

Games AI tech demo

Behaviour trees/Waypoint finder and “accidentally” added Steering Behaviours

General

This placeholder tech demo is an extension of Lab 1, which will implement a more deterministic pathfinder and examining the differences between the new pathfinder against the naïve pathfinder and focusing on the sophisticated movements that the AI agents can perform.

This demo is presented on “Tanks!”, a Unity Tutorial with modified scripts that implements AI agents with Behaviour trees through using the NPBehave library.

Video

<https://www.youtube.com/watch?v=Ke7egefMlts>

Behaviour trees

Using this bottom-up architecture, agent actions prioritises the most valuable action to the least valuable by placing the most critical action to execute at the top of the tree, down to the least valuable action.

Several complex actions were implemented, which is about to create 2^8 distinct combinations (2^9 if I split the pathfind Boolean into Naïve and NavMesh).

Implantation of agent

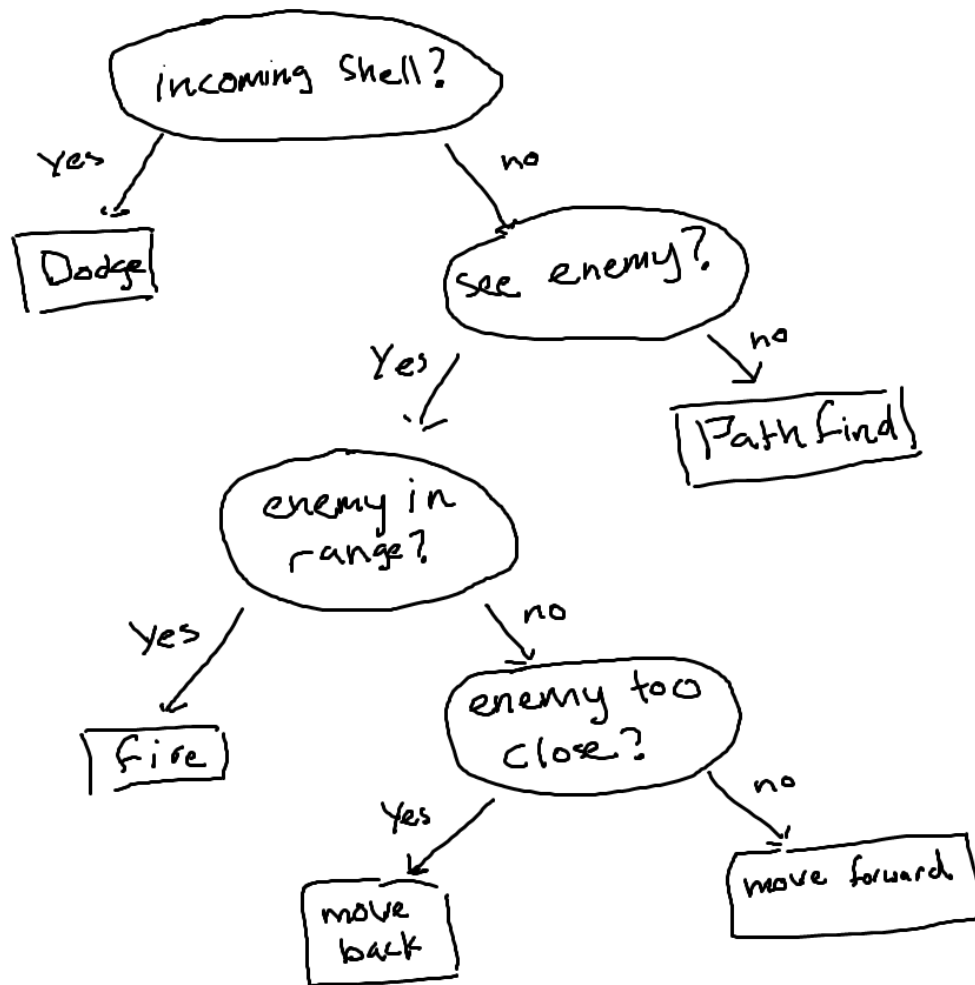
Each Boolean grants the agent permission to perform a set of action and adjusting its core attributes turn rate, speed and fire rate. Sight Distance is to determine how far the agent can shoot and dodge range is to determine when is the trigger distance for dodge on how close the shell is.

```
ClassMagna(float turnRate, float speed, float fireRate, float sightDistance, float backwardModifier, float dodgeRange, bool pathFind, bool dodge, bool defensiveDodge, bool counter, bool forwardMove, bool backwardMove, bool backwardFire, bool fire)
```

