Building an RPG with Unreal

Now that Unreal Engine 4 has become one of the most cutting-edge game engines in the world, developers are looking for the best ways of creating games of any genre with the engine. Building an RPG with Unreal will lay the foundations of creating a turn-based RPG with Unreal Engine 4.

The book starts by walking you through creating a turn-based battle system that can hold commands for party members and enemies. You’ll get your hands dirty by creating NPCs such as shop owners, and important mechanics that are a feature of every RPG, such as a currency system, an inventory, dialog, and character statistics. Although this book specifically focuses on the creation of a turn-based RPG, there are a variety of topics that can be utilized when creating games of other genres.

By the end of the book, you will be able to build upon core RPG framework elements to create your own game experience.

Who this book is written for

If you are new to Unreal Engine and have always wanted to script an RPG, you are this book’s target reader. The lessons assume you understand the conventions of RPGs and have some awareness of the basics of using the Unreal editor to build levels.

What you will learn from this book

- Program gameplay elements in C++ in Unreal
- Create custom game data for entities such as players and enemies
- Create a turn-based combat engine
- Design menu systems and Blueprint logic
- Create an NPC and dialog system
- Integrate equipment and items
- Develop the foundations of a saving and loading system
- Create reusable code chunks and elements that can easily be integrated into other games

In this package, you will find:

- The author's biography
- A preview chapter from the book, Chapter 1 'Getting Started with RPG Design in Unreal'
- A synopsis of the book’s content
- More information on Building an RPG with Unreal
About the Authors

**Steve Santello** is a well-seasoned educator and veteran of the game industry from Chicago, Illinois, USA. His cross-disciplinary study in art, design, programming, and project management has allowed him to explore every facet of game development. As an educator, he has used his passion to teach others about game development, and has populated the game industry with many talented developers over the years. He has worked on over 10 different titles as a 2D and 3D artist for the Chicago game developer Babaroga. With Babaroga, he worked as one of the four artists on many mobile titles published by EA such as *Spore Origins*, *The Godfather Game*, and *Pictionary: The Game of Quick Draw*. He also worked on many mobile titles published by Disney Interactive, such as *Hannah Montana In Action* and *Meet the Robinsons*. He and his development team also developed a number of original IPs such as *Babaroga Eats Children* and *BEES!*

Since 2006, Steve has also been a university professor. He helped pioneer the *Digital Entertainment and Game Design* program at ITT Technical Institute in St. Rose, LA, and also the *Game and Simulation Programming* program at DeVry University in Addison, IL. He has also served as an adjunct professor at the Illinois Institute of Art, Chicago, where he taught game prototyping as a team manager and acting producer. On the side, he developed game and simulation prototypes and plans to release his first two independent games, which were developed with Unreal Engine 4 sometime in 2015. He has writing contributions in *Game Development Essentials: Game Interface Design 2nd Edition*, Kevin Saunders and Jeannie Novak, Delmar Cengage Learning. In that book, he wrote about the past, present, and future of the user interface in games, which included breaking down the HUD in games such as *Deadspace*, and talked about how he and his team designed the user interface in *Spore Origins*.

Steve is currently working towards tenure as a CIS gaming instructor at the College of DuPage in Glen Ellyn, Illinois, USA. Although he is very proud of his success, he knows that all his combined experiences have played a major role in where he resides today.
Alan R. Stagner is an independent developer. He was introduced to programming by his father; he sought out different ways to create games in a variety of languages. Most recently, he found the Unity game engine and was instantly hooked, and discovered his love of multiplayer game development. He has also dabbled in database and server programming from time to time, mostly involving PHP and MySQL, with recent forays into ASP.NET.
Now that Unreal Engine 4 has become one of the most cutting-edge game engines in the world, developers both AAA and Indie alike are looking for the best ways of creating games of any genre using the engine. Upon Unreal's first release, it was known as a great first-person shooter game engine, but with the success of games such as WB's Mortal Kombat, Chair Entertainment's Shadow Complex, and Epic Games' Gears of War, along with highly anticipated upcoming games such as Capcom's Street Fighter 5, Comcept's Mighty No. 9, and Square Enix's Final Fantasy VII Remake, Unreal has proven itself to be one of the greatest engines to use when creating virtually any genre of game. This book will lay the foundations of creating a turn-based RPG in Unreal Engine 4.

**What this book covers**

*Chapter 1, Getting Started with RPG Design in Unreal*, alerts the reader to the various preparation steps required to make an RPG before jumping into Unreal. In order to avoid potential obstacles to progress, the example content is provided and briefly introduced.

