Unity AI Programming Essentials

Use Unity3D, a popular game development ecosystem, to add realistic AI to your games quickly and effortlessly

Foreword by Bill Klein (aka Prime), CEO, Rival Theory

Curtis Bennett  Dan Violet Sagmiller
In this package, you will find:

- The authors biography
- A preview chapter from the book, Chapter 1 “Pathfinding”
- A synopsis of the book’s content
- More information on *Unity AI Programming Essentials*

### About the Authors

**Curtis Bennett** has been a developer in the games and computer graphics industry for several years. He has worked on developing immersive virtual environments, published research in visual simulation, taught college courses in game development, and worked for various game studios, and he was also an engineer on early versions of the RAIN AI plugin for Unity. Currently, he is the Technical Director for Creative Services at Ideum, which focuses on creating interactive media projects. I’d like to thank all the Unity AI plugin developers who make implementing AI with Unity so easy.

**Dan Violet Sagmiller** has always had a strong passion for game development, although most of his work leads him to senior business development roles. He developed several games, including Teams RPG, a space shooter (which won a game development competition at Technology Center of DuPage), and some casual games. He started teaching game development at Heartland Community College in 2005, taking over the existing course and expanding it to six courses. Later, he took a position with Microsoft and expanded the game development curriculum for Bellevue College, including classes on AI, physics, testing and designing using C# and XNA. Later, he moved on to a senior position with Wizards of the Coast, where he also taught game development and AI internally. He also has given talks at multiple schools about getting into game development and programming as a career. More recently, he released a game development book on C# and XNA, which had 2,000 downloads in the first week. He also runs Learn Build Play, a small private school dedicated to teaching game development and design mostly with C# and Unity 3D.

He can be contacted at Dan.Sagmiller@LearnBuildPlay.com.
Unity AI Programming Essentials

Welcome to *Unity AI Programming Essentials*. This book will guide you through all the skills necessary to put realistic game AI into your Unity games. We won't be spending much time discussing AI theory or how to implement popular AI algorithms from scratch. Instead, we will take the more efficient approach of using third-party Unity AI plugins to set up AI for your games easily. We will cover all the essential game AI skills, such as pathfinding to have your characters navigate a game scene, behavior trees to let them "think", and sensors so that they can react to their environment. We'll also cover more specialized tasks such as setting up crowds and cars for driving and integrating animation. By the end of the book, you should know all the basic skills you need to create game AI with Unity.

What This Book Covers

*Chapter 1, Pathfinding*, covers how to set up basic pathfinding so that game characters can navigate a game scene realistically.

*Chapter 2, Patrolling*, extends our pathfinding to have characters patrol routes in a scene.

*Chapter 3, Behavior Trees*, explains behavior trees and how they are used to give AI characters the ability to make decisions.

*Chapter 4, Crowd Chaos*, focuses on creating a wander behavior that can be used to create ambient crowds.

*Chapter 5, Crowd Control*, demonstrates using specialized crowd plugins how to generate groups of AI characters.

*Chapter 6, Sensors and Activities*, shows how to set up sensors and have AI characters change their activity based on what they sense in the environment.

*Chapter 7, Adaptation*, shows how to have AI characters react and adapt to different events in the game.

*Chapter 8, Attacking*, discusses different techniques to have an enemy AI attack the player.

*Chapter 9, Driving*, shows how to set up driving AI that uses a car's physics to generate realistic driving behavior.

*Chapter 10, Animation and AI*, discusses how to integrate character animation with AI using both Unity's legacy and Mecanim animation systems.

*Chapter 11, Advanced NavMesh Generation*, discusses more advanced options for setting up navigation meshes to handle mesh creation for different geometries and multiple NavMeshes in a scene.
Pathfinding

Probably the most useful game AI is pathfinding. Pathfinding is all about making your way from one point to another while navigating around obstacles. Unity excels at linking the worlds of designers and programmers. AI is no different, and we'll see how to make simple pathfinding AIs without ever touching the code. Pathfinding is probably the most common AI game task, and there are many Unity plugins for it. We will be looking at three different ones.

In this chapter, you will learn:

- Working with pathfinding
- Applying pathfinding in Quick Path, React, and RAIN AI packages
- Behavior trees
- Applying characters to Character Controller
- Unity's NavMesh

An overview

Pathfinding is a way to get an object from point A to point B. Assuming that there are no obstacles, the object can just be moved in the direction of the target. But the AI part of it is all about navigating the obstacles.

