Nesibe Betül Döner 2023719039
SWE585 – Term Project

AI Pathfinding with Static and Dynamic Obstacles

1. Overview

This project focuses on evaluating the impact of static and dynamic obstacles on pathfinding performance in Unity's NavMesh system. The experiment compares CPU usage, memory usage, and AI update times across different scenarios, including static and dynamic obstacles with various agent and obstacle counts. Two optimization techniques are applied to improve performance in high-load environments. The results show that dynamic obstacles have a more significant impact on performance metrics compared to static obstacles, and one of the optimization techniques reduce the computational load.

1.1. Goal

The main goals of this project are:

- To measure the performance impact of static and dynamic obstacles on pathfinding.
- To evaluate how different agent and obstacle counts affect CPU usage, memory usage, and AI update times.
- To compare the efficiency of two optimization techniques in reducing performance overhead caused by dynamic obstacles.

1.2. Technical Challenge

The primary challenge is maintaining real-time pathfinding performance when the NavMesh is dynamically updated due to moving obstacles. Dynamic carving can significantly increase CPU usage and delay path recalculations for agents. Optimization techniques such as reducing carving frequency and caching behaviours are tested to overcome these challenges.

2. Experiments

To be able to observe the effects of static and dynamic obstacles, their counts, and agent counts to NavMesh and thus game performance metrics, an experiment environment is created in Unity for various experiments to compare the observed features.

2.1. Environment Design

A 3D environment is created in Unity using the following game objects for each scene:

- a plane for the floor,
- boxes for the walls,
- a prefab using sphere for the agent,
- a prefab using capsule for the target,
- prefabs using boxes for the obstacles,
- an empty game object

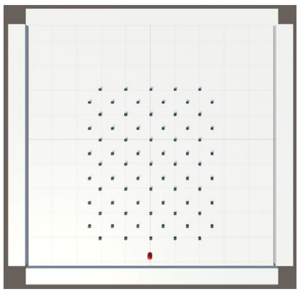
There are 4 scenes created for the experiments.

- Scene with 71 static obstacles
- Scene with 171 static obstacles
- Scene with 71 dynamic obstacles
- Scene with 171 dynamic obstacles

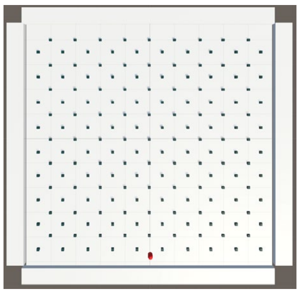
In addition to the above scenes there are two scenes which are the main scene without obstacles and the initial main scene which is used for the initial version of the experiment.

2.1.1. Screenshots of the environments

- Environment with 71 obstacles



- Environment with 171 obstacles



2.1.2. Components of game objects

In this section, the components which differ from the default game object components are mentioned.

2.1.2.1. Floor:
- Floor is created from plane.
- NavMeshSurface is added in order for NavMesh to accept floor as surface which is the place where the agent moves.

2.1.2.2. Agent:
- Agent is a prefab created from sphere.
- Nav Mesh Agent is added in order for NavMesh to accept this as agent which is the game object that moves on the NavMesh.
- Agent Controller script is added in order to control the behaviours of agent(s).
- Material in order to add colour.

2.1.2.3. Target:
- Target is created from capsule.
- Material in order to add colour.

2.1.2.4. Dynamic obstacle:
- Dynamic obstacle is a prefab created from boxes.
- Nav Mesh Obstacle is added in order for NavMesh to accept this as an obstacle and carve the NavMesh accordingly.
- Obstacle Mover script is added in order to control the movements of dynamic obstacles. In the optimization techniques, this script is used to change carvingMoveThreshold carving setting for the first approach to reduce the frequency of NavMesh updates and to change the carvingTimeToStationary carving settings for the second approach to delay carving for obstacles that remain stationary for a while.

2.1.2.5. Static obstacle:

- Static obstacle is a prefab created from box.
- Static feature is marked in order to bake the NavMesh properly.

2.1.2.6. Agent manager:

- Agent manager is an empty game object.
- Agent Spawner script is added in order to control the pathfinding and agent count.

