CRYENGINE Game Development Blueprints

Perfect the art of creating CRYENGINE games through exciting, hands-on game development projects

Richard Gerard Marcoux III  Chris Goodswen
Riham Toulan               Sam Howels
In this package, you will find:

- The authors biography
- A preview chapter from the book, Chapter 1 'Getting Started'
- A synopsis of the book’s content
- More information on CRYENGINE Game Development Blueprints
Richard Gerard Marcoux III is a very hardworking and intelligent software engineer with a passion for teaching and helping others. He has captured the attention of over 600,000 people through his YouTube channel with his efforts to educate beginners in the field of game development using the CRYENGINE technology. He also has an extensive IT background, working in the computer / software diagnosis / repair field for the past 7 years. Lately, he has been heavily involved in C++ games and middleware development, where he plans to create compelling 2D and 3D video games for all ages in genres ranging from platform games all the way to RPGs.

Chris Goodswen is a 3D character artist currently working at Crytek with 4 years of experience working with CRYENGINE. He has also worked on Ryse: Son of Rome for Xbox One as well as Warface and Hunt.

Chris is responsible for modeling and texturing characters as well as developing technical systems for characters; alongside this, he also works with universities, mentoring students and giving lectures on 3D character art and video game development.

Riham Toulan is a senior technical artist/animator working at Dice EA, who specializes in character rigging. She has more than 4 years of experience working with CRYENGINE. She worked on the highly cinematic Xbox One launch game Ryse: Son of Rome at Crytek, where she was responsible for developing rigging pipelines and tools in Maya, helping the R&D team develop new CRYENGINE technologies, and consulting the CRYENGINE licensees.
Sam Howels is a senior designer at the Nottingham-based Deep Silver Dambuster Studios. He is currently working on the upcoming sequel to the 2011 game Homefront, titled Homefront: The Revolution. He was recruited at the age of 18 after his dedicated contribution to the modding scene. He has a strong passion for technical problem solving as well as creating engaging and diverse gameplay experiences. Before joining Deep Silver, he worked on multiple AAA titles at both Crytek Frankfurt and Crytek UK, and he has over 8 years of experience in developing content with the CRYENGINE toolset.
Preface

This book is authored by Richard Marcoux, Chris Goodswen, Riham Toulan, and Samuel Howels; they will be your instructors and friends throughout this book. You are about to embark on a journey of discovery and find out what you can achieve in CRYENGINE, learn some tricks of the trade, the game programming techniques, the new aspects of the CRYENGINE code, and most importantly, how to create full working games. If there is one thing that we want you to take away after reading this book is that it's not the game we will be making, but instead the techniques and the problem solving that went into making it. The goal is to arm you with the knowledge and out-of-the-box thinking that is required to create a CRYENGINE game.

What this book covers

Chapter 1, Getting Started, shows you the CRYENGINE "Blank" Game Starter-Kit that was specifically designed to teach developers how to create a CRYENGINE game from scratch, and provide a blank slate for them to start with. You will install and compile this kit.

Chapter 2, Creating a Playable Character, shows you how to create a completely playable character from scratch and control its movement with the keyboard.

Chapter 3, Implementing Weapons and Ammo, shows you how to implement a weapon and ammo system, as the player will need a way to defeat bad guys.

Chapter 4, Creating an Enemy AI, shows you how to create an enemy AI and give it some basic intelligence.

Chapter 5, Creating User Interfaces, shows you how to create a complete start and end game menu by using Scaleform, Flash, and C++.
Chapter 6, The Modeling Workflow for Game Characters and Tools, gives an overview of the character art workflow principles, terminologies, and how to prepare for the tasks ahead.

Chapter 7, Highpoly Modeling, discusses why we need to create a highpoly model and its uses in the game in current and next-gen game development. In this chapter, we will be working with some of the principles of highpoly modeling and going through a basic workflow to create the highpoly model in Zbrush.

Chapter 8, Lowpoly Modeling, covers the lowpoly generation and some of the most important areas to remember, such as efficient topology, areas to remember for deformation, other important topics, such as an efficient UV mapping, and how to generate LODs from the original lowpoly.

Chapter 9, Texturing and Materials, explores the techniques required to create and bake textures. In this chapter, we will take a look at the tools, such as Photoshop and Zbrush, used for creating texture maps, how to bake the highpoly information to the lowpoly model by using xNormal, and also see how these baked maps can be used in the creation of textures.

Chapter 10, Building the Character Rig, shows you how to build an animator friendly rig for the character in Maya. We will also discuss the folder structure for the character files and explore the already made deformation skeleton on the character, and how to create a simple and efficient rig using that skeleton.

Chapter 11, Exporting the Character to CRYEngine, shows you how to export a character to CRYENGINE step-by-step and explains the animation pipeline and how to use Character Editor to debug and add extra secondary animations to your characters with CRYENGINE physics.

Chapter 12, Initial Level Blockout and Setup, covers the good working practices and tips used for quickly jumping into making a new level in the SDK, giving us a good base to start adding more complex scripted content later on.

