AI Maze

23.06.2023

Gavin Lampe

Changelog

Version	Date	Changes
1.0.0	20/06/2023	Initial Setup
2.0.0	23/06/2023	Final Version

Contents

Introduction

In the TDD. Which namespaces (Includes) did you include in your project, What functionality did each namespace provide to your code.

Rationale

The goal of this project is to demonstrate the uses of Unity components such as navmeshes and animations and coding techniques such as state machines in creating NPC AI agents that can navigate a dynamic environment with doors and different navigation areas.

Background

As the purpose of this project is to demonstrate the multiple ways navmeshes and state machines can be used, there are a number of requirements:

- a maze with doors that open and close altering the paths
- navmesh surfaces on the maze for the different agents
- three AI agents that must navigate the maze in different ways to find different objects
- different areas in the maze that affect the agents in different ways
- nav mesh links to demonstrate the agent jumping or moving from one platform to another
- collectable objects for the agents to find
- a state machine allows the agent to determine which object or destination to go to next.
- Animated characters that change their animation depending on their state

Terminology

Agent – A computer controlled character in the game that moves around the environment.

NavMesh – A unity component for mapping out the areas of an environment where an agent is able to move. The AI uses the navmesh to plan a path.

NavMesh Links – A way of joining two different NavMesh surfaces together so that an agent can go between them. For example, jumping down from a higher floor.

Area Modifier – An area modifier allows you to change part of the navmesh for one or more agents to create different terrain. For example, you make make the agents slow down when they are running through mud.

State Machine – A system for deciding what an agent should be doing now (what state they are in) or what action they should take next.

Proposed Design

First, a maze will be required will moving walls to create altering paths, and some areas that can have area modifiers attached to change how the agents move across them or are blocked. Steps over walls of differing height will be created as well as stepping stones across water, a pit to jump across, and sand traps to slow down some agents.

The project will require 3 agents to be created with different models to differentiate them and animations for standing idle, running, and jumping. The agents will be:

- A normal agent who can climb some stairs but runs at a medium pace and cannot jump far.
- A short agent who cannot go up stairs or jump far but can run really fast.
- A jumping agent who runs at a normal pace but can climb all stairs and jump across pits and sand traps.

A state machine will be needed to decide which point each agent runs to next and which animation to trigger for each state. States will include idle, move to, dance, door wait, and freedom for when they finally exit the maze.

Goals will be required for the agent to pursue in different orders. These will be grouped as 'treasure', 'keys', and 'doors' with the goals and order randomly assigned to each agent, except for the doors which are the final destination unless they haven't found a key yet (in which case that becomes the destination before returning to the door).

Non-Goals

It would be nice to make the agents spawn in a random location and have the stepping stones moving to give more of challenge and see which agent gets out first. In that respect, it would also be nice to have some win animations and focus on the agent that got out first or some punishment for the agents that failed.

Software and Hardware Requirements

Unity Game Engine 2021.3.23f– Personal Licence – Free (not a commercial project)
JetBrains Rider 2022.3.2 – Student Licence – Free
Visual Studio 2022 – Student Licence – Free
Mixamo Animations – Free under licence from Adobe (You must have an Adobe account)
Synty Polygon Assets - ~$2 the assets used purchased as part of a bundle with full unrestricted commercial licence to use and modify the assets.

The project is made for use under Windows as that is the test environment available.

Minimum Hardware Requirements
Processor: Dual-Core 1.5ghz processor
RAM: 1GB
Graphics: 512MB Video Memory – DX9 Compatible
Storage: 100MB
OS: Windows 10 or above

System Architecture

Data types

- **Integer** – Used to store whole numbers. In this project used for things such as array indexes
- **Float** – Used to store numbers with a decimal place – used for various speeds and heights
- **String** – Used to store a group of alphanumeric characters – used to change GUI button text
- **Vector3** – Used to store 3 floats as X, Y, Z co-ordinates – used to store various game object positions and rotations (as Euler angles)
- **GameObject** – A data type for storing Gameobjects which are the base entities of a Unity scene
- **Transform** – A data type that corresponds with the Transform component and contains a GameObject's position, rotation, and scale.
- **List** – A list is an ordered collection of elements which can be any data type. In this project it is used to store the goals each agent must reach.
- **Array** – An array is a collection similar to a list except that it has a predefined size, making it less flexible but more efficient. It's used to store the list of doors
- **Bool** – A true/false data type. Used in this project to switch things on and off or change between left and right (when changing camera positions)
- **Enumeration** – An 'enum' is a set of named constants. That means, similar to variables, they are values with names but those values don't change. In this project the value is irrelevant as the constants are simply used to represent the different states in the state machine.