2.2. Experiment Design

The experiment is conducted to compare various combinations of agent counts, obstacle types, and obstacle counts. In order to achieve these comparisons, 4 different scenes are created for different obstacle types and counts. Additionally, for each scene, agent count can be changed from AgentManager game object. Therefore, data collected from the following experiment setups:

- Scene with 71 static obstacles for 50 and 500 agents
- Scene with 171 static obstacles for 50 and 500 agents
- Scene with 71 dynamic obstacles for 50 and 500 agents
- Scene with 171 dynamic obstacles for 50 and 500 agents
- Scene with 171 dynamic obstacles for 500 agents using optimization technique in approach 1
- Scene with 171 dynamic obstacles for 500 agents using optimization technique in approach 2

2.2.1. Metrics

The following metrics are used to evaluate the performance of the system:

- CPU Usage: Measures the computational load of the pathfinding system. Unity's Profile Analyzer is used to get the values.
- PreUpdate.AIUpdate: Tracks AI-related updates, including NavMesh carving and pathfinding. Unity's Profile Analyzer is used to get the values.
- Memory Usage: Monitors the total memory committed during the simulation. Profiler is used to get the values.

2.3. Experiment Results

2.3.1. Main scene with 1 agent

# of run	CPU Usage Mean (max-min) (ms)	PreUpdate.AIUpdate Mean (max-min) (ms)	Memory Usage
1	14.86 (832.61 – 7.34)	0.18 (1.27 – 0.06)	397
2	13.07 (831.97 – 6.31)	0.17 (0.57 – 0.08)	440
3	11.79 (825.87 – 6.53)	0.16 (0.70 – 0.06)	448
	13.24	0.17	428

This is the base scenario with 1 agent which shows minimal CPU usage, memory usage, and AI update times for the test environment. This is tested to serve as the baseline to compare other scenarios.

2.3.2. Scene with 71 static obstacles

2.3.2.1. 50 agents

# of run	CPU Usage Mean (max-min) (ms)	PreUpdate.AIUpdate Mean (max-min) (ms)	Memory Usage
1	14.11 (965.40 – 8.09)	0.93 (1.80 – 0.78)	445
2	13.56 (867.90 – 8.33)	0.93 (3.10 – 0.76)	450

3	13.62 (907.83 – 8.36)	0.92 (1.37 – 0.78)	455
	13.76	0.93	450

When 50 agents are following the target with 71 static obstacles, CPU and memory usage increase slightly compared to the baseline due to the presence of static obstacles and more agents. As static obstacles do not dynamically update the NavMesh, AI update times are relatively low to the following cases.

2.3.2.2. 500 agents

# of run	CPU Usage Mean (max-min) (ms)	PreUpdate.AIUpdate Mean (max-min) (ms)	Memory Usage
1	21.24 (876.84 – 14.57)	5.55 (8.64 – 3.88)	495
2	21.42 (947.93 – 14.22)	5.53 (7.72 – 3.88)	530
3	21.27 (902.89 – 14.68)	5.62 (7.60 – 3.82)	535
	21.31	5.56	520

When 500 agents are following the target with 171 static obstacles, metrics increased compared to 50 agents as pathfinding load scales with the agent count.

2.3.3. Scene with 171 static obstacles

2.3.3.1. 50 agents

# of run	CPU Usage Mean (max-min) (ms)	PreUpdate.AIUpdate Mean (max-min) (ms)	Memory Usage
1	15.08 (1061.4 – 8.36)	1.02 (1.82 – 0.83)	502
2	13.87 (903.83 – 8.72)	1.00 (1.92 – 0.84)	458
3	15.31 (1090.1 – 8.60)	1.02 (2.24 – 0.85)	467
	14.75	1.01	475

When 50 agents are following the target with 171 static obstacles, metrics increase slightly compared to the 2.3.1. due to the number of static obstacles. As static obstacles do not dynamically update the NavMesh, metrics are close to the metrics of 2.3.1..

2.3.3.2. 500 agents

# of run	CPU Usage Mean (max-min) (ms)	PreUpdate.AIUpdate Mean (max-min) (ms)	Memory Usage
1	21.91 (958.50 – 14.96)	5.97 (9.11 – 4.21)	501
2	21.28 (878.37 – 14.62)	5.97 (8.34 – 4.19)	540
3	21.95 (915.43 – 14.50)	6.09 (8.89 – 4.25)	535
	21.71	6.01	525

When 500 agents are following the target with 171 static obstacles, metrics increased compared to 50 agents as pathfinding load scales with the agent count.

