Part 5 (Algorithm description):

My strategy for pathfinding relies on how I've set up the level: There are three main grids, each overlaid on top of a platform. These grids are 25 x 15, and so have 375 nodes each. Once our agents are spawned, each one gets a destination. Then, depending on where this destination is relative to the Agent's current location, we decide to walk, take a teleporter or take a bridge.

- If the agent is on the side platform and wants to travel to the top or bottom, take a bridge. If the agent is on the top or bottom platform and wants to travel to the side platform, we also have it take a bridge.
- If the agent is on the same platform as its destination, it simply walks there.
- If the agent is on the top platform and wants to get to the bottom one, use a teleporter.
- If the agent is on the bottom platform and wants to get to the top, use a teleporter.

Once a destination is set and a path is found an agent's travel() function is called, which gradually has them travel along the path.

If heading to a bridge, the PathfindingManager will first give the agent a path to a BridgeNode, of which there are 2 per bridge. Once the agent reaches a BridgeNode, its grid is changed to the one on the other side, and a new path is calculated with the first node always being the bridge node on the other side. When agents get onto a bridge, they also "broadcast" the fact that they are on the bridge and in which direction they are going. So, any other agents making their way to the bridge in an incompatible direction (any direction other than the agent currently on the bridge is going) are halted, and wait until the bridge is freed (if this takes too long, they timeout and create a new path).

Teleporters are made up of 6 "TeleporterNodes", with 3 "WaitArea" nodes and 3 "Teleport" Nodes. These teleporters operate on a fixed schedule, which goes through 3 states: First, "Empty", which removes any Agents in the teleporter nodes. Then, "Enter", which allows up to 3 agents in the waiting nodes to go onto the teleport nodes. Finally, "Teleport", which teleports up to 3 agents on the teleporter nodes. These operations take 2s, then 2s, then 1s for teleportation (as specified in the assignment). This chain of states alternates for the bottom and top two teleporter pads. If an agent is heading to a teleporter, it can only go into a WaitNode, and it only travels there if said WaitNode is available. If it is occupied, the agent waits until it is freed or until a timeout.

The two algorithms I used to compute the actual pathfinding were the simple A* algorithm and the RRT algorithm, both of which operate nearly identically to what we've seen in class. The A* is slightly edited to ensure we don't use TeleporterNodes as part of a normal path (not detected as neighbours), and the RRT decides visibility using a Raycast. Since paths always go on the same grid, there are never any cases where a Raycast goes "in between" platforms so we don't get agents flying through the air.

The logic behind this works very well within the code - the only glaring problem is that to prevent NPC overlap, I simply push NPCs slightly out of each other's way. This is fine with a small/moderate amount of agents, but when many NPCs are concerned, this can sometimes result in things looking off, though everything works properly behind the scenes (for example, an NPC pushed off of the teleporter node will still be teleported. This makes sense because in the code, the NPC is still occupying the teleporter, but visually it obviously looks a bit off. Similarly, this sometimes makes NPCs seem as though they are floating, or they get pushed down, or as if they are going side-by-side on a bridge, when in the code they are not). You can remove the "AgentPushScript" in the Agent prefab to see that everything works correctly when collisions are off, except for (obviously) the overlap prevention.

Unfortunately, I ran out of time to figure out a better solution for preventing NPC overlap.

Part 6 (Data analysis):

A* Data Analysis (averaged over three 2min executions)

	Number of findPath() calls	Avg. time spent per pathfinding call	Number of agent repaths	Number of abandoned paths
4 Agents	59	1.05799e-4	0.6666666	0
8 Agents	125	7.65959e-5	2	1.3333333
16 Agents	238.33333	8.71382e-5	3.3333333	5.6666666
32 Agents	447.33333	7.78596e-5	7.3333333	39.333333
64 Agents	8284	1.75036e-6	21	112.33333
128 Agents	79463.6666	2.51970e-7	29.6666666	578.66666
256 Agents	403199.666	9.48543e-7	49.3333333	1419.3333

From this table we can see that the amount of findPath() calls seems to increase exponentially, since it starts sharply increasing. Next, the average time spent per call is fairly consistent but does get smaller - this is likely due to having less space, so shorter paths are created and so pathfinding is faster. Finally, since timeouts can take up to 10s and there is a lot of randomness involved, the repaths and abandonments are fairly low (though in some tests they shoot quite high, so again I attribute this to randomness. It may also be that somewhere in my code too little / too much data is being output due to some oversight)

We stopped at 256 agents because with 512, there were some cases where all agents were attempted to be spawned on all the same grid (due to randomness), which broke down the program. This is clearly unacceptable.

RRT Data Analysis (averaged over three 2min executions)

	Number of findPath() calls	Avg. time spent per pathfinding call	Number of agent repaths	Number of abandoned paths
4 Agents	70.3333333	3.02933e-5	0	1
8 Agents	151.333333	8.77624e-5	1	4
16 Agents	283.333333	1.19990e-5	2.6666666	15.333333
32 Agents	618.333333	9.11827e-6	10.666666	72.333333
64 Agents	3713.33333	2.24073e-6	19.333333	304.33333
128 Agents	65725	2.19910e-7	32	869
256 Agents	402338.666	3.49062e-7	40.333333	1723.3333

RRT seems to have more abandoned paths than A*, but given that abandoned paths and number of repaths can fluctuate wildly according to randomness, this doesn't tell us much and is likely a coincidence. However, we can see that A* and RRT have roughly the same amount of findPath() calls and similar trends in the avg. time spent per pathfinding call, which makes sense. However, we do realize that if we calculated the average total time spent for the whole 2mins during pathfinding, RRT will have less time, which is consistent with what we know from class (that RRT is rarely optimal but usually faster, especially in a scenario like this where we have no tight corridors).

Other notes are that on 256 Agents, unexpected behavior occurs (eg. Agents should timeout and repath but this does not occur and they sit there for much longer). Also, as discussed in A*, at 512 agents there may not be enough room on a grid to spawn all the agents, so the program breaks down.

So, it seems that both operations can support roughly the same amount of agents, up to a cap which is limited by the size of our grids - at 375 nodes and up, this possibility happens - and not the algorithms themselves. They could likely support many more agents before breaking down as long as the grid(s) are large enough, although as mentioned RRT *sometimes* has some strange behavior from 256 agents onwards so that one may not be able to handle much more.