*Chapter 2, Scripting and Data in Unreal*, walks the reader through using C++ to program gameplay elements in Unreal, creating Blueprint graphs, and working with custom game data in Unreal.

*Chapter 3, Exploration and Combat*, walks the reader through creating a character that runs around the game world, defining character data and party members, defining enemy encounters, and creating a basic combat engine.

*Chapter 4, Pause Menu Framework*, covers how to create a pause menu with inventory and equipment submenus.

*Chapter 5, Bridging Character Statistics*, covers how to keep track of the player's stats within the menu system.
Chapter 6, NPCs and Dialog, covers adding interactive NPCs and dialogue to the game world. The reader will learn how to use Blueprints to define what happens when an object or NPC is interacted with, including using a set of custom Blueprint nodes to create dialogue trees.

Chapter 7, Gold, Items, and a Shop, covers adding interactive NPCs and objects to the game world. The reader will learn how to use Blueprint to define what happens when an object or NPC is interacted with, including using a set of custom Blueprint nodes to create dialogue trees. The user will also be creating items that can be bought in a shop using the gold dropped by enemies.

Chapter 8, Inventory Population and Item Use, covers populating an inventory screen with items and using the items when not in combat.

Chapter 9, Equipment, covers the creation of equipment and equipping weapons and armor from an equipment screen.

Chapter 10, Leveling, Abilities, and Saving Progress, covers adding abilities to the game, keeping track of experience for each party member, awarding experience to party members after combat, defining leveling and stat updates for a character class, and saving and loading player progress.
Getting Started with RPG Design in Unreal

Role-playing games are highly complex things. Even within the RPG genre, there is a diverse range of games with wildly different mechanics and controls.

Before even a single line of code is written, it's important to figure out what kind of RPG you want to make, how the game is played, whether the game should be turn-based or real-time, and what stats the player will have to concern themselves with.

In this chapter, we will cover the following topics which show how to design an RPG before you begin making it:

- Tools for game design
- The design and concept phase
- Describing the game's features and mechanics
- Tropes in existing RPGs
- RPG design overview

Tools for game design

While you can always type everything in Notepad and keep track of design decisions that way, there are a variety of tools available that can help when working on the design document.

Of particular note is the Google suite of tools. These tools come free with a Google account and have many applications, but in this case, we'll take a look at applying them to game design.
Google Drive
Google Drive is a cloud-based file storage system like Dropbox. It comes free with a Google account and has up to 15 GB of space. Google Drive makes sharing files with others very easy, as long as they also have a Google account. You can also set up permissions, such as who is allowed to modify data (maybe you only want someone to read but not change your design document).

Google Docs
Integrated with Google Drive is Google Docs, which is a fully featured online word processing application. It includes many features such as live collaborative editing, comments, and a built-in chat sidebar.

The bulk of your design document can be written in Google Docs and shared with any potential collaborators easily.

Google Spreadsheets
Just as with Google Docs, Google Spreadsheets is also directly integrated with Google Drive. Google Spreadsheets provides an Excel-style interface that can be used to keep track of data in a handy row/column format. You can also enter equations and formulas into cells and calculate their values.

Spreadsheets might be used, for example, to keep track of a game's combat formulas and test them with a range of input values.

Additionally, you can use spreadsheets to keep track of lists of things. For example, you may have a spreadsheet for weapons in your game, including columns for name, type, damage, element, and so on.

Pencil and paper
Sometimes, nothing beats the trusty method of actually writing things down. If you have a quick idea popped up in your head, it's probably worth quickly jotting it down. Otherwise, you'll most likely forget what the idea was later (even if you think you won't—trust me, you probably will). It doesn't really matter whether you think the idea is worth writing down or not—you can always give it more thought later.
The design and concept phase

Just as how a writer works from an outline or mind map, or an artist works from a rough sketch, nearly all games start from some sort of a rough concept or design document.

A design document's purpose is to describe nearly everything about a game. In the case of an RPG, it would describe how the player moves around the game world, how the player interacts with enemies and NPCs, how combat works, and more. The design document becomes the basis upon which all the game code is built.

Concept

Usually, a game starts with a very rough concept.

For example, let’s consider the RPG we’ll be making in this book. I might have the idea that this game would be a linear turn-based RPG adventure. It's a very rough concept, but that's OK—while it may not be a terribly original concept, it's enough to begin fleshing out and creating a design document from.