Breakdown of actions (Booleans):

- **Boolean pathFind, is it allowed to find the enemy?**
- **Boolean dodge, is it allowed to dodge shells?**
- **Boolean defensiveDodge, is the dodge movement inversed?**
- **Boolean counter, is it allowed to fire shells while dodging?**
- **Boolean forwardMove, is it allowed to move forward if the spotted enemy is too far?**
- **Boolean backwardMove, is it allowed to move backwards if the spotted enemy is too close?**
- **Boolean backwardFire, is it allowed to fire while “backwardMove” action is being triggered?**
- **Boolean fire, is it allowed to fire under a normal firing condition? (no prerequisite action needed unlike backwardFire and counter)**

Tree structure

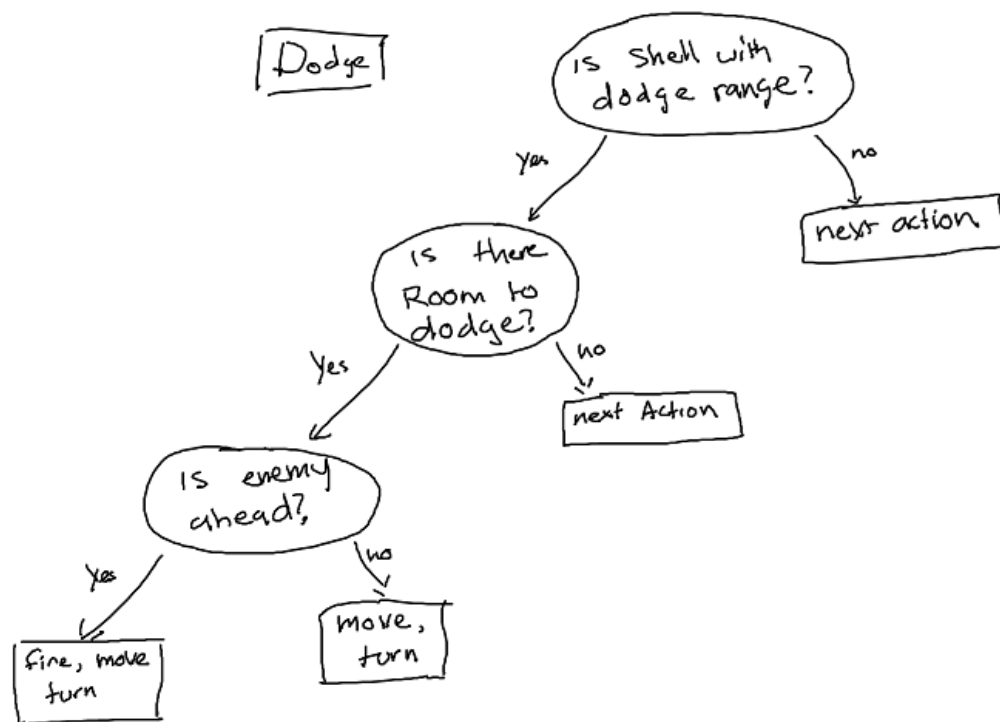


Agent's core structure

This is the core structure of the behaviour tree, which some actions can be disabled for the sole purpose of debugging and good programming technique and can generate multiple distinct agents using the same structure.