Interface/API/Namespaces Definitions

The components that will be referenced are:

GameObject
Transform
Canvas
Button
TextMeshPro (No official API documentation as it is still a package)
NavMesh
NavMeshAgent
OffMeshLink
Animator
Collider
SceneManager

Which namespaces (Includes) did you include in your project?

Microsoft C# namespaces:

System.Collections
System.Collections.Generic

Unity namespaces: (Unity API Manual - namespaces don't have direct links)

UnityEngine
UnityEngine.AI
UnityEngine.Random
UnityEngine.SceneManagement
TMPro

What functionality did each namespace provide to your code?

System.Collections provides the IENumerator functionality which allows co-routines. I use co-routines to make the doors open and close.

System.Collections.Generic provides the functionality for the List data type. There are two List variables both storing Transform data types, goalList and keyList, which store all the goals and all the keys respectively

UnityEngine provides all the main functionality for accessing and manipulating Unity game objects. It contains all the basic classes Unity needs to function, and without it the code won't run.

UnityEngine.AI provides the NPC AI functions in the game. It provides the classes for accessing and using the Unity AI components NavMesh and NavMeshAgents, among others, in our code.

UnityEngine.Random allows random number generation. This is used in the code to create chance in the objects that the agents have to pick up.

UnityEngine.SceneManagement allows us to work with the scene files, or scene *assets*. Here this namespace gives us the functionality of reloading the scene to start the demo again.

TMPro is the TextMeshPro namespace and contains the classes that allows the code to work with TextMeshPro assets in the Unity UI. In the scene, this is used to switch the text on a GUI button.

Risks

The biggest risk is trying to implement moving platforms as it may not be possible for the agents to find a reliable path across them to their destination. Research will have to be done to find out if this is at all possible.

Alternatives

An initial attempt to use Unity's built-in navmesh component was abandoned due to it's limitations and replaced with an experimental plugin that was actually advised to be used initially, but that good advice was ignored.

Pseudocode

System Pseudocode

AgentState (State machine for agents)
State struct storing state variables Idle, MoveTo, Dance, DoorWait, Freedom

First, set state to Idle
Then set up agent by:
Randomly assign treasures, keys, and an exit to agent
Add the positions of these objects to a list of waypoints
Set which OnFind… method to run when the waypoint is reached is found based on destination type.
Set state as MoveTo to move to first waypoint and set animation state to running.
Each frame check the current state and go to the appropriate method.

Idle
Make sure running animation is off
Trigger idle animation
If idling too long, trigger extended idle animation
If the path to the destination is not working, remain idle until it is
Once a new destination or path has been found, change to MoveTo state and start running animation

MoveTo
If the path is not working, switch to the idle state.
If distance to destination is ~0, stop, then run the correct OnFind… method for when a destination of this type is found (key, treasure, door)
Else keep running but if the agent hits a sand trap lower its speed or make it jump depending on the agent otherwise it's at normal speed

Dance
Play the dance animation and wait for it to finish
Set the next way point and change the state to Move To
Make sure the animation is set to running.

DoorWait
When agent reaches a door, wait until the door is open before moving to next way point.

Freedom
Stop the agent, play its freedom animation

OnFind…

When an agent reaches a treasure, disable the mesh and play the dance animation

When an agent reaches a key, if it has already gone to that door then return to that door. Otherwise add the correct door for that key to the list of destinations.

When an agent reaches a door, if they don't have the correct key, add the key to the list of destinations, and set them moving to that destination. Otherwise, wait for the door to open before moving to the final destination.

Goals

First, get the list of gameobjects for each type – treasure, keys, doors
Then split the treasures evenly and randomly between the agents.
Create a list of Transforms for those treasures to store their positions
Randomly decide if an agent gets to find a key before trying a door and if so, add it to the list
Randomly choose one of the two doors to add to the end of the list.

ExitDoors

Check if the agent has the key, if they do run a coroutine to open the doors.

Sliding walls and rotating/pivoting walls

Slide a wall from its original position to a new position pause and then move back. Do the same for rotating walls but make it rotate 90 degrees before pausing and rotating back.

Evaluation

Reflection

My performance on this project was pretty awful. I've realised that I don't work well alone (I'm much better with the inspiration of others) and that technical documentation definitely ain't my jam! I also was stumped for more than two days trying to think about how to justify the use of namespaces in such a small project before I realised the reference to them in the requirements was simply to detail the built-in Unity namespaces used and not our own.

I have learned a lot about AI and navigation in a game and the use of state machines, and I hope to expand on them further to understand the different and more complex ways they can be used to create more realistic NPCs.