Design

The design document for the game is based on the previous rough concept. Its purpose is to elaborate on the rough concept and describe how it works. For example, while the rough concept was linear turn-based RPG adventure, the design document's job is to take that further and describe how the player moves around the world, how the turn-based combat works, combat stats, game over conditions, how the player advances the plot, and a lot more.

You should be able to give your design document to any person and the document should give them a good idea of what your game will be like and how it works. This, in fact, is one of the big strengths of a design document—it's incredibly useful, for example, as a way of ensuring that everyone on a team is on the same page so to speak.

Describing the game's features and mechanics

So, assuming you have a very rough concept for the game and are now at the design phase, how do you actually describe how the game works?
There are really no rules for how to do this, but you can divide your theoretical game into the important core bits and think about how each one will work, what the rules are, and so on. The more information and the more specific, the better it is. If something is vague, you'll want to expand on it.

For instance, let's take combat in our hypothetical turn-based RPG.

*Combatants take turns selecting actions until one team of combatants is dead.*

What order do combatants fight in? How many teams are there?

*Combatants are divided into two teams: the player team and the enemy team. Combatants are ordered by all players and followed by all enemies. They take turns in order to select actions until one team of combatants is dead (either the enemy team or the player team).*

What sort of actions can combatants select?

*Combatants are divided into two teams: the player team and the enemy team. Combatants are ordered by all players and followed by all enemies. Combatants take turns in order to select actions (either attacking a target, casting an ability, or consuming an item) until one team of combatants is dead (either the enemy team or the player team).*

And so on.

**Tropes in existing role-playing games**

Even though RPGs can vary wildly, there are still plenty of common themes they frequently share—features that a player expects out of your game.

**Stats and progression**

This one goes without saying. Every RPG—and I do mean *every* RPG—has these basic concepts.

Statistics, or *stats*, are the numbers that govern all combat in the game. While the actual stats can vary, it's common to have stats such as max health, max MP, strength, defense, and more.

As players progress through the game, these stats also improve. Their character becomes better in a variety of ways, reaching maximum potential at (or near) the end of the game. The exact way in which this is handled can vary, but most games implement experience points or XP that are earned in combat; when enough XP has been gained, a character's *level* increases, and with it, their stats increase as well.
Classes
It's common to have classes in an RPG. A class can mean a lot, but generally it governs what a character's capabilities are and how that character will progress.

For instance, a Soldier class might define that, as an example, a character is able to wield swords, and mainly focuses on increased attack power and defense power as they level up.

Special abilities
Very few role-playing games can get away with not having magic spells or special abilities of some sort.

Generally, characters will have some kind of magic meter that is consumed every time they use one of their special abilities. Additionally, these abilities cannot be cast if the character does not have enough magic (the term for this varies—it might also be called mana, stamina, or power—really, anything to fit the game scenario).

RPG design overview
With all that aside, we're going to take a look at the design for the RPG we will be developing over the course of this book, which we'll call Unreal RPG.

Setting
The game is set in a tower. Players advance to the top of the tower, fighting enemies along the way until they reach the top and encounter the villain. The goal of the game is to defeat the villain in combat.

Exploration
While not in combat, players explore the world in an isometric view, similar to games such as Diablo. In this view, players can interact with NPCs and props in the world, and also pause the game to manage their party members, inventory, and equipment.
Dialogue
When interacting with NPCs and props, dialogue may be triggered. Dialogue in the game is primarily text-based. Dialogue boxes may be either linear, the player simply presses a button to advance to the next dialogue page, or multiple-choice. In the case of multiple-choice, the player is presented with a list of options. Each option will then proceed to a different page of dialogue. For instance, an NPC might ask the player a question and allow the player to respond "Yes" or "No", with different responses to each.

Shopping
A shop UI can also be triggered from a dialogue. For example, a shopkeeper might ask the player whether they want to buy items. If the player chooses "Yes", a shop UI is displayed.

While in a shop, players can buy items from the NPC.

Gold
Gold can be attained by defeating monsters in battle. This gold is known as a type of enemy drop.

The pause screen
While the game is paused, players can do the following:

- View a list of party members and their statuses (health, magic, level, effects, and so on)
- View abilities that each party member has learned
- View the amount of gold currently carried
- Browse an inventory and use items (such as potions, ethers, and so on) on their party members
- Manage items equipped to each party member (such as weapons, armor, and so on)

Party members
The player has a list of party members. These are all the characters currently on the player's team. For instance, the player may meet a character in the tower who joins their party to aid in combat.
Equipment
Each character in the player's party has the following equipment slots:

- **Armor**: A character's armor generally increases defense
- **Weapon**: A character's weapon generally provides a boost to their attack power (as given in the attack formula in the Combat section of this chapter)

Classes
Player characters have different classes. A character's class defines the following elements:

- The experience curve for leveling up
- How their stats increase as they level up
- Which abilities they learn as they level up

The game will feature one player character and class. However, based on this player character, we can easily implement more characters and classes, such as a healer or black mage, into the game.