Only the dodge and countering function will be worth mentioning due to word count because of its sheer complexity

Dodging and countering (Steering behaviour)



Dodge.mp4



Counter.mp4

Core structure of dodging/countering and a Dodge.mp4

This is the main structure of dodging, where if the closest shell is within the agent's dodge range, it will trigger this event, then check its surroundings if there's an area that the agent can go to in order to dodge the shell, then checks if the enemy is directly ahead of the agent so it can perform a counter firing action.

It will either move forward or backwards depending if defensive dodge is enabled and it will try to turn at an angle which is perpendicular to the incoming shell and will select left or right depending which side has a smaller angle to be perpendicular and if the agent is able to move to that area through ray casting.

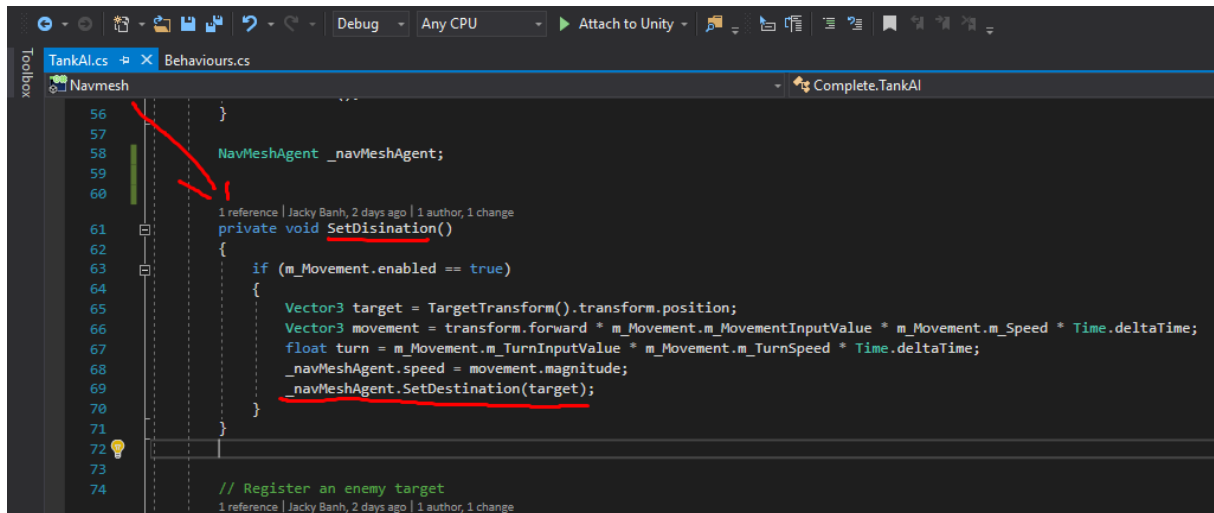
To examine this feature, use behaviour 6 (only dodge) or 4 (dodge and counter) or enable the second Boolean for dodging and fourth for countering when calling ClassMagna, the sixth float viable is used to edit the dodge range.

Waypoint finder

The two pathfinding algorithms that are currently implemented are Naïve and Unity's NavMesh in order to find and destroy the enemy tank.


Implementation:

Using the Unity's NavMesh Library, I was able to implement the new pathfinder by firstly by generating the static mesh from the game engine and then editing TankAI.cs script to implement the SetDestination() function and Behaviour.cs, to deploy the function.



```
56 }
57
58 NavMeshAgent _navMeshAgent;
59
60 1 reference | Jacky Banh, 2 days ago | 1 author, 1 change
61 private void SetDisination()
62 {
63     if (m_Movement.enabled == true)
64     {
65         Vector3 target = TargetTransform().transform.position;
66         Vector3 movement = transform.forward * m_Movement.m_MovementInputValue * m_Movement.m_Speed * Time.deltaTime;
67         float turn = m_Movement.m_TurnInputValue * m_Movement.m_TurnSpeed * Time.deltaTime;
68         _navMeshAgent.speed = movement.magnitude;
69         _navMeshAgent.SetDestination(target);
70     }
71 }
72
73 // Register an enemy target
74 1 reference | Jacky Banh, 2 days ago | 1 author, 1 change
```