2.3.4. Scene with 71 dynamic obstacles

2.3.4.1. 50 agents

# of run	CPU Usage Mean (max-min) (ms)	PreUpdate.AIUpdate Mean (max-min) (ms)	Memory Usage
1	17.11 (887.87 – 9.69)	1.18 (6.46 – 0.73)	418
2	14.61 (910.19 – 8.81)	1.04 (3.26 – 0.75)	505
3	14.40 (882.28 – 8.65)	1.02 (2.09 – 0.76)	490

	15.34	1.08	471

When 50 agents are following the target with 71 dynamic obstacles, the metrics are increased more compared to static obstacles as NavMesh carving occurs during the runtime.

2.3.4.2. 500 agents

# of run	CPU Usage Mean (max-min) (ms)	PreUpdate.AIUpdate Mean (max-min) (ms)	Memory Usage
1	22.66 (979.76 – 14.53)	5.88 (11.22 – 3.09)	540
2	22.52 (924.32 – 14.44)	6.23 (12.15 – 3.26)	550
3	24.59 (917.56 – 14.91)	6.60 (15.03 – 3.18)	550
	23.26	6.24	547

When 500 agents are following the target with 71 dynamic obstacles, the metrics are increased more compared to static obstacles as NavMesh carving occurs during the runtime.

2.3.5. Scene with 171 dynamic obstacles

2.3.5.1. 50 agents

# of run	CPU Usage Mean (max-min) (ms)	PreUpdate.AIUpdate Mean (max-min) (ms)	Memory Usage
1	17.87 (1168.1 – 9.68)	1.53 (7.01 – 0.61)	530
2	16.65 (935.53 – 9.48)	1.55 (6.81 – 0.63)	520
3	16.94 (935.70 – 9.67)	1.69 (6.12 – 0.63)	520
	17.15	1.59	523

With more dynamic obstacles, metrics increase further. The NavMesh must be updated more frequently to accommodate the obstacle movements.

2.3.5.2. 500 agents

# of run	CPU Usage Mean (max-min) (ms)	PreUpdate.AIUpdate Mean (max-min) (ms)	Memory Usage
1	30.09 (919.27 – 15.65)	11.74 (27.06 – 2.92)	550
2	33.33 (1302.7 – 16.64)	12.66 (33.78 – 3.00)	590
3	31.17 (956.75 – 15.61)	12.38 (35.73 – 2.95)	580
	31.53	12.26	573

The combination of 171 dynamic obstacles and 500 agents creates the most computationally intensive scenario, with the highest CPU usage, AI update times, and memory usage observed.

2.3.6. Scene with 171 dynamic obstacles for 500 agents using optimization technique in approach 1 – time to stationary is set 0.5 instead of 0.1

# of run	CPU Usage Mean (max-min) (ms)	PreUpdate.AIUpdate Mean (max-min) (ms)	Memory Usage
1	23.89 (988.77 – 14.81)	5.92 (31.83 – 2.93)	620
2	23.80 (988.83 – 14.55)	5.60 (30.69 – 2.94)	580
3	25.60 (1028.1 – 15.06)	6.39 (39.10 – 3.08)	590
	24.43	5.97	596

Increasing the frequency of carving for stationary obstacles lowers CPU usage and AI update times compared to the unoptimized scenario. Memory usage is higher due to the large number of agents and obstacles.

2.3.7. Scene with 171 dynamic obstacles for 500 agents using optimization technique in approach 2 – move threshold is set 1 instead of 0.5

# of run	CPU Usage Mean (max-min) (ms)	PreUpdate.AIUpdate Mean (max-min) (ms)	Memory Usage
1	36.66 (993.66 – 15.82)	14.40 (40.25 – 2.97)	590
2	44.95 (1015.9 – 16.46)	16.36 (51.14 – 2.89)	580
3	33.41 (1158.3 – 16.83)	13.16 (31.76 – 2.92)	590
	38.34	14.64	586

Increasing the move threshold reduces NavMesh update frequency, but this optimization technique leads to higher CPU usage and AI update times compared to Approach 1 and the default settings. This might be due to delayed updates causing more recalculations for agents navigating dynamic environments.