Soldier
The Soldier class focuses on increasing attack, max HP, and luck. Additionally, special abilities revolve around dealing with lots of damage to enemies.

Therefore, as the Soldier class levels up, they deal more damage to enemies, withstand more hits, and also deliver more critical blows.

Combat
While exploring the game world, random encounters may be triggered. Additionally, combat encounters can also be triggered from cut scenes and story events.

When an encounter is triggered, the view transitions away from the game world (the field) to an area specifically for combat (the battle area), an arena of sorts.

Combatants are divided into two teams: the enemy team and the player team (consisting of the player's party members).

Each team is lined up, facing each other from the opposite ends of the battle area.
Combatants take turns, with the player team going first, followed by the enemy team. A single round of combat is divided into two phases: decision and action.

Firstly, all combatants choose their action. They can either attack an enemy target or cast an ability.

After all combatants have decided, each combatant executes their action in turn. Most actions have a specific target. If, by the time the combatant executes their action, this target is not available, the combatant will pick the next available target if possible, or else the action will simply fail and the combatant will do nothing.

This cycle continues until either all enemies or players are dead. If all enemies are dead, the player's party members are awarded with XP, and loot may also be gained from the defeated enemies (usually, a random amount of gold).

However, if all players have died, then it is game over.

**Combat stats**

Every combatant has the following stats:

- **Health points**: A character's **health points** (HP) represents how much damage the character can take. When HP reaches zero, the character dies. HP can be replenished via items or spells, as long as the character is still alive. However, once a character is dead, HP cannot be replenished—the character must first be revived via a special item or spell.

- **Max health**: This is the maximum amount of HP a character can have at any given time. Healing items and spells only work up to this limit, never beyond. Max health may increase as the character levels up, and can also be temporarily increased by equipping certain items.

- **Magic points**: A character's **magic points** (MP) represents how much magic power they have. Abilities consume some amount of MP, and if the player does not have enough MP for the ability, then that ability cannot be performed. MP can be replenished via items.

  It should be noted that enemies have effectively infinite MP, as their abilities do not cost them any MP.

- **Max magic**: This is the maximum amount of MP a character can have at any given time. Replenishing items only work up to this limit, never beyond. Max magic may increase as the character levels up, and can also be temporarily increased by equipping certain items.
• **Attack power**: A character's attack power represents how much damage they can do when they attack an enemy. Weapons have a separate attack power that is added to regular attacks. The exact formula used to deal with damage is as follows:

$$\text{max} (\text{player}.\text{ATK} - \text{enemy}.\text{DEF}, 0) + \text{player}.\text{weapon}.\text{ATK}$$

So firstly, enemy defense is subtracted from the player's attack power. If this value is less than zero, it is changed to zero. Then, the weapon's attack power is added to the result.

• **Defense**: A character's defense reduces the damage they take from an enemy attack.

The exact formula is as given just previously (defense is subtracted from the enemy's base attack value and then the enemy's weapon attack power is added).

• **Luck**: A character's luck affects that character's chance of landing a critical hit, which will double the damage dealt to an enemy.

Luck represents the percent chance of dealing with a critical hit. Luck ranges from 0 to 100, representing the range from 0% to 25%, so the formula is as follows:

$$\text{isCriticalHit} = \text{random}(0, 100) \leq (\text{player}.\text{Luck} \times 0.25)$$

So, if the player's luck is 10, given that the random number falls at the number 10 within its range of 0 to 100, then the chance of dealing a critical hit is 2.5%.

The critical hit multiplier is applied after the damage is calculated, as follows:

$$2 \times (\text{max}(\text{player}.\text{ATK} - \text{enemy}.\text{DEF}, 0) + \text{player}.\text{weapon}.\text{ATK})$$

**Combat actions**

Actions during combat are divided into three categories: attack and ability.

**Attack**

Every character has an attack ability that costs zero MP and, for player characters, is shown as the first option in the action menu during the decision phase of a round.