Implementation of NavMesh on TankAI.cs



```
1 // Register an enemy target
2
3 /**Navmesh
4 0 references | Jacky Banh, 2 days ago | 1 author, 1 change
5 private Node NavMesh(float speed, float sightDistance, bool pathFind)
6 {
7     _navMeshAgent.stoppingDistance = sightDistance;
8     return new BlackboardCondition("pathFind", Operator.IS_EQUAL, pathFind, Stops.IMMEDIATE_RESTART,
9         new BlackboardCondition("inSight", Operator.IS_EQUAL, false, Stops.IMMEDIATE_RESTART,
10             new Sequence(
11                 new Action(() => Move(speed)),
12                 StopTurning(),
13                 new Action(() => SetDisination())
14             )
15         )
16     );
17 }
18
19 // Register an enemy target
```

Deployment of NavMesh on Behaviour.cs

How to use each pathfinding method:

```

644
645
646 //classMagna
647 /**dynamic modifications
648 9 references | Jacky Banh, 1 day ago | 1 author, 2 changes
649 private Root ClassMagna(float turnRate, float speed, float fireRate, float sightDistance, f
650 {
651     return new Root(
652         new Service(0.1f, UpdatePerception,
653             new Selector(
654                 /**dodge incoming bullets
655                 Dodge( dodgeRange, speed, dodge, defensiveDodge, counter),
656                 /**pathfinding
657                 new Selector(
658                     /**check if AI can see enemy,if it cant then crappy pathfind
659                     Naive(speed,pathFind),
660                     /**NavMesh(speed, sightDistance, pathFind),
661                     /**turns to enemy if not facing directly at it
662                     Turning(turnRate, speed)
663                 ),
664                 /**action
665                 new Selector(
666                     /**move towards enemy depending on sightDistance
667                     MoveToEnemy(turnRate, speed, sightDistance, forwardMove),
668                     /** move backwards if the enemy is too close
669                     MoveBackwards( turnRate, speed, fireRate, sightDistance, backwardModif
670

