Roblox Game Development: From Zero to Proficiency (Beginner)

A Step-by-Step Guide to Building Your First Games in Roblox Studio with Lua.

Patrick Felicia

OceanofPDF.com

Roblox Game Development: From Zero to Proficiency (Beginner)

Copyright © 2024 Patrick Felicia

All rights reserved. No part of this book may be reproduced, stored in retrieval systems, or transmitted in any form or by any means, without the prior written permission of the publisher (Patrick Felicia), except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either expressed or implied. Neither the author and its dealers and distributors will be held liable for any damages caused or alleged to be caused directly or indirectly by this book.

Published by Patrick Felicia

OceanofPDF.com

Credits

Author: Patrick Felicia

OceanofPDF.com

About the Author

Patrick Felicia is a lecturer and researcher at Waterford Institute of Technology, where he teaches and supervises undergraduate and postgraduate students. He obtained his MSc in Multimedia Technology in 2003 and PhD in Computer Science in 2009 from University College Cork, Ireland. He has published several books and articles on the use of video games for educational purposes, including the Handbook of Research on Improving Learning and Motivation through Educational Games: Multidisciplinary Approaches (published by IGI), and Digital Games in Schools: a Handbook for Teachers, published by European Schoolnet. Patrick is also the Editor-in-chief of the International Journal of Game-Based Learning (IJGBL), and the Conference Director of the Irish Symposium on Game-Based Learning, a popular conference on games and learning organized throughout Ireland.

OceanofPDF.com

Download The Color Images

In case, you have a black and white version of the book, or just a low resolution, you can download a file that includes a color version of all the figures and screenshots included in this book here:

http://www.learntocreategames.com/book_downloads/roblox/book1/color_images.pdf

OceanofPDF.com

Share Your Feedback

Once you have read this book, I would love to hear your feedback. So please click here to access the review page for the book and leave your feedback; this will help other readers, and it will also help me to monitor and improve the book's quality.

OceanofPDF.com

Download a Free PDF Copy of This Book

Thanks for purchasing this book.

So that you can read your book on any of your devices with color pictures, you can download the pdf of this version for free; along with your free PDF book, you will also gain access to discounts, newsletters, and great free and useful content in your inbox every week and access to the book's companion course with video tutorials, extra resources and access to a community of like-minded programmers.

To do so, you can send me an email at learntocreategames@gmail.com, with your proof of purchase and you will receive your pdf version.

OceanofPDF.com

This book is dedicated to Mathis

OceanofPDF.com

Table of Contents

OceanofPDF.com

Contents

Credits

About the Author

Download The Color Images

Share Your Feedback

Download a Free PDF Copy of This Book

Table of Contents

Preface

Content Covered by this Book

What you Need to Use this Book

Who this Book is for

Who this Book is not for

How you will Learn from this Book

Format of each Chapter and Writing Conventions
Special Notes

How Can You Learn Best from this Book

Feedback

Improving the Book

Supporting the Author

1 The Benefits of Using Roblox
What is Roblox?
Key Features of Roblox
Benefits of Using Roblox
The Evolution of Roblox
How Roblox Empowers Creators
The Future of Roblox
Summary

2 Getting Started with Roblox Studio
Installing Roblox Studio

Exploring the Interface
Basic Navigation and Tools
Keyboard Shortcuts
Customizing Your Workspace
Understanding Basic Components
Using the Toolbox
Exploring Your New Scene
Basic Scripting with Lua
Testing and Debugging Your Script
Saving and Loading Projects
Overview of Roblox Community and Resources
Summary and Review
Quiz

3 Building Basics
Understanding Part Properties
Grouping and Ungrouping Parts
Using the Anchor and Collision Properties
Creating Simple Structures and Maps
Using the Terrain Editor: Basics
Adding Water and Vegetation
Lighting Basics
Adding Sounds and Music
Summary
Quiz

Challenge

4 LUA Programming Concepts
Introduction
Statements
Comments

Variables
Operators
Conditional Statements
Combining if/elseif
Loops
Functions
Tables
Single-Dimensional Tables (Arrays)
Multidimensional Tables
Using Loops with Tables
Scope of Variables
Events
Understanding Events
Common Events in Roblox
Creating Custom Events
Practical Example of Events: Opening a Door
Summary
Quiz
Challenge

5 Introduction to Scripting in Roblox Studio
Starting to code in Lua
Using Variables
Using Basic Operators and Expressions
Using Conditional statements
Combining events and conditional statements
Using Arrays and Loops (For, While)
Debugging Scripts with the Output Window
Basic User Input Handling
Introduction to Services

Simple Game Mechanic: Collectible Items
Summary and Next Steps
Quiz
Challenge

6 Building Your First Interactive Game in Roblox Studio
Designing the Game Environment
Adding Collectible Items
Using Sound Effects
Creating an Interactive bridge
Adding a Script for the Dead Zone
Adding Moving Platforms
Creating a Generalized Script for All Boxes
Creating a Simple Scoring System
Summary
Quiz
Challenge