3. Reviewing Experiment Results

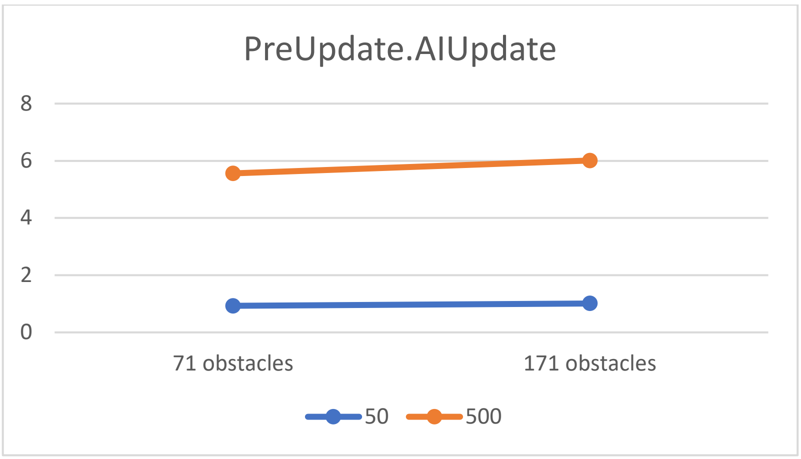
3.1. Static Obstacles

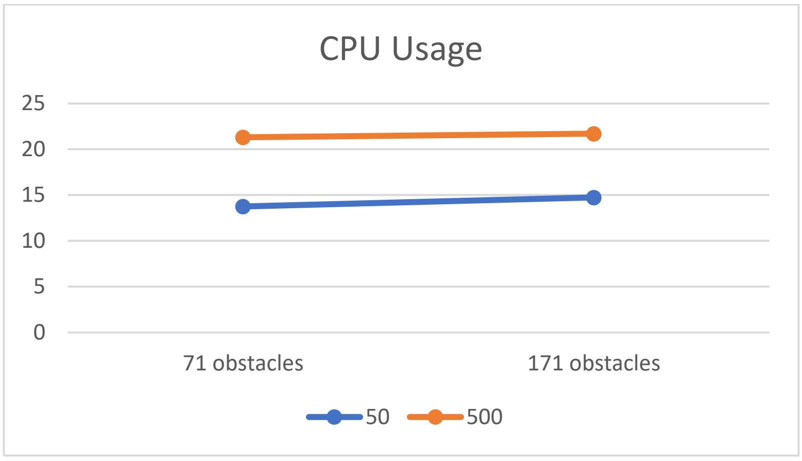
Hypothesis 1: Increasing the number of static obstacles is not expected to challenge much the game performance metrics.

Test results: It is found to be not correct. Even though the impact on metrics is minimal, the test results indicates that static obstacles contribute some load to the system.

Hypothesis 2: Increasing the number of agents is expected to challenge the game performance metrics while using static obstacles.

Test results: It is found to be correct. The test results show that increasing the number of agents significantly increases metrics due to the higher pathfinding workload for each of the agents.





To sum,

- Increasing the number of static obstacles from 71 to 171 had minimal impact on metrics.
- Pathfinding time for agents is consistent. It shows that static obstacles do not significantly challenge performance whereas agent count has increasing effect on metrics.