```

In “_Completed-Assets” folder, edit “Behaviours.cs” at line 659-660 to change the agent’s pathfinding method.

```

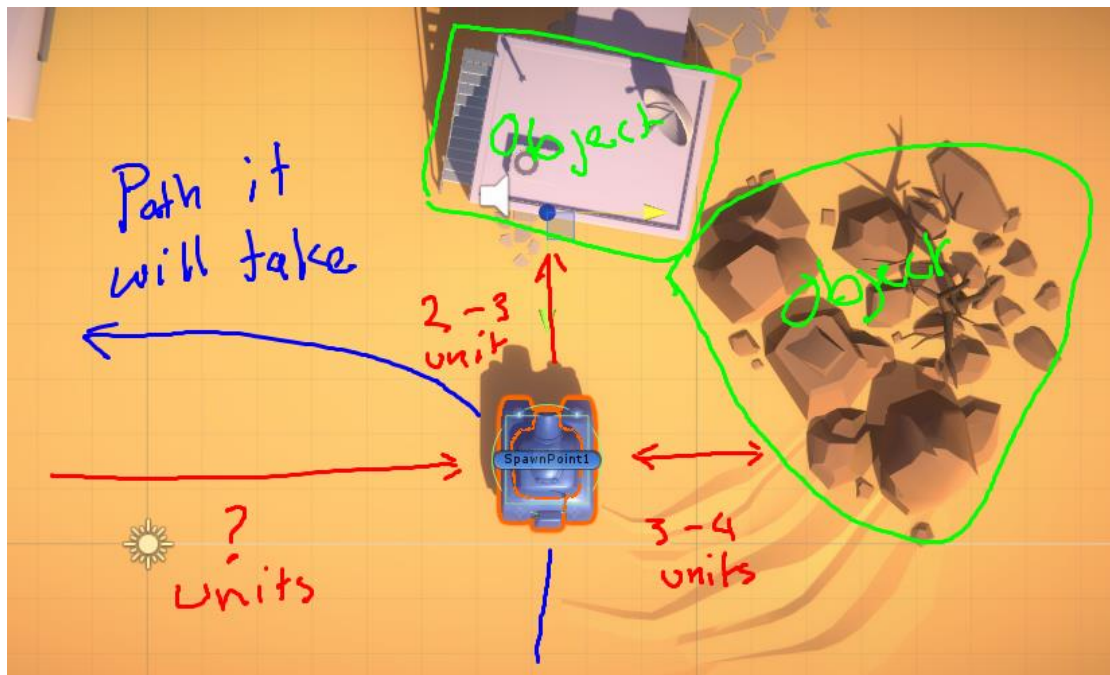
41
42
43 case 4:
44
45
46
47 /** debugging
48 /** dodge/ counter
49 return ClassMagna(1f, 1f, 0f, 40, 0.7f, 30, false, true, false, true, false, false, false, false);
50
51 case 5:
52 /** debugging
53 /** only fire
54 return ClassMagna(1f, 1f, 0f, 100, 0.7f, 15, false, false, false, false, false, false, false, true);
55
56 case 6:
57 /** debugging
58 /** test for dodging
59 return ClassMagna(1f, 1f, 0f, 40, 0.7f, 20, false, true, false, false, false, false, false, false);
60
61 case 7:
62 /** debugging
63 /** test for pathfinding
64 return ClassMagna(1f, 1f, 0f, 40, 0.7f, 20, true, false, false, false, false, false, false, false);
65
66 case 8:
67 /** debugging
68 /** case 1 no dodge
69 return ClassMagna(1f, 1f, 0f, 40, 0.6f, 12, true, false, false, true, true, true, true, true);
70
71 default:
72 return new Root (new Action(()=> Turn(0.1f)));
73

```

To solely test the pathfinding implementations, either use case 7 or if you are creating your own agent, you must set true for the first Boolean to enable pathfinding.

Naïve (Steering behaviour)

The Naïve approach navigates the agent randomly until it can “see” the enemy by using its surroundings.



Example 1: Naïve implementation

Example 1 shows the principle on how the method works by checking an object in front of it and checking which side has the furthest object, in this case, the AI agent picks left because there's “nothing” there.

There are a couple positive traits that this method has:

- **Low processing cost**
- **(Minor) Self-contained**
- **Can be applied to most common scenarios**

