://www.gamasutra.com/blogs/KevinWong/20140516/217856/Stealth\\_Game\\_Design.php](https://www.gamasutra.com/blogs/KevinWong/20140516/217856/Stealth_Game_Design.php)