Chapter 13, The Flow Graph Workflow, introduces you to the concept of Game Tokens to communicate with the Flow Graphs. We'll also cover how to set up the level logic to modularly accommodate the various scripted elements that go into making a single player level in CRYENGINE.

Chapter 14, Scripting Gameplay Content, dives deep into creating all the elements that make up a Crysis style action bubble, as we now have a solid grounding in how to efficiently and cleanly produce content for a level in CRYENGINE.

Chapter 15, Maintaining Our Work, covers testing the content or fixing the bugs that take place in the last few minutes of a half hour level, which can be a repetitive and time-wasting process.
Getting Started

In this chapter, we will discuss the following topics:

- How to download the CRYENGINE "Blank" Game Starter-Kit
- How to install the starter-kit
- Important classes of the starter-kit and the roles they play
- How to compile a game project

Downloading the starter-kit

In order to create any CRYENGINE game, there is a need to have a clear place to start. For us, this comes as a blank slate using my very own starter-kit. By using it, we rid ourselves of overcomplications and gain several advantages:

- All code written is ours, so we have a clear and precise understanding of what the code does.
- All code written is specifically for our game, so we don’t have unnecessary code bloat.
- Since all of our code is exactly and only what we need, our code will run faster.
- All starter-kit code is written cleanly and is heavily documented. This allows us to have a better understanding of how to use CRYENGINE's interfaces.
Now that we know why we will be using my starter-kit, let's see how to download it:

1. Navigate to the download page at http://www.cryengine.com/community/viewtopic.php?f=355&t=124265. It should look like this:

2. Scroll down the page to the Download section and download Version 2.1.0 by clicking on its link.
For the sake of clarity and consistency, even if there is a newer version of the starter-kit available, it is strongly recommended that you stick with Version 2.1.0. Version 2.1.0 of the starter-kit is fully compatible with CRYENGINE Version 3.8.1 and, as such, it is strongly recommended that you use that version for the remainder of this book.

Installing the starter-kit
Before the starter-kit can be used, it must be installed. Installing the starter-kit is very easy and it is responsible for adding a project wizard to Visual Studio and setting up the needed CRYENGINE environment variables. So let’s get started.

Part 1
Now that we have downloaded the starter-kit, it’s time to install it:

1. Unzip the downloaded archive using 7-Zip (http://www.7-zip.org/) or any other file archiver that supports the 7z file format with LZMA2 compression.
2. You should now see an application (.exe) called CRYENGINE Game Starter-Kit - Blank Installer V2.0.8.exe; launch it. It should look like this:
3. Click on the **Browse** button to search for the path to CRYENGINE's Code folder. It should look like this:

![Browse For Folder Image](image)

It is important to note that CRYENGINE's Code folder is the one that contains all of CRYENGINE's code. It contains the CryENGINE, GameSDK, and SDKs subfolders. In the event that you do not have a Code folder at all, you will need to extract CRYENGINE_GameCode_xxxxx or a similar zip archive located in CRYENGINE's root folder. It is very important that the Code folder is in the right location. There should be exactly 2 subdirectories in CRYENGINE's root folder, like this: CRYENGINE_ROOT/CRYENGINE_pc_eaascode/Code.
4. Check all of the **Visual Studio** versions you want to install the starter-kit to, shown here:

![Image of Visual Studio version selection]

5. Lastly, click on the **Install** button to actually install the starter-kit. After the installation completes, you will get the following message:

![Image of installation completion]

---

**Part 2**

Now that we have successfully installed the CRYENGINE "Blank" Game Starter-Kit into Visual Studio, we can create our CRYENGINE game project:

1. Launch your preferred and supported Visual Studio IDE.

For this book, I will be using **Visual Studio 2013 Ultimate**. You are free to use any supported version you wish. However, you may notice some slight differences. Current supported versions of Visual Studio include any edition of 2012, 2013, and 2015.
2. Now that we have launched Visual Studio, we can create our game project by using the CRYENGINE "Blank" Game Starter-Kit we installed earlier. To do this, go to File | New | Project. You will see the following menu options:

3. Go to Templates | Visual C++ | CRYENGINE. Select CRYENGINE Game Starter-Kit - Blank and fill in your project information.
You are free to choose whatever name and location you wish. However, I strongly recommend you choose a location outside your CRYENGINE install directory. Throughout this book, we will be calling our game project GAS, which stands for Great Action Side-Scroller. Make sure to uncheck Create directory for solution.

4. Click the OK button. This will launch the CRYENGINE game starter-kit Project Creation Wizard. It should look like this:

5. Fill in all of the information for your game project, as shown here:
Since we have chosen to call our game GAS, it is recommended to set **Project Name** as GAS too. Also, it is strongly advised that you check the **Include PluginSDK** and **Set As Active Game** options. Including the PluginSDK developed by Hendrik Polinski allows users to add plugins to their game without modifying any actual game code. The advantage is that everything is modularized and features can be bundled up and shared between many projects. If you would like to get in contact with Hendrik, you can do so at https://github.com/hendrikp. Setting our game as active allows CRYENGINE to detect and use our game by modifying the `system.cfg` file in CRYENGINE’s root folder. Although this is all advised, you are free to fill in the data however you wish.