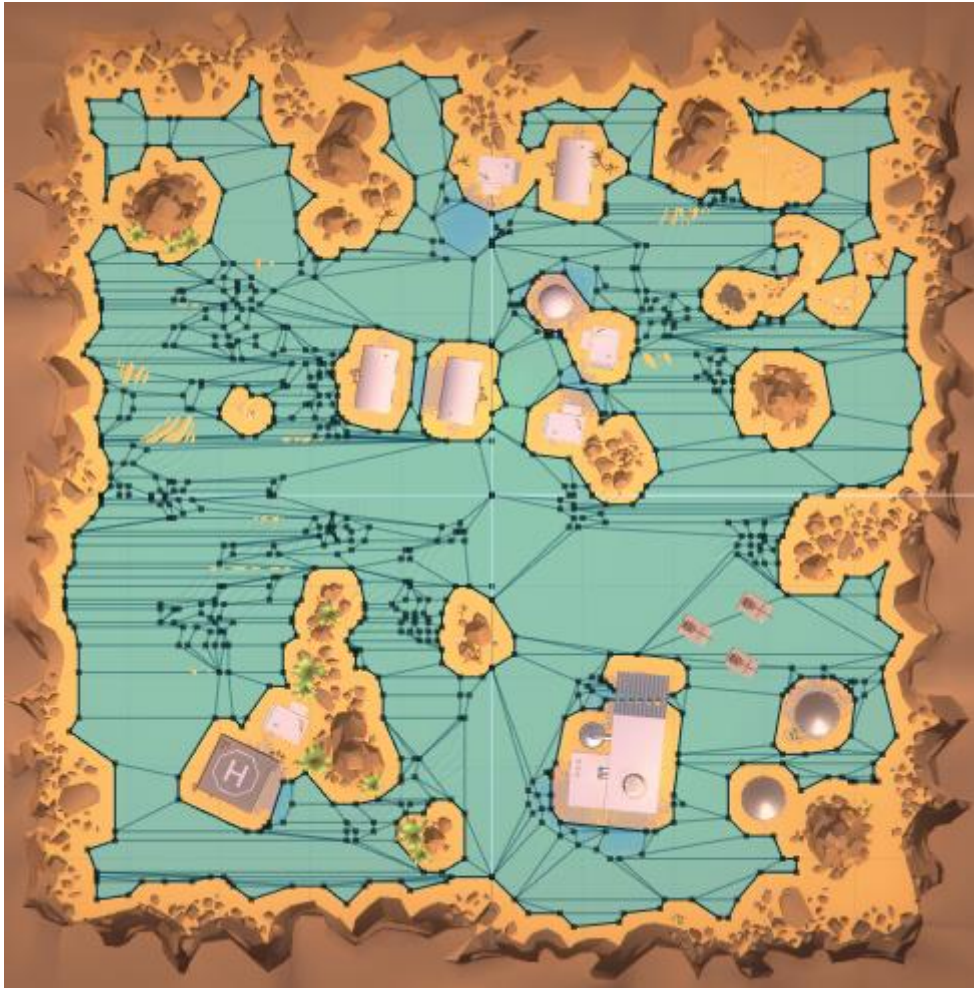
Negative traits:

- **Does not have a deterministic outcome (random)**
- **May take a very long route to reach the target**
- **May not even reach the target**
- **(Minor) looks dumb/stupid to the player**

Given enough time it can eventually reach the target, however it can be very impractical to deploy this method.

Navigation mesh

Creates a navigation mesh for the agent to find a route to reach the target.



Example 2: Unity's Navigation Mesh

Example 2 shows the Unity's NavMesh being generated by creating an area that allows the agent to transverse to then puts several waypoints and join them up as polygons. This Mesh feeds the information to the agent so that it can pick a appropriate route to reach the target. This method has several improvements than waypoint graphs because when deploying a pathfinding algorithm, any spot on a ploygon can be assumed as a node and this also gives the agent more of a "fluid" movement to the target.

Advantages:

- **Creates a deterministic route**
- **(Minor) Looks smart/human-like to the player**

Disadvantages:

- **High cost**
- **Requires new mesh for every new terrain**

Evaluation - Waypoint finder

To evaluate how optimal a pathfinding algorithm is, it must be compared to a core example, in this case, the naïve implementation is used to demonstrate the “base” route of this terrain against other pathfinders which does a better job.