A poor AI might try walking a **Non-Player Character (NPC)** directly to the target. Then, if it is blocked, it randomly tries to go to the right or left to look for a space that might help. The character can get caught in different areas and become permanently stuck.
Pathfinding

A better AI will walk an NPC in an intelligent way to a target, and will never get stuck in different areas. To compute a good path for the NPC to walk, the AI system will use a graph that represents the game level, and a graph search algorithm is used to find the path. The industry-standard algorithm for pathfinding is A* (A Star), a quick graph search algorithm that uses a cost function between nodes—in pathfinding usually the distance—and the algorithm tries to minimize the overall cost (distance) of the path. If you want to learn to code your own pathfinding AI, try A* because it is simple to implement and has a lot of simple improvements that you can apply for your game's needs.

The AIs we are about to discuss will take care of the pathfinding algorithms for you, and ultimately reduce the time it takes to breathe AI life into your game. Now let's look at Quick Path AI.

Quick Path AI

Alekhine Games' Quick Path is a $10 AI that you can pick up from the Unity Asset Store. Although the next two AIs have more features, this AI is added because of its blocky nature. This block approach creates a grid-based path and is used with many types of games, but this AI works especially well with the excitement in the voxel game genre; it is suited for cubed topography.

To start with, perform the following steps:

1. Create a new 3D scene and import the Quick Path AI from the Asset Store.
2. Next, set up some cubes, planes, or other objects as your terrain, and then place all of these game objects into an empty game object. Name this game object Terrain.
3. Next, on the Inspector panel, add a component, QuickPath | Grid. Immediately, you should see a series of blue lines that show up on the cubes. These indicate all the points where a character can move in the AI.

![Image of Quick Path AI](image)
4. Now, we need a character to move around the scene. Create a sphere, or any object, and name it NPC.

5. Then, we'll add a Component, **QuickPath | AI | Follow Mouse Object**.

6. Now, when you run the scene, assuming it is lit up and has the camera pointing where you want it to, you'll see NPC on Terrain.

7. Click somewhere on the Terrain object, and watch the NPC object move to that point.

8. Although we might say that the pathfinding in this is clearly working, we should also add an obstacle to the scene: something that shouldn't be stepped on. To do this, add another cube somewhere. Go to the Inspector panel for the obstacle and tag it with Obstacle by selecting that tag from the drop-down, or if it is not an option select Add Tag... and add Obstacle to the tag list.

9. Next, in the Terrain game object, in the Grid component, expand Disallowed Tags, increase the size to 1 and enter Obstacle for the new element.
Pathfinding

10. Next, click on the Bake button at the top of the Grid component. Now you will see that the grid markers skipped the cube as an option. If you want to test more, click somewhere else on the Terrain object and watch the NPC object move to the clicked point avoiding the obstacle.

Now, we've seen how to set up pathfinding with Quick Path, so let's look at another way to set up pathfinding with React AI.

**React AI**

Different Methods' React, a $45 AI, introduces a behavior tree and the use of a navigation mesh, or NavMesh. A NavMesh is a series of interconnected polygons forming a complex area used for travel. It creates a simplified graph of the level that is inputted into the pathfinding system. This simplified graph that it creates is smoother and tends to have characters that travel better than a grid-based graph. A behavior tree is a parent-child structure used for making decisions in many AIs. We will look at behavior trees and navigation meshes in more detail in the later chapters. NavMesh is a basic feature available in Unity, but the behavior tree is not. Unlike the other two AIs shown, this AI requires a bit more coding to get started, but not much.
To begin with, you'll need a new scene, as well as to import React AI from the Asset Store. Perform the following steps:

1. Add a plane or another ground type. Then add several obstacle objects, such as cubes. Make sure that each of the objects we just created are marked static at the top of the Inspector, or the NavMesh won't identify them later on. The scene should look like this:

![Scene with plane and obstacles](image)

2. Next, find the **Window** menu and select **Navigation**. At the bottom of the **Navigation** tab, click on the **Bake** command. You have now generated a simple navigation mesh for your characters to navigate. It will highlight the areas that NavMesh AIs can walk, as seen here:

![Navigation mesh](image)

3. Let's add a player who can move around the world now. Add a capsule and name it **Player**. Fortunately, the demo contains a simple script for controlling a player who you can find (and add) by navigating to **Add Component** | **Scripts** | **Simple Player Control**. Now, this doesn't move the object around on its own; instead it drives a **Character Controller** object.
Character Controller is a type of an object that you can inherit in your code classes that many AIs can operate. In this case, there is a basic Character Controller type to simply move a given object around.

4. When adding the component, just start typing Character Controller in the search box, and it will show you all the similar component names. Add Character Controller. Now, the player should be controllable. You will probably need to increase the speed to 1 to detect the player movement.

Make sure that the game object, and any body parts, do not have collider components. Controllers detect colliders to determine whether or not they can move to a given place.