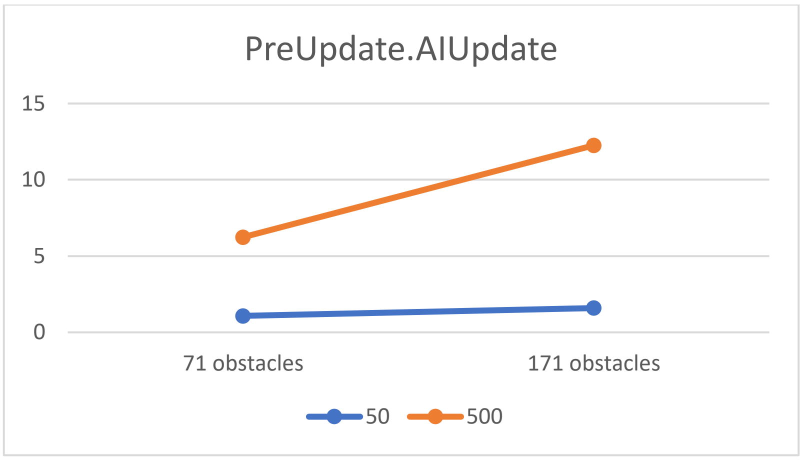
3.2. Dynamic Obstacles

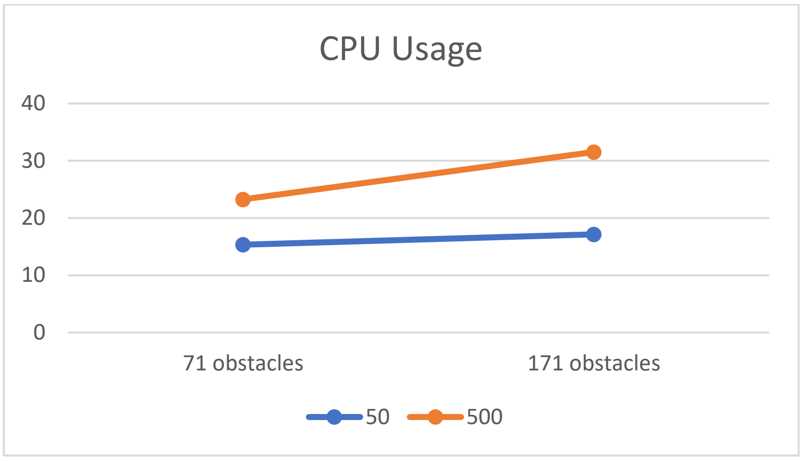
Hypothesis 3: Increasing the number of dynamic obstacles is expected to challenge the game performance metrics.

Test result: It is found to be correct. The results shows that dynamic obstacles introduce a noticeable increase on metrics due to frequent NavMesh updates.

Hypothesis 4: Increasing the number of agents is expected to challenge the game performance metrics while using dynamic obstacles.

Test result: It is found to be correct. The metrics are increased highly as the number of agents and dynamic obstacles increases as a result of requirement for more calculations.





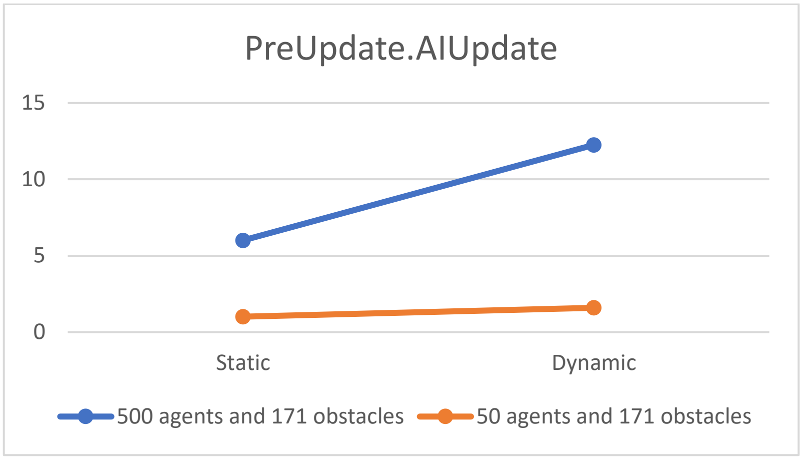
To sum,

- Increasing dynamic obstacles significantly increased metrics especially for when there are more agents.
- With 171 dynamic obstacles, agents experienced delays in pathfinding due to frequent NavMesh updates.

3.3. Static vs Dynamic Obstacles

Hypothesis 5: Static obstacles is not expected to challenge the game performance metrics as much as dynamic obstacles.

Test result: It is found to be correct. Dynamic obstacles had a greater impact on metrics compared to static obstacles especially for higher agent counts as it requires more calculations for pathfinding.