7 Creating and Managing Multiple levels
How to Implement Multiple levels
Structuring Levels
Creating a Checkpoint
Creating a Starting Point
Transitioning from the Checkpoint to the Second Level
Modifying the second dead zone
Creating and Displaying a Timer
Creating a Starting Screen
Checking for Winning Conditions
Displaying Messages to the user
Completing the End screen
Publishing and Sharing the Game

Summary
Quiz
Challenge

8 Answers to Quizzes
Chapter 2
Chapter 3
Chapter 4
Chapter 5
Chapter 6
Chapter 7

9 Frequently Asked Questions
Getting Started with Roblox Studio
Scripting and Game Mechanics
Section 3: Building and Object Manipulation
Section 4: Visual and Audio Enhancements
Section 5: Publishing and Sharing Your Game

10 Thank you

OceanofPDF.com

Preface

As someone with extensive experience teaching Unity and game development, I have always been passionate about helping others understand the core concepts of programming and game design. Over the years, I've seen how powerful platforms like Unity, Unreal or Godot, can be in fostering creativity and technical skills. However, I also realized that for many aspiring developers, especially beginners, there is a need for platforms that are even more accessible—both in terms of learning to code and quickly building games. That's when I turned my attention to Roblox.

Roblox provides a unique opportunity for learners of all ages to dive into game development without the steep learning curve that comes with some other tools. Its intuitive drag-and-drop interface, combined with the flexibility of Lua scripting, makes it a perfect starting point for anyone looking to get into game creation. This book series assumes no prior knowledge on the part of the reader, and it will get you started on Roblox so that you quickly master all the wonderful features that this software provides by going through an easy learning curve. By completing each chapter, and by following step-by-step instructions, you will progressively improve your skills, become more proficient in Roblox, and create a survival game using Roblox's core features in

terms of programming (i.e., Lua), game design, and drag-and-drop features.

This first book in the series is designed for absolute beginners. You will learn how to use Roblox Studio, understand its interface, and create interactive worlds. Along the way, you will develop essential skills like object manipulation, basic scripting, creating game environments, and understanding physics concepts such as collisions. You will also build up to a project where you create an interactive game with multiple levels, player objectives, and a scoring system.

In addition to mastering these core features, you will also explore techniques commonly used in video game development, including object creation, collision detection, lighting effects, and simple animations. By the end of the book, you will have a game where players need to collect items, navigate obstacles, and progress through levels based on specific objectives. This game will include a 3D platform game, giving you a well-rounded understanding of how to structure levels, manage player interactions, and create engaging experiences.

You will learn how to manipulate parts, add scripts to control game mechanics, design fun and challenging levels, and even implement conditions that allow players to advance or be reset based on their performance. Additionally, you will gain an understanding of how to create custom assets, such as items and obstacles, and integrate them

seamlessly into your game world. Finally, you will also get to export your game and share it with friends on the web, allowing them to experience the world you've created directly from the Roblox platform.

This book is designed to be both fun and educational. As you work through the chapters, you'll see your progress take shape in the form of a playable game that can be shared and expanded upon. I'm excited to be your guide on this journey, and I can't wait to see the amazing things you'll create with Roblox!

OceanofPDF.com

Content Covered by this Book

Chapter 1: The Benefits of Using This chapter introduces game engines and their importance in game development. It explains why Roblox is an excellent choice for beginners and experienced developers alike, due to its accessible interface and vast user base. You will discover the benefits of using Roblox, including its drag-and-drop features, ease of use, and how it empowers creators to build games seamlessly.
Chapter 2: Getting Started with Roblox In this chapter, you will learn how to install and set up Roblox Studio. It introduces you to the main components of Roblox Studio, such as the Explorer, Properties, and Toolbox. You will also learn how to navigate the workspace, customize your environment, and create your first project.
Chapter 3: Building Basics. This chapter teaches you how to build simple structures by creating, resizing, and manipulating parts. It covers essential properties like color, material, and transparency, as well as grouping and ungrouping objects. Additionally, you will learn how to create terrains, add basic lighting, and optimize your builds for performance.
Chapter 4: Lua Programming In this chapter, you will explore Lua, the programming language used in Roblox. You will learn about variables, operators, and basic Lua syntax. Furthermore, you will discover how

to control the flow of your code using if statements, loops, and functions, which are key to adding logic to your game.

Chapter 5: Introduction to Scripting in Roblox This chapter introduces scripting in Roblox Studio, building on the basics of Lua. You will learn how to use variables, arrays, and other programming tools to manage data in your game. Additionally, you will create simple game mechanics, such as collectible items, and learn how to debug your scripts using the Output window.

Chapter 6: Building Your First Interactive Game in Roblox In this chapter, you will design a basic game environment with collectibles, platforms, and sound effects. You will add interactivity by implementing scripts that respond to player actions, such as picking up objects or moving platforms. Moreover, you will learn how to create a scoring system and display it on the screen, enhancing player engagement.

Chapter 7: Creating and Managing Multiple This chapter focuses on creating and managing multiple levels in your game. You will learn how to add triggers and win conditions that determine when a player progresses to the next level or completes the game. Additionally, you will explore how to track player progress across levels and implement checkpoints to save progress between stages.

Chapter 8: Answers to This chapter provides answers to the quizzes found at the end of each