5. Next, we’ll add an enemy in the same way, with Capsule. The enemy needs a component called Nav Mesh Agent, which is a component capable of using a NavMesh to move around, so add it. Now, the game object has the ability to walk around, but it has nowhere to go. To get it moving, we need to add the enemy AI agent.

6. Next, we get to the AI for the enemy agent. In React, a behavior tree is called a Reactable. To add a reactor, we start the Project explorer, in a folder of our choice, by navigating to Create | Reactable.

7. Once created, rename it to EnemyMovement. In the Inspector, it has a list of behaviors for it. We’ll need to add a script, which can be found in the book's contents: \Scripts\React AI\FollowThePlayer.cs. Without going in-depth in the code, let me explain the following key points:
   - The C# file was copied from a sample script provided with React AI that made a character move away from a target.
   - It was rebuilt to make the player the target destination, and also to turn seeking on and off by using a button. It is not hard to adapt these scripts.
   - Unlike normal mono behaviors, you use a special Go method. The go method is called by React AI only if it is selected to be used.
   - In the Start method, we see it obtain the NavMeshAgent that we attached to the enemy in the Inspector panel.
   - In the Go method, we see it feeding the destination to the NavMeshAgent, and then checking to see whether it has already found a path. Once it does, it just goes.
All uses of that agent are still following standard Unity calls to use NavMesh, and can be applied without using the AI, by placing this code in a traditional behavior Update method.

This script needs to be added to the Inspector for the EnemyMovement asset, and also to the Enemy game object.

8. Once the script is attached to the enemy, the Inspector will reveal that it has a target. Drag the player from the Hierarchy panel into the player attribute on the Inspector panel.

9. Finally, we have the behavior tree to set up. In the Project panel, right-click on the EnemyMovement asset, and click on Edit Reactable. A behavior tree pops up an editor, which is how we train our AI.

For this chapter, we'll just give it a one track mind to follow the player. With Root selected, click on the Action button under Leaf, as shown in the following screenshot:

Since we only have one action in the behavior list, it selects it by default. What makes the behavior tree nice is that we can make decisions, or check whether the target is within X distance then try to follow, otherwise do something else—all from the designer. The next section on RAIN also uses a behavior tree, and most of the same basic types are used in both RAIN and React.

This took more steps than the previous AI, but there is also more going on. It is playable now.

---

**Downloading the example code**

You can download the example code files for all the Packt Publishing books you have purchased. If you purchased this book elsewhere, you can visit [http://www.packtpub.com/support](http://www.packtpub.com/support) and register to have the files e-mailed directly to you.
**Pathfinding**

**RAIN AI**

Rival Theory's RAIN AI is a very full-featured AI to use and it is free. It includes a behavior tree with similar functionality to React, but has more features built in. In fact, this one won't require any additional scripting to go from point A to point B.

To get this going, we'll need the following bases:

- A map to move around on
- A character to move around in the map
- A route (series of waypoints) for the character to follow
- A navigation mesh that knows what to avoid
- An AI to control the character
- A behavior tree to tell the AI what to do

To start with, you'll need to start a new Unity project, import Unity's Character Controller, and import the RAIN AI package.

Don't get the RAIN AI package that is found in the Asset store. The current release (at the time of writing this book) can be found at the Rival Theory site, rivaltheory.com/rain/.

Perform the following steps:

1. To create our map, add a plane. Then, add a couple of objects to act as obstacles. It is best if they create a \( U \) or \( V \) shape to potentially trap the player:
2. Next, drag the predefined character found in Project | Standard Assets | Character Controllers | Sources | Prototype Character | "constructor" into the scene. From the scene depicted in the preceding screenshot, I recommend placing him (the character) on the back left-hand side of the plane.

3. Next, we need a route. Although there is more than one waypoint system in RAIN, I found that the route will be the fastest for this demo. In the RAIN menu, click on Create Waypoint Route. Name its game object GreenPath. We will need to call this later, so we want simple, easy names to remember.

4. In the Inspector panel for GreenPath, click on the Add (Ctrl W) button. This adds a waypoint. In fact, we need to add three. Place the first one on the inside of the V, the second on the tip of the V, and the last on the far edge of the plane, as shown in the following screenshot:

5. Just as in React AI and Unity NavMesh, we need a navigation mesh for this as well. We clearly defined the route, but as you can see, the path is blocked. In the RAIN menu, click on Create Navigation Mesh. Align and stretch it so that it surrounds the area where you want the paths to be determined.
The Inspector panel for the NavMesh has a property called **Ignored Tags**. Add **Player** to this list. (You might need to make sure that the player object actually has that tag selected as well.) Otherwise, the NavMesh will not generate where the player stands, and will prevent its ability to find a path. Then click on **Generate Navigation Mesh** at the bottom of the **Inspector** panel. The result should look like this:

6. Next, we need to add an AI to control the character. Select the player object in the **Hierarchy** panel, and from the **RAIN** menu, click on **Create AI**.