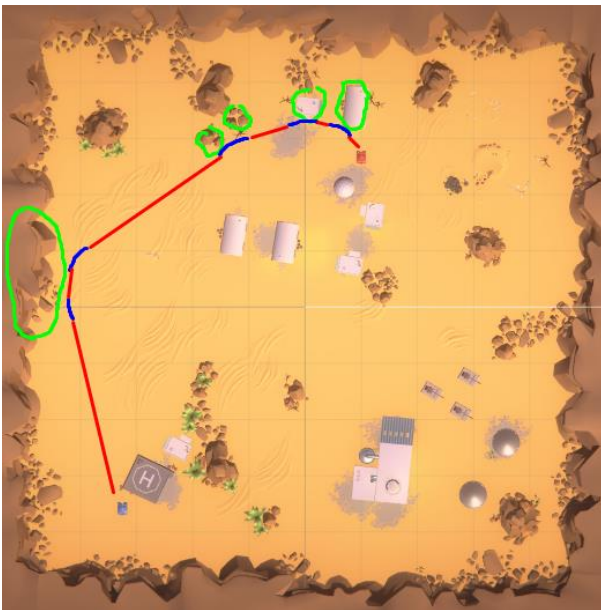


Naive.mp4



NavMesh.mp4

Shows the paths that the two algorithms took, Naïve.mp4 and NavMesh.mp4

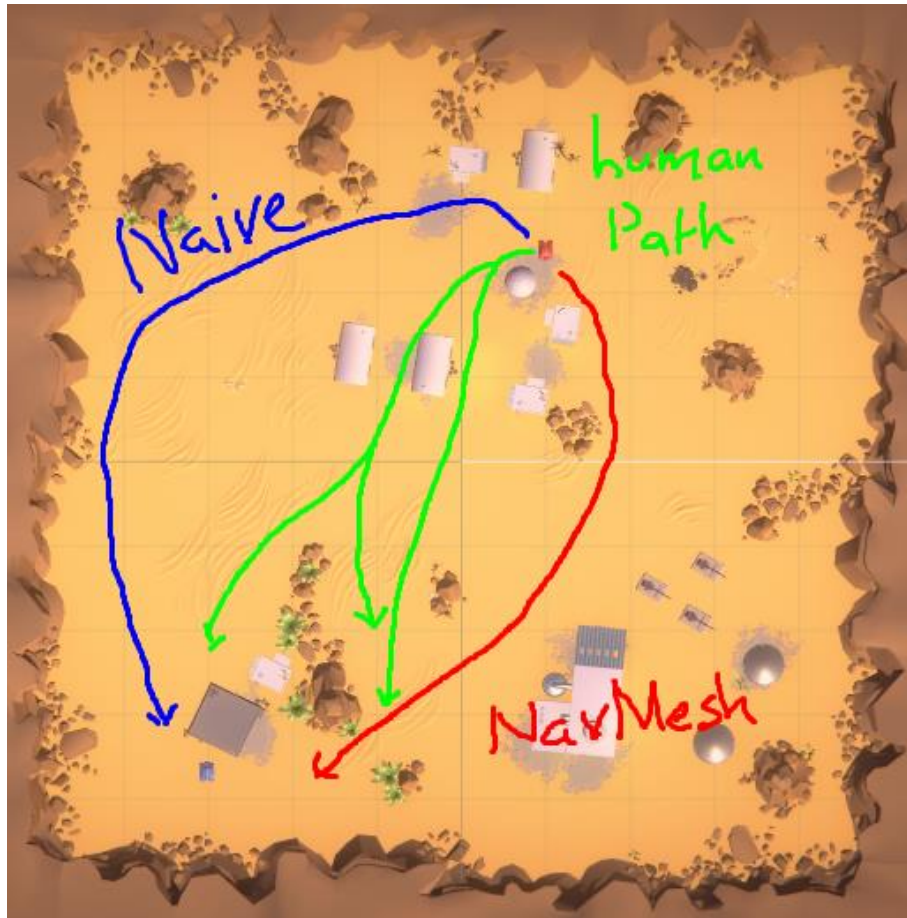


Example 3: The pathfinder's routes

The terrain objects happen to guide the naïve approach to the target, but took a significantly long route that it even exceeded the Gyazo's time limit.

The NavMesh approach took an optimal route to reach the target and with a fluid motion.

Potential interest to gaming industry



Example 4: Human-like paths

By observing the common “human” paths, you can see a contrast between the two pathfinding algorithms, where the NavMesh implementation is more human-like than Naïve. Although implementing NavMesh has more disadvantages than Naïve, it has more appeal to the gaming industry, where the entire concept of an AI agent is to portray itself as a “real human” player.

In terms of the gaming industry, the two algorithms display the both ends of the pathfinding spectrum, where one is dumb and random, and the other is smart and deterministic, but the ideal solution is to create a pathfinding algorithm that produces be suboptimal results along with a slight randomness to appear more “human like”.