**A1**

TASK 1

* **Explain the sequence your code needs to take to update any paths depending on the obstacles generated.**

(posted as a commit on github)

Using Pathfinding’ lets the script know that class ‘Path’ exists. Class Seeker handles pathfinding calls attached to a single unit, in our case the AI. On Start, a path to the target object’s position is calculated using StartPath method on the Seeker component and returns the result to the callback function (a start position, an end position and a callback function. The callback function then calls the Coroutine which moves the AI 0.5 units every 0.1 seconds towards the target.

TASK 2

* **A\* pathfinding pro implements this functionality using a specific feature. Explain what this feature is.**

Feature: **Local Avoidance**

- it is based on RVO: Reciprocal Velocity Obstacles which is divided into 2 parts

- Core simulation code and Unity interface

TASK 3

* **What is the difference between a grid graph, point graph and a navmesh?**

**Grid Graph**

The most straightforward graph from all 3, where nodes are generated in a predictable grid pattern width \* depth ( x \* z ). One of the main usages of grid graphs is when graph changes during runtime is needed. A small part of the grid can be updated, preventing any lag unlike a complete rescan would. They are suitable when already having a grid-based world and for the majority of scenes with minimal configurations. Raycast and funnel algorithm are supported, along with applying penalty and walkability values from a supplied image. Hence, making areas unwalkable make grid graphs ideal for terrain worlds.

**Point Graph**

The simplest of all graph types which allows a substantial amount of customization where users place linked points. Scanning a point graph consists of taking a root Transform and treating its children as nodes, and by using raycasts to check connections between the nodes to see whether they could be linked. A disadvantage of such graph is not getting good, smooth paths since only a point of walkability is defined, unlike the other graphs which define areas. However, the raycast modifier handles this issue well enough.

**Navmesh Graph**

Navmeshes are meshes of polygons, each defining a walkable area. AI is able to get further information on how it can walk. Pathfinding data is expressed as a triangle mesh, unlike Grid Graph which uses squares and Point Graph which uses point. It is suitable for fast and smooth pathfinding in which graph changes during runtime are not needed. This graph usually contains fewer nodes than grid graph, hence requires less searching and makes it faster.

* **What is the biggest computer bottleneck for AI pathfinding?**

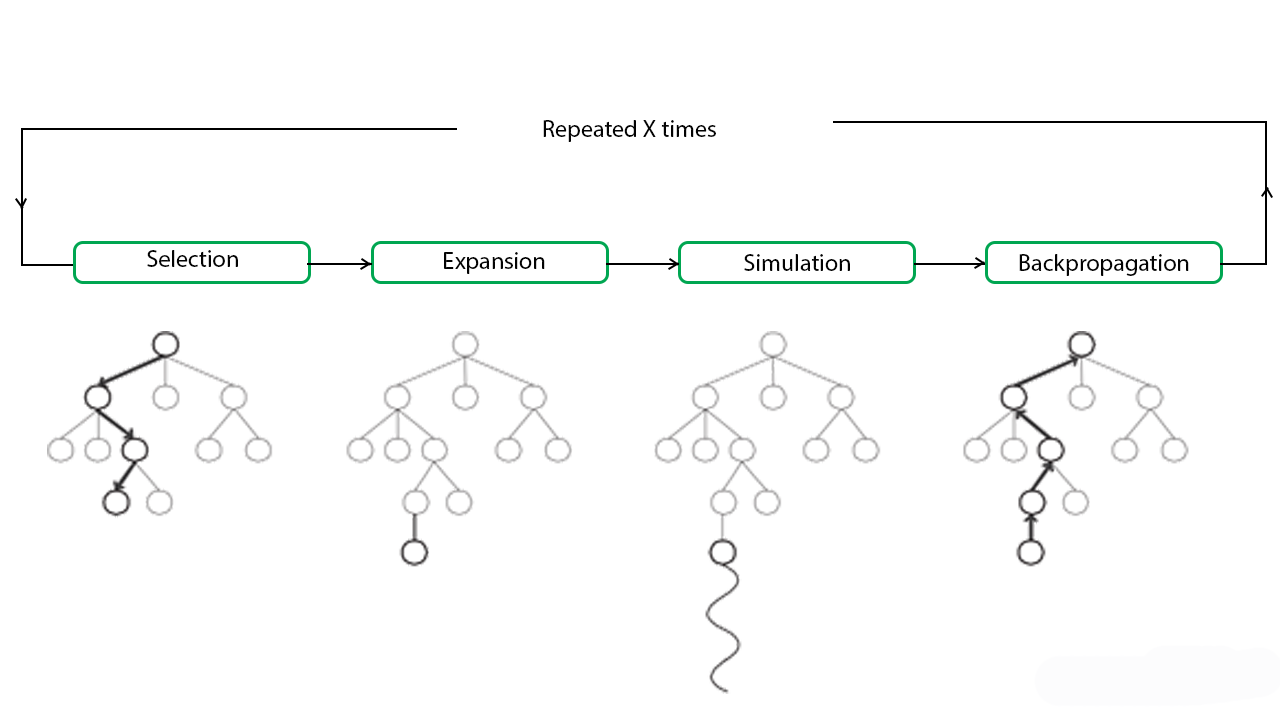
In AI pathfinding, a very common bottleneck is when working with large maps. These cause significant performance bottlenecks where the game can be slow down noticeably, or the AI can become stupid due to improper, nonoptimized pathfinding, and data structure.

TASK 5

* **Find out what a search tree is and explain the kind of search tree that is being used for pathfinding. Are the same algorithms used for combat AI?**

Pathfinding usually gets the AI from point A to point B in the most direct way possible. However, the Monte Carlo Tree Search (MCTS) method provides a more engaging game experience by creating additional obstacles for the player to overcome. In Artificial Intelligence, the MCTS algorithm is used as a search technique, combining the classic tree search implementations and machine learning principles of reinforcement learning. It works in four phases:

1. **Tree Traversal Phase**: uses the UCB1 formula,
2. **Node expansion Phase**: add extra nodes into the tree,
3. **Rollout Phase**: random simulation of the game, or the problem being solved, in order to find the value,
4. **Backpropagation Phase**: take the value from the rollout and put that in appropriate places in the tree all the way up in the top node.



A commonly used Pathfinding search tree is the **A\* algorithm**, which is a combination of 2 other algorithms: Dijkstra and Greedy. It considers both distance from root node and heuristic distance to the goal, unlike Dijkstra which only considers distance from root. Both algorithms find a shortest path. The similarity with Greedy Best-First-Search is that both use a heuristic to guide themselves, hence both are equally fast. The A \* algorithm then terminates once goal node is found.

In combat AI, the common goal is making the AI more human, hence why its ability of being efficient is crucial in such games which include a level of combat. The Monte Carlo Tree Search is used in such games as it adapts to different circumstance without relying on predefined actions patterns or tactics which end up making them predictable to human players. Some games create AI players with a persona, so each character in the game would have different behaviours. A study by Ishii, R (2018) shows that using a variant of MCTS called Puppet-Master MCTS makes characters behave accordingly to their given persona, since it not only controls all characters in the game, but also introduces a new evaluation function, making each character take their actions according to their given persona.

TASK 6

* **A\* pathfinding is a popular AI plugin for the Unity game engine. Compare it with one other pathfinding plugin you find on Github based on 5 points.**

**NavMeshPlus Plugin:**

NavMesh building components provide the ability of creating navigation, automatically generated meshes from the Scene geometry, allowing characters to move intelligently around the game world.

* Uses <https://docs.unity3d.com/Manual/class-NavMeshSurface.html> as base implementation,
* Implements world bound calculation from Tilemap bounds,
* Implements source collector of tiles, sprites and 2d colliders, because NavMeshBuilder.CollectSources will not work,
* Creates walkable mesh box from world bounds,
* Convert tiles to sources as NavMeshBuilder would do.

References:

Doxygen, 2016, The A\* Pathfinding Project for Unity 3D. [ONLINE] Available at: https://arongranberg.com/astar/docs\_dev/graph\_types.php. [Accessed 22 January 2021].

Elgabry, E. 2016. Path Finding Algorithms. [ONLINE] Available at: https://medium.com/omarelgabrys-blog/path-finding-algorithms-f65a8902eb40. [Accessed 31 January 2021].

Ishii, R, Ito, S, Ishihara M, Harada, T, Thawonmas, R. 2018. Monte-Carlo Tree Search Implementation of Fighting Game AIs Having Personas. [ONLINE] Available at: https://ieeexplore.ieee.org/document/8490367. [Accessed 31 January 2021].

Patel, A 2020, Introduction to A\*. [ONLINE] Available at: http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html. [Accessed 31 January 2021].

Roy, R 2019, ML | Monte Carlo Tree Search (MCTS). [ONLINE] Available at: https://www.geeksforgeeks.org/ml-monte-carlo-tree-search-mcts/. [Accessed 31 January 2021].

Wikipedia. 2021. Artificial intelligence in video games. [ONLINE] Available at: https://en.wikipedia.org/wiki/Artificial\_intelligence\_in\_video\_games#Video\_game\_combat\_AI. [Accessed 31 January 2021].