Generally, an attack takes a single enemy target and deals damage to that enemy. The damage formula is as given previously for the *attack power* stat.
Ability
Every character, as mentioned earlier, has a set of abilities they know. Excluding attack, abilities cost some amount of MP and have a variety of effects. Abilities can have different types of targets, as follows:

• A single enemy
• All enemies
• A single ally
• All allies

Abilities can heal targets, revive dead targets, remove some effects, summon temporary allies, temporarily increase a character’s stats, and more. However, abilities never restore MP.

Abilities have a set MP cost. This is the amount of MP the character must have in order to perform that ability, and the amount of MP that will be consumed upon casting the ability.

After combat/victory
Once all enemy combatants have died, the player wins the fight. Upon winning the fight, the player is rewarded with random loot, and experience points are divided between party members.

Loot
Every enemy defines the loot that is received upon defeating the enemy. This includes how much gold is received to defeat this enemy.

Experience
Each enemy defines how much experience it is worth. After combat, the experience of every defeated enemy is summed up. Then, this value is evenly divided between all currently living players (any party member who has died does not receive any EXP) and rounded up to the nearest integer (for example, if the total experience is 100 and there are three party members, then $100/3 = 33.3333$, which is rounded up to 34).
Experience and leveling
As party members earn experience, they will level up.

The amount of experience required to go from one level to the next is given by the following formula:

\[ f(x) = (xa) + c \]

Here, \( x \) is the current level, \( a \) is a positive value greater than one (affecting how steeply the curve increases), and \( c \) is the base offset, which is the amount of experience required to go from level 1 to level 2. This defines a simple exponential value increase. The \( a \) and \( c \) values are defined by the character's class.

To get the total amount of experience required to level up from the current level, the preceding formula is calculated and summed for each level up to the current level. For instance, if we want to know how much total EXP is required to get to level 31 (from level 30), we calculate it in the following way:

\[ f(1) + f(2) + f(3) + \ldots + f(30) \]

When a player levels up, their stats increase and they may also learn a new ability. Stat increases and learned abilities are defined on the character class.

The maximum level of any character in the game is 50.

Stat increases
For a given character class, for every character stat, the class defines a starting value at level 1 and an ending value at 50. For example, using standard math library functions, the value of attack for any given level would be a simple linear interpolation between the starting value and the ending value, using the character's level (divided by max level) as the interpolation value (the result would then be rounded up to ensure it is a whole integer number).

So, for example, if a soldier's max HP starts at 100 at level 1 and ends at 1,000 at level 50, then at level 25 the soldier's max HP will be 550.

Learning abilities
Each character class defines a table of abilities. Each entry in the table references which ability will be learned and at what level that ability is learned. When the character levels up, any abilities in the table that has the given level will be added to that character's known abilities.
Any abilities learned at level 1 are automatically added to a character's skill set.

Game over
If all players have died, either during combat or in the field, then the game ends.

Victory
If the player defeats the main villain in combat, they are given an ending cut scene before being returned to the main menu.

The game cannot be saved after a victory. Loading the game from the main menu will put players at their last save (prior to the final fight).

Choosing the right formula
In the preceding sections, the design of this sample game describes a small range of stats for characters in the game. It also outlines a variety of formulas for calculating damage, leveling up, and so on.

One thing to keep in mind is that these stats, values, and formulas are simply there to show you how to implement the core functionality of an RPG. These are not the be-all-end-all of stats or formulas. In fact, the design has intentionally used a limited set of stats and simple formulas to keep the scope simple.

With this in mind, when you're working on your own game, you will have to decide these things for yourself—what stats your characters use, how combat works, and what formulas the game will use to calculate the outcome of battle. So, how do you come up with all of these things on your own?

Unfortunately, the answer is "it depends". There's no silver bullet to balance your game and keep it fun. What stats you use depends on how your combat works (it makes no sense to have a "hit chance" stat if, for instance, your game takes place from a first-person perspective using guns).

Another thing to keep in mind is that the actual values and formulas you use don't matter. What does matter is that the end result is fun, fair, and balanced. It doesn't matter if it takes one, one hundred, or one million experience points to level up if the end result is still fun and feels fair.
Summary

In this chapter, we took a look at what tools are at your disposal to design the RPG of your dreams, how important it is to design your game before you begin developing, how to come up with a rough concept and design, and how to describe your game’s mechanics. We've also seen an overview of the game that we will be developing over the course of this book.

In the next chapter, we'll start to dive into Unreal and learn about scripting gameplay elements and working with game data in Unreal Engine.
Where to buy this book

You can buy Building an RPG with Unreal 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.