6. Click on the **Create** button. After the project has been created, you will get the following screen:
Important classes within the starter-kit

As we have concluded the setting up of the starter-kit, it is probably a good time to bring up a few disclaimers:

- While starter-kits in general are extremely helpful, they are technically not a true blank slate.
- There are a few interfaces and classes that will need to be explained before writing code.
- Since there is absolutely no gameplay-specific or genre-specific code or classes, the gameplay mechanics and many other things must be implemented from scratch. Although I consider this to be a good thing, some may not.

To address the previous concerns, let's take a closer look at some of the more important classes within the starter-kit that we will be using on a regular basis, keeping in mind that the names of these classes will be different for you as they are based on the project's name:

- CEditorGASGame: This class implements CRYENGINE's IEditorGame interface. It is used to create, update, and shut down your game while you are running it inside the CRYENGINE Sandbox Editor. You often need your game to behave differently while testing inside the editor; this class allows us to do just that. It acts as a proxy for our game, giving us the ability to add or remove functionality that should only exist while playing your game in the editor. For now, we simply forward most calls to the actual Game class so that our game behaves similarly in the editor and in the Launcher.

- CGASGame: This class implements CRYENGINE's IGame interface. It is used to update your game and the rest of CRYENGINE's systems. It acts as a hub that manages every system that your game will use, controls communication between them, and facilitates all of the core mechanics of your game. You can view this class as your game's manager—something that orchestrates all of the little moving parts of your game. For now, we simply update the Game Framework, which, in turn, updates the rest of CRYENGINE's systems such as the Renderer, Input Manager, and Physics System.

- CGASGameRules: This class implements CRYENGINE's IGameRules interface. It is used to carry out your game's rules and works hand-in-hand with your Game class. It is a class that does and should receive notifications about any gameplay-specific events so that it may decide how best to handle them in accordance with your game's rules. It is also common for this class to dispatch gameplay-specific events to your Game class. For now, this class simply creates the player when asked to do so.
• CGASModule: This class is a completely custom class in that it doesn't derive from any CRYENGINE interface. I created this class to help you; by definition, it is a helper class. Its sole purpose is to provide project-wide access to all of your game's systems and instantiate them only when needed. It does so by exposing itself as a global pointer called g_pCGASModule, which can be used inside any scope, inside any class, and inside any file to retrieve/create any of your game's systems.

It is strongly advised that if you create a custom system for your game, you should add and implement a singleton get method, as shown in the other get methods in this class.

• CGASStartup: This class implements CRYENGINE's IGameStartup interface. It is used to create and initialize all of CRYENGINE's systems and your Game class. This class is instantiated automatically from outside your Game DLL by the Launcher, and is expected to create and initialize all of CRYENGINE's modules and ultimately create and run your game. For now, we load CryGameFramework.dll, retrieve its exported factory method, and call it to instantiate an instance of the IGameFramework interface. We then proceed to initialize it, which, in turn, loads, creates, and initializes all of CRYENGINE's modules.

• CPlayer: This class implements CRYENGINE's IActor interface. Every CRYENGINE game needs to have a player implementation. This is the place to implement the logic that concerns the player, such as moving around the world and interacting with objects.

• CSmoothCamera: This class implements CRYENGINE's IGameObjectExtension and IGameObjectView interfaces. It's used to provide a view for our player so that we may see the world. The CSmoothCamera class implements IGameObjectView so that it may control a CRYENGINE IView instance. It implements IGameObjectExtension so that they may be added to a game object that's in the game's world. This class is slightly advanced and it would be better to read the in-code documentation to get a clearer understanding of how this class works.

• IWindowEventSystem: This interface is a completely custom interface in that it doesn't derive from any CRYENGINE interface. I created this interface to help you; by definition, it is a helper interface. Its sole purpose is to provide a mechanism for which to dispatch and handle various window events, such as window activation, closing, and various mouse/keyboard events.
• **CScriptBind.Game**: This class implements CRYENGINE's `CScriptableBase` class. It's used to expose your game's functionality to Lua. Although not required, any game-specific functionality you want to be exposed to Lua should be added here. For now, only the ability to set the game's current spawnpoint has been exposed.

By now, you should have a good understanding of what each of the classes do and what they are used for. For a more thorough understanding, you may want to take a look at the in-code documentation.

## Compiling our game

Now that we have all of the setup ready and have a clear understanding of what all of the important classes in our game do, let's compile our project. Get ready to venture into the world of game programming and create the *Great Action Side-Scroller*.

Right-click on your game project in the **Solution Explorer** and click on **Build**, as shown here:
Summary
This concludes the chapter. If you come across any issues while compiling the game project, I suggest that you take a look at the starter-kit Crydev forum thread at \url{http://www.cryengine.com/community/viewtopic.php?f=355&t=124265}. In this chapter, you learned how to download and install my CRYENGINE "Blank" Game Starter-Kit, what all of the important classes in the kit do, and how to compile a CRYENGINE game project. The future is very bright as we move forward with the rest of the book. Good job!
Where to buy this book
You can buy CRYENGINE Game Development Blueprints 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.