7. Next, select the middle button of the character running in the **AI Inspector** panel, then click on the **Add Existing Animations** button at the bottom. This will add all the player’s animations: idle, walk, jump, pose, and run. You can refer to the following screenshot:
8. Next, we need to add a behavior tree. Behavior trees are a way to define decisions and actions for AI characters. We will discuss them in more detail in Chapter 3, Behavior Trees. For now, add one by clicking on the head/brain icon in the Inspector panel and then click on the Open Behaviour Editor button. On the right-hand side is a behavior tree drop-down selector, so click on it and choose Create New Behaviour Tree.

9. Name it FollowGreenRoad. It will already have one element, SEQ (Sequence), under the root BT (behavior tree) node. Sequence it means that it will run any child nodes in order. Right-click on the SEQ node and navigate to Switch To | Parallel, which means that it will run all its child nodes simultaneously.

10. Let's add the child nodes and then set them up. Right-click on the PAR node, then navigate to Create | Actions | Animations. Right-click on PAR again and navigate to Create | Actions | Choose Patrol Waypoints. Then right-click on the new WAY node and navigate to Create | Actions | Move.

Because the decision is to run things in parallel, it will animate the character and follow the waypoints at the same time.

11. Click on the green animate node, and set its animation state to walk. It is case sensitive, but you can select which animation the character should use.

12. Next, select the WAY node. Here, you need to set Waypoint Route to use. This was the navigation route we created earlier with the three waypoints. We called it GreenPath.

13. For the loop type, we'll make it One Way so that the character only travels to the end and stops there. Also, change the name of the loop to Follow Green Path. This shows up next to the WAY node, and helps explain what is happening.
14. Finally, set the Move Target Variable to **NextWayPoint**. This is a variable that we are setting with the next waypoint in the path. When it is reached, the patrol route will set the variable to the next location in the path. We use this in the Move node.

15. Select the *move* node, and in the properties, set the **Move Target** to **NextWayPoint**, the variable that is being set by the patrol route we just configured. And set the **Move Speed** to a number, such as 3. This is how fast the character will move.

16. Now that we have created the behavior tree, we need to set the Character AI to use it. Select the AI object under the player object in the Hierarchy panel. On the Mind icon, for the **Behavior Tree Asset**, set it to **FollowGreenPath**. This can be found by navigating to **Project | AI | Behavior Trees**, or from the selector in the **Inspector** panel, choose the **Assets** tab, and it should be right on top.

The demo should be able to run now. The character will move around the block and walk to the last waypoint in the path.

### Comparing AI solutions

Each AI has its own strengths and weaknesses, ranging from price to flexibility to designer friendliness. Also, each AI has more than one way to accomplish this chapter's task of moving a character from point A to point B. We selected paths that were faster and easier to start with, but keep in mind that each of them has plenty of flexibility. All three proved to work well as terrain/trees as well as simple planes and cubes.
Experiences of working with all three:

- Quick Path is a good choice for a beginner. It has the fewest steps to do to get going, and works easily. Quick Path is focused on just pathfinding and the others are larger AI systems that can be expanded to many more areas because of their use of behavior trees.

- RAIN has many features beyond pathfinding that we will discuss in future chapters. The learning curve for RAIN is higher than Quick Path and unlike other AI solutions, the source code is unavailable, but it is a good all-round solution for game AI. And while RAIN has the ability to be customized through user-defined scripts, the focus is on easy AI setup through the Unity GUI without needing to write scripts often.

- React includes a behavior, but requires more code to get it running, which is good if you are interested in coding more. You build all the actions it can use, and let the designers focus on the tree. RAIN can do this too, but with React, you are building the blocks from square one.

Overall, the best AI for you is the one best suited for your game and that you enjoy using. We will be looking at these three and other AI systems in detail throughout this book.

**Summary**

Our AI characters need to be able to move between different points in our scene in an intelligent way, and we looked at pathfinding AI systems that helped us do that. We tried three different ones: Quick Path, React, and RAIN. But our characters need to be able to do more than just walk from one point to a second one in our levels. In the next chapter, we will extend what we have learned about pathfinding here by seeing how to set up patrolling behaviors for our characters. This will be the start for having characters walk around a level in a realistic way.
Where to buy this book

You can buy Unity AI Programming Essentials from the Packt Publishing website.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

Click here for ordering and shipping details.