4. Optimization Techniques for Dynamic Obstacles

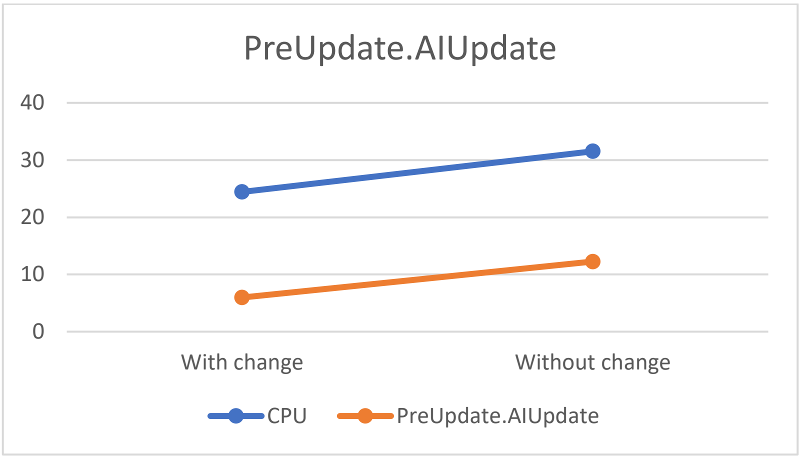
Dynamic obstacles in Unity uses NavMesh carving. It updates the NavMesh in real time to reflect the movement of dynamic obstacles. As it requires dynamic calculations, the process can result in computational load. In this experiment, two optimization techniques are tested to reduce the computational load caused by dynamic obstacles.

4.1. Changing the carvingMoveThreshold property of carving

The carvingMoveThreshold property is the minimum distance of a dynamic obstacle movement before NavMesh carving update is triggered. In this experiment this property is set to 1 instead of default 0.5 value. By increasing this threshold, the game reduces the frequency of NavMesh updates, as obstacles need to move further before carving is triggered and the movements of dynamic obstacles are limited in the experiment environment.

This change has the following effects:

- Since the frequency of carving operations is reduced, the computational load is reduced (+).
- Since the NavMesh updates are delayed, minor inaccuracies may occur in agent pathfinding (-)

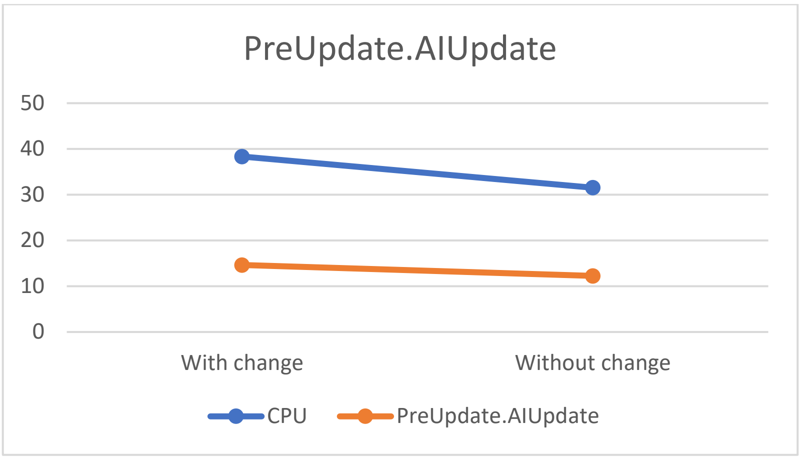


4.2. Chaning carvingTimeToStationary property of carving

The carvingTimeToStationary property is the duration a dynamic obstacle remains stationary before it stops carving the NavMesh. In this experiment this property is set to 0.5 instead of default 0.1 value. By increasing this time, the system takes longer to stop carving for stationary obstacles, reducing the frequency of carving updates.

This change has the following effects:

- Since there are no stationary obstacles in the experiment environment, delaying the recognition of stationary obstacles could not affect the metrics positively.



5. Conclusion

- Static vs. Dynamic Obstacles:
    - Static obstacles had negligible impact on performance because they are pre-baked into the NavMesh.
    - Dynamic obstacles caused frequent NavMesh updates and increases the metrics.
- Effect of Agent Count:
    - Increasing agents from 50 to 500 amplified the performance differences between static and dynamic obstacles, highlighting scalability challenges in dynamic scenarios.
    - In scenarios with dynamic obstacles, the performance difference was larger, as a higher number of agents increased the frequency of NavMesh recalculations.
- Optimization Techniques:
    - Only the first technique reduced computational overhead but had trade-offs as explained above.