Unreal Engine 4 AI Programming Essentials

Unreal Engine is a powerful game development engine that provides rich functionalities to create 2D and 3D games. Developers have the opportunity to build cross-platform mobile and desktop games from scratch. This book will show you how to apply artificial intelligence (AI) techniques to your Unreal project using blueprints as your scripting language. You will start with an introduction to AI, and learn how to apply it to gaming. Then you’ll jump right in and create a simple AI bot and apply basic behaviors to allow it to move randomly. As you progress, you’ll find out how to implement randomness and probability traits. Using NavMesh, you will impart navigation components such as character movement, MoveTo nodes, settings, and world objects, and implement Behavior Trees. At the end of the book, you will troubleshoot any issues that might crop up while building the game.

Who this book is written for
This book is for programmers and artists who want to expand their knowledge of game AI in relation to Unreal Engine 4. You are recommended to have some experience of exploring Unreal Engine 4 prior to this book because we jump straight into game AI.

What you will learn from this book
- Understand the fundamental components of game AI within Unreal Engine 4
- Skillfully introduce game AI within Unreal Engine 4
- Configure, customize, and assign navigation and AI components to your pawn
- Create, debug, and analyze game AI behavior
- Design responsive game AI using the Behavior Tree methodology
- Create smart objects designed to interact with AI
- Utilize advanced AI features within your project to maximize the user experience

In this package, you will find:

- The author's biography
- A preview chapter from the book, Chapter 1 'Introduction to Game AI'
- A synopsis of the book’s content
- More information on Unreal Engine 4 AI Programming Essentials
Peter L. Newton gravitated toward computers at a young age. As his appetite for technology grew, web applications were his first exploration into development. The excitement of programming is what kept Peter diving further into different software designs and programming patterns. He is a self-taught programmer who has spent countless hours in reverse engineering assembly and arm instruction executables just for the joy of learning. Peter has several years of experience as a web developer, software developer, database architect, and hardware technician. His recent years were dedicated to the Virtual Reality/Gaming industry experience, working with such companies as Create, Sony Pictures, and the developers of Unreal Engine 4, Epic Games.

Peter's most recent VR project was Can You Walk The Walk?, which won Digital Hollywood’s "Best In Virtual Reality Based on a Cinematic or Television Experience" award.
**Jie Feng** is originally from Jiaxing, China. He is currently a PhD candidate at Columbia University, specializing in machine learning and computer vision. He has conducted research on problems ranging from detecting and recognizing objects in images and retrieving similar images from large-scale databases to understanding human behavior in videos. Jie's work has been published at top international conferences, and he has been granted a U.S. patent. He is also a software designer and developer and has worked at Microsoft, Amazon, and Adobe. Jie is passionate about applying Artificial Intelligence to real-world problems. His project using Microsoft Kinect to analyze motion for fitness has won People's Choice Award at Innovative Health Tech NYC competition, 2013. Jie is currently working on a fashion discovery product named EyeStyle.

Video games are the very thing that motivated him to study computer science. His favorite genre is action adventure. Titles including Resident Evil, Tomb Raider, and Uncharted inspire him in innovative thinking. This book is a unique experience for Jie to put his knowledge on Artificial Intelligence to game design and examine the potential of creating intelligent characters using Unreal Engine 4.
Artificial Intelligence (AI) is an essential part of any game. It makes the virtual world we create more immersive and fun to play in. Game AI is different from the general scientific AI that we know; it is more targeted to solve key problems in game design, including navigation, which is how a nonplayer character (NPC) should move from one point to another and avoid obstacles; decision making, which is how to perform certain actions based on different situations; and environment sensing, which is the ability to understand what exists in the environment and what its status is. These techniques make it possible to create a dynamic and realistic gameplay so that the players will be more engaged in the world that is created for them.

Game AI is complicated and brings a lot of challenges if you want to develop on your own. Unreal Engine 4 is a powerful game engine that provides rich functionalities to create cross-platform 3D and 2D games. It is well known for its advanced graphics and highly customizable components. Now, it is free to use and open source, which makes it one of the most popular game engines out there. Unreal Engine 4 comes with a complete suite of tools for game AI, including NavMesh, Behavior Trees, and Environment Query System. With these tools in hand, it is much easier to bring AI to your games. For game designers, you can even use a visual scripting tool called Blueprints to build your game logic, including AI, by just connecting nodes and without even writing a single line of code.

This book is our effort to introduce these wonderful tools in Unreal Engine 4 to build game AI to game creators who are interested in making their virtual world more interesting. It will cover all the components we have mentioned and show you how to use each tool to build different character behaviors and combine them to create more complex scenes.

We can't wait to see what you will create!
What this book covers

Chapter 1, Introduction to Game AI?, introduces the basic idea of AI and how it directly affects and enhances the gaming experience. You will learn the differences between the traditional and game-specific goals of AI.

Chapter 2, Creating Basic AI, helps you create your first AI step by step and talks about the techniques we will demonstrate along the way. We will dive right into Unreal Engine 4, using the bare components needed to create a single state with random movement for your AI.

Chapter 3, Adding Randomness and Probability, teaches you how to create random and probability techniques that can be used to add randomness, chance, and character to AI, which will make the game unpredictable and more interesting. We will cover how these are used within Unreal Engine 4.

Chapter 4, Introducing Movement, explains how to introduce movement to our AI characters within Unreal Engine 4. Path Finding will be used to allow our character to intelligently navigate within a level.

Chapter 5, Giving AI Choices, explains how to introduce autonomous behavior to our characters using Behavior Trees. Behavior Trees are a methodology that allows you to construct your AI logic visually in a tree structure and can be reused in different characters.

Chapter 6, How Does Our AI Sense?, explains how to use the different components available within Unreal Engine 4 to enable our AI to sense other AI and the pawns we will place within the world.

Chapter 7, More Advanced Movement, focuses on flocking and more advanced path-following behaviors. Flocking allows us to create group behaviors for several AI characters.

Chapter 8, Creating Patrol, Chase, and Attack AI, combines some of the components we used in the previous chapters, including AI Sense and Movement, to have our AI character navigate. Then, we will apply randomness to the time that the AI character will spend chasing after the characters it detects.

Chapter 9, What Have We Learned?, briefly glances over the previous chapters. We will also talk about additional examples of what we can achieve with these combined lessons.
Introduction to Game AI

This chapter will introduce the basic idea of Artificial Intelligence (AI) and how it directly affects and enhances the gaming experience. You will learn the differences between the traditional and also the game-specific goals of AI. We will introduce various techniques used in game AI, including navigation, Behavior Tree, sensor systems, and so on. You will learn in brief which tools we utilize for AI within Unreal Engine 4's editor. After this chapter, readers will gain a basic understanding of how AI can be applied to game development for a better gaming experience. The AI techniques that we will briefly cover here will be taught in the subsequent chapters.

Game Artificial Intelligence

When you first think of Artificial Intelligence, robots immediately come to mind. AI is derived from the idea of intelligence that helps living creatures make decisions. We take inputs, context, and our personal reasoning to decide on the actions we will perform. In AI, we try to virtually replicate this process to create systems that can have autonomous behavior. Assuming you have a fairly extensive gaming history, you would know that game AI is generally not smarter than some older games where your enemy may get stuck in a corner and fail to get out. Game AI now is by no means comparable to the general AI in scientific research. Game AI is designed to work in a well-controlled, predictable virtual world. It mainly consists of hardcoded rules to allow game actors to make proper actions corresponding to different situations. Game AI is meant to be fun, so it only needs to seem smart to the player within this context.

It is fair to say that AI is a very broad topic, so implementing every possible technique isn't the plan. So, it goes without saying that we will only cover what is necessary for you to create an awesome game AI. Keep in mind, though, that we will only touch on very specific game AI techniques; the world of AI is as vast as it is great.
How AI affects the gaming experience

Players seek a realistic and immersive experience in games. AI plays a huge part in forming this gaming experience by bringing realism and fun to the virtual world. Imagine that you are accompanied by a dog as you walk around or a flock of birds scatters when you make some noise. An enemy opponent is perhaps the most common and important implementation of game AI. The few types of game AI—navigating, fighting, assisting, or analytical—add the missing elements to other players to make them feel real and challenging to compete. This dates back to when it was used most notably in Chess, Nim, Pong, and Pac-Man. Up until now, it has been used in a war frame, with procedurally built levels. As the modern game design moves quickly by introducing new features to game play, such as the open world, massive in-game characters, and social interaction, it also introduces problems because these features cause AI decision making to require more input in unpredictable environments. Even now, AAA titles have their own complications with AI that result in poor user satisfaction. We will introduce in the following sections some powerful techniques to help create this important AI module and discuss how they are implemented in Unreal Engine.

Techniques and practices of game AI

There exist many techniques to cover different aspects in game AI, from fundamental movement to advanced environment sensing and decision making. Let’s look at them one by one.

Navigation

Navigation for AI is usually built up of the following tools:

- **Navigation Mesh**: Using tools such as Navigation Mesh, also known as NavMesh, you can designate areas in which AI can traverse. NavMesh is a simplified polygonal representation of a level (the green region in the following screenshot), where each polygon acts as a single node connected to its nearby ones. Usually, this process is automated and doesn’t require designers to place nodes manually. Using special tools in Unreal, they analyze the geometry of the level and generate the most optimized Navigation Mesh accordingly. The purpose, of course, is to determine the playable areas in the level by the game agents. Note that this is the only path-finding technique available; we will use NavMesh in the examples provided in this book because it works well in this demonstration.
• **Path Following (Path nodes):** A similar solution to NavMesh, Path nodes can designate the space in which the AI traverses:

![Path nodes diagram]

• **Behavior Tree:** Using Behavior Tree to influence your AI's next destination can create a more varied player experience. It not only calculates its requested destination, but also decides whether it should enter the screen with a cart wheeling double-back flip, no hands, or the triple somersault and jazz hands.

• **Steering behaviors:** Steering behaviors affect the way AI moves while navigating to avoid obstacles. This also means using Steering to create formations with your fleets that you have set to attack the king's wall. Steering can be used in many ways to influence the movement of the character.

• **Sensory systems:** Sensory systems can provide critical details, such as the nearby players, sound levels, nearby cover, and many other variables of the environment that can alter movement. It's critical that your AI understands the changing environment so that it doesn't break the illusion of being a real opponent.
While all these components aren't necessary to achieve AI navigation, they all provide critical feedback, which can affect the navigation. Navigating within a world is limited only by pathways within the game. We can see an example of group behavior with several members following a leader here:

---

**Achieving realistic movement with Steering**

When you think of what Steering does for a car, you would be right to imagine the same idea applied to game AI navigation. Steering influences the movement of AI as it goes to its next destination. The influences can be supplied as necessary, but we will go over the most commonly used. Avoidance is used to essentially avoid colliding with oncoming AI. Flocking is another key factor in steering and is useful in simulating interesting group movement, such as a complete panic situation, or a school of fish. The goal of Steering behaviors is to achieve realistic movement and behavior within the player's world.
Creating a character with randomness and probability

AI with character is what randomness and probability add to the bot's decision making abilities. If a bot attacked you in the same way, always entered the scene in the same way, and annoyed you with its laugh after every successful hit, it wouldn't make for a unique experience. Using randomness and probability, you can instead make the AI laugh based on probability or introduce randomness to the AI's skill of choice. Another great by-product of applying randomness and probability is that it allows you to introduce levels of difficulty or lower the chance of missing the skill cast, and even allows bots to aim more precisely. If you have bots who wander around looking for enemies, probability, and randomness could be used to work with the bot's sensory input to make a more rational decision.

Creating complex decision making with Behavior Tree

Finite State Machines (FSM) is a model to define how a finite number of states transit among each other. For example, this allows it to go from gathering to searching and then attacking, as shown in the following image. Behavior trees are similar, but they allow more flexibility. A behavior tree allows hierarchical FSM, which introduces another layer of decisions. So, the bot decides among branches of behaviors that define the state it is in. There is a tool provided by UE4 called Behavior Tree. This editor tool allows us to modify AI behavior quickly and with ease.

Here's a diagram of the FSM model:

![Diagram of FSM model](image)
Let's take a look at the components of Behavior Tree:

![Behavior Tree Diagram]

Now, we will discuss the components found within UE4 Behavior Tree.

**Root**

This node is the beginning node that sends the signal to the next node in the tree. This connects to a composite, which begins your first tree. What you may notice is that you are required to use a composite first to define a tree and then to create a task for this tree. This is because hierarchical FSM creates branches of states. These states will be populated with other states or tasks. This allows an easy transition among multiple states. You can see what a root node looks like as shown in the following screenshot:
Decorators

Decorators are conditional statements (the blue part on top of a node) that control whether or not a branch in the tree or even a single node can be executed. I used a decorator in the AI we will make to tell it to update to the next available route.

In the following image, you can note the **Attack & Destroy** decorator that defines the state on top of the composite. This state includes two tasks, **Attack Enemy** and **Move To Enemy**, which also has a decorator telling it to execute only when the bot state is **Search**:

In the preceding screenshot, you can note the **Attack & Destroy** decorator that defines the state on top of the composite. This state includes two tasks, **Attack Enemy** and **Move To Enemy**, which also has a decorator telling it to execute only when the bot state is **Search**.

Composites

These are the beginning points of the states. They define how the state will behave with returns and execution flow. They have three main types: Selector, Sequence, and Simple Parallel. This beginning branch has a conditional statement, if the state is equal or greater than Search state:
Selector executes each of its children from left to right and doesn't fail; however, it returns success when one of its children returns success. So, this is good for a state that doesn't check for successfully executed nodes. The following screenshot shows an example of Selector:

![Selector Example](image)

Sequence executes its children in a similar fashion to Selector but returns fail when one of its children returns fail. This means that it's required that all nodes return success to complete the sequence. You can see a Sequence node in the following screenshot:

![Sequence Example](image)

Last but not least, Simple Parallel allows you to execute a task and a tree essentially at the same time. This is great for creating a state that requires another task to always be called. To set it up, you need to first connect it to a task that it will execute. The second task or state connected continues to be called with the first task until the first task returns success.
Services

Services run as long as the composite it is added to stays activated. They tick at the intervals you set within the properties. They have another float property called Tick Interval that allows you to control how often this service is executed in the background. Services are used to modify the state of AI in most cases because it’s always called. For example, in the bot that we will create, we will add a service to the first branch of the tree so that it’s called without interruption and will be able to maintain the state that the bot should be in at any given movement. The green node in the following screenshot is a service with important information explicitly:

![Service Node Example](image)

This service, called Detect Enemy, actually runs a deviating cycle that updates Blackboard variables such as State and Enemy Actor.

Tasks

Tasks do the dirty work and report success or failed if it's necessary. They have blueprint nodes that can be referred to in Behavior Tree. There are two types of nodes that you’ll use most often when working with Task: Event Receive Execute, which receives the signal to execute the connected scripts, and Finish Execute, which sends the signal back and returns true or false on success. This is important when making a task meant for the Sequence composite node.
Blackboard

A Blackboard is an asset to store the variables to be used within the AI Behavior Tree. They are created outside Behavior Tree. In our example, we will store an enumeration variable for the state in the State, EnemyActor object to hold the currently targeted enemy, and Route to store the current route position that the AI is requested to travel to, just to name a few. You can see all current variables as keys in Blackboard panel as follows:

![Blackboard panel](image)

They work just by setting a public variable of a node to one of the available Blackboard variables in the drop-down menu. The naming convention in the following screenshot makes this process streamlined:

![Details panel](image)
Sensory systems

A sensory system usually consists of several modules, such as sight, sound, and memory, to help the AI capture information about the environment. A bot can maintain the illusion of intelligence using sounds within their environment to make a deliberate risk assessment before engaging a hazardous threat or aiding a nearby teammate who is calling for help. The use of memory will allow the bot to avoid an area where it remembers seeing a severe threat or rush back to an area where it last saw its group. Creating a sensory system in the case of an enemy player is heavily based on the environment where the AI fights the player. It needs to be able to find cover, evade the enemy, get ammo, and other features that you feel create immersive AI for your game. A game with AI that challenges the player creates a unique individual experience. A good sensory system contributes critical information that makes for reactive AI. In this project, we will use the sensory system to detect the pawns that the AI can see. We will also use functions to check for the line of sight of the enemy. We will check whether there is another pawn in the way of our path. We can check for cover and other resources within the area.

Machine learning

Machine learning is a branch on its own. This technique allows AI to learn from situations and simulations. Inputs are taken from the environment, including the context in which the bot allows it to make decisive actions. In machine learning, the inputs are put within a classifier that can predict a set of outputs with a certain level of certainty. Classifiers can be combined into ensembles to increase the accuracy of probabilistic prediction. We won't dig deep into this subject, but there exist a vast amount of resources for studying machine learning, ranging from text books (*Pattern Recognition and Machine Learning* by Christopher M. Bishop, Springer) to online courses (*Machine Learning* on coursera.org).

Tracing

Tracing allows another actor within the world to detect objects by ray tracing. A single line trace is sent out, and if it collides with an actor, the actor is returned along with information on the impact. Tracing is used for many reasons; one way it is used in FPS is to detect hits. Are you familiar with the hit box? When your player shoots in a game, a trace is shot out that collides with the opponent's hit box, determining the damage to the player, and if you're skillful enough, it results in death. Other shapes available for traces, such as spheres, capsules, and boxes, allow tracing for different situations. Recently, I used Box Trace for my car to detect objects near it.
**Influence Mapping**

Influence Mapping isn't a finite approach; it's the idea that specific locations on the map would be attributed information that directly influences the player or AI. An example of using Influence Mapping with AI is presence falloff. Let's say we have other enemy AI in a group; their presence map would create a radial circle around the group with the intensity based on the size of the group. This way, the other AI knows by entering this area that they're entering a zone occupied by other enemy AI.

Practical information isn't the only thing people use it for, so just understand that it's meant to provide another level of input to help your bot make more additional decisions. As shown in the following image, different colors represent zones occupied by different types of AI, and color intensity indicates the influence with respect to each AI character:

![Influence Mapping Image](image-url)

Practical information isn't the only thing people use it for, so just understand that it's meant to provide another level of input to help your bot make more additional decisions.
Unreal Engine 4 tools

Unreal Engine 4 provides a complete suite of tools to add common AI capability to your game. We will go into the details of each tool within this book. Here is a list of the tools that are covered:

- **Behavior Tree**: This is used to create different states and the logic behind AI.
- **Navigation Component**: This handles movement for AI.
- **Blackboard Asset**: These are used to store information. They act as the local variable for AI.
- **Enumeration**: This is used to create states, which you can alternate between.
- **Target Point**: Our Waypoints class is derived from the Target Point class, which we will use to create a basic form of Path node.
- **AI Controller and Character**: This controller will handle communication between the world and controlled pawn for AI.
- **Navigation Volumes**: This is used to create Navigation Mesh in the environment to enable Path Finding for AI.

Let's look at the following screenshot:

There are two types of NavMesh volume. The first, the NavMesh Bounds volume, defines the area for NavMesh. The Nav Modifier volume, when supplied with a Nav Area class, affects the NavMesh Bounds volume's navigation attributes where the two intersect.
Summary

In this chapter, we started by introducing game AI and discussing why it is important for our gaming experience. Then, we illustrated most of the used game AI techniques and what they are capable of. The corresponding UE4 tools for game AI were also mentioned to provide a bigger picture of the content we will cover throughout this book. In the next chapter, we will create our basic AI by setting up an AI-controlled player and adding some simple behavior to it.
Where to buy this book

You can buy Unreal Engine 4 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.