JUSTIN SCHEMBRI

MSD 6.2A

SCFG ASSIGNMENT 1

KU3

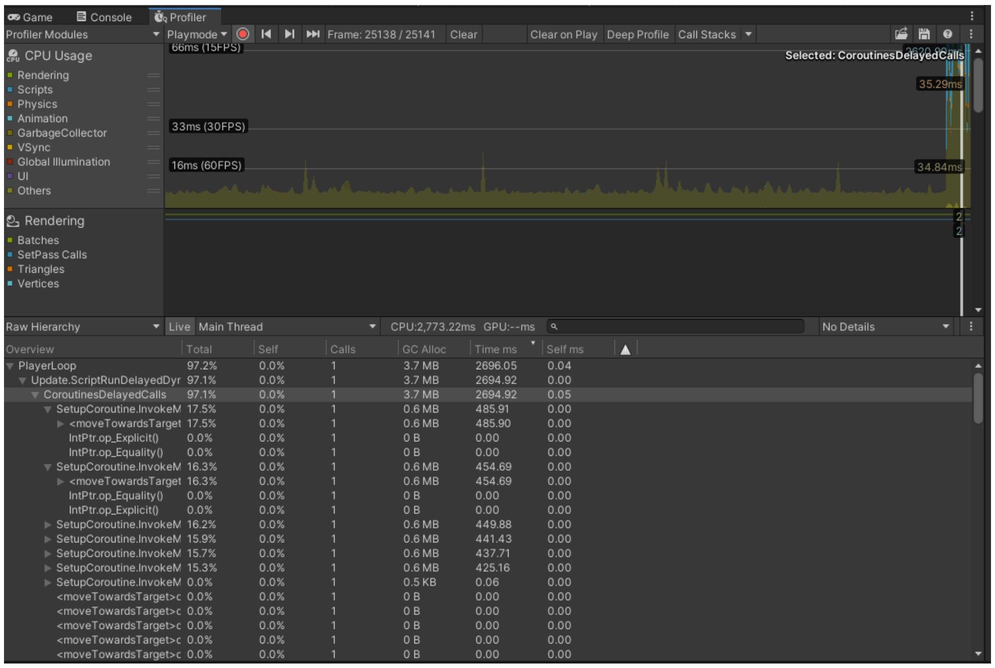
a)

a. Grid Graph:
A Grid Graph generates nodes in a grid pattern using width and depth. (1)

b. Point Graph:
A Point Graph consists of a lot of user placed points that are linked together. The point graph is scanned by taking the transform of the Root and treating every child as a node. It then uses casts a raycast to each point to check if they should be linked together. A point graph can only be used to define the walkability of a play space and nodes should not be placed too far apart from one another. (1)

c. Navmesh:
A Navmesh graph uses triangles meshes to generate pathfinding data. Infact it is a mesh with polygons that describe a walkable area. It is a perfect implementation for smooth and fast pathfinding especially in instances where the graph doesn't change during runtime. (1)

b) The Biggest bottleneck for AI pathfinding is the coroutine that allows the enemy to move in the path to the target.



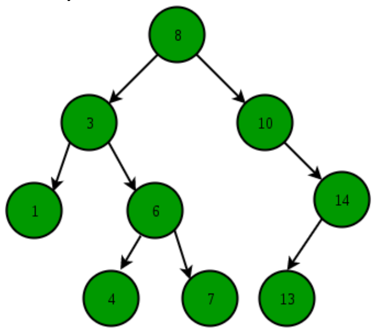
CPU:2,773.22ms GPU:--ms

Overview	Total	Self	Calls	GC Alloc	Time ms	Self ms
PlayerLoop	97.2%	0.0%	1	3.7 MB	2696.05	0.04
Update.ScriptRunDelayedDyr	97.1%	0.0%	1	3.7 MB	2694.92	0.00
CoroutinesDelayedCalls	97.1%	0.0%	1	3.7 MB	2694.92	0.05
SetupCoroutine.InvokeM	17.5%	0.0%	1	0.6 MB	485.91	0.00
<moveTowardsTarget	17.5%	0.0%	1	0.6 MB	485.90	0.00
IntPtr.op_Explicit()	0.0%	0.0%	1	0 B	0.00	0.00
IntPtr.op_Equality()	0.0%	0.0%	1	0 B	0.00	0.00
SetupCoroutine.InvokeM	16.3%	0.0%	1	0.6 MB	454.69	0.00
<moveTowardsTarget	16.3%	0.0%	1	0.6 MB	454.69	0.00
IntPtr.op_Equality()	0.0%	0.0%	1	0 B	0.00	0.00
IntPtr.op_Explicit()	0.0%	0.0%	1	0 B	0.00	0.00
SetupCoroutine.InvokeM	16.2%	0.0%	1	0.6 MB	449.88	0.00
SetupCoroutine.InvokeM	15.9%	0.0%	1	0.6 MB	441.43	0.00
SetupCoroutine.InvokeM	15.7%	0.0%	1	0.6 MB	437.71	0.00
SetupCoroutine.InvokeM	15.3%	0.0%	1	0.6 MB	425.16	0.00
SetupCoroutine.InvokeM	0.0%	0.0%	1	0.5 KB	0.06	0.00
<moveTowardsTarget>c	0.0%	0.0%	1	0 B	0.00	0.00
<moveTowardsTarget>c	0.0%	0.0%	1	0 B	0.00	0.00
<moveTowardsTarget>c	0.0%	0.0%	1	0 B	0.00	0.00
<moveTowardsTarget>c	0.0%	0.0%	1	0 B	0.00	0.00
<moveTowardsTarget>c	0.0%	0.0%	1	0 B	0.00	0.00

KU5

A Search Tree is a tree where every subtree of a node has less keys than that subtree of the node to its right. (2)

A good example of this would be a Binary Search Tree.



AI Pathfinding can utilize a number different search trees, these include:

- Shortest Path: calculates the shortest distance between 2 nodes,
- All Pairs Shortest Path: which is an optimized calculation of the shortest path from all nodes to all other nodes,
- Single Source Shortest Path: which is the shortest path calculated from the root node to all other nodes and then traverses to the next unvisited node with the lowest weight from the root node,
- Minimum Spanning Tree: which is the shortest path connecting all nodes and traversing to the next unvisited node with the lowest weight from any visited node(3)

The Monte Carlo Search Tree for example is a search tree that is used primarily in games like Tic-Tac-Toe, Rubik's Cube, Sudoku and Chess to predict the path that should be taken to reach the winning solution. It works by figuring out the best move out of a set of moves through a 4 step process.

- Selecting: which revolves around selecting the node from the tree that has the highest possibility of winning.
- Expanding: which creates child nodes on that newly selected nod. These nodes will indicate future moves that will be played in the game.
- Simulating or Exploring: With the use of Reinforcement Learning to make random decisions from every child node and rewarding each node based on the calculation that leads the closest to the solution.
- Updating: With the scores handed out, the scores on each node will then be updated and can change depending on the selection process of the future nodes.(4)

The Monte Carlo Search Tree is ideal for decisions that need to be made by combat AI.

However as seen in this paper based on a 2D search space, the monte carlo algorithm was made to solve a 40x40 puzzle and managed in 21 minutes which according to the study "To the best of our knowledge there is no other efficient solution for this puzzle where the size of the problem is considerably large"(5)

KU6

For this Question I will be comparing Arongranberg ‘s A* PathFinding (https://arongranberg.com/astar/front) and Mattatz unity path finding (https://github.com/mattatz/unity-path-finding).

Arongranberg’s A* pathfinding has a handful of features, 5 of these are:

- The Ability to do all 3 styles of path finding (Navmesh, Grid Graphs, Point Graphs)
- Can do 2D pathfinding
- Has an avoidance system based on a selected layer.
- Has a penalty/weight system to identify how hard it is to traverse specific nodes.
- Can cut through navmeshes especially those made up of triangles.

Mattatz pathfinding algorithm on the other hand uses Dijkstra’s algorithm specifically to find the shortest possible route and can only perform pathfinding in a 2D environment. It does not have an avoidance system and it cannot cut through meshes. It can travel to the next node with the lowest weight till it reaches the final destination.

References

1. Arongranberg.com. 2021. *A* Pathfinding Project: Graph Types*. [online] Available at: <https://arongranberg.com/astar/docs_dev/graph_types.php> [Accessed 14 January 2021].

2. Xlinux.nist.gov. 2021. *Search Tree*. [online] Available at: <https://xlinux.nist.gov/dads/HTML/searchtree.html> [Accessed 14 January 2021].

3. O'Reilly Online Learning. 2021. *Graph Algorithms*. [online] Available at: <https://www.oreilly.com/library/view/graph-algorithms/9781492047674/ch04.html> [Accessed 14 January 2021].

4. Medium. 2021. *Monte Carlo Tree Search*. [online] Available at: <https://towardsdatascience.com/monte-carlo-tree-search-158a917a8baa> [Accessed 14 January 2021].

5. Ieee-cog.org. 2021. [online] Available at: <https://ieee-cog.org/2019/papers/paper_254.pdf> [Accessed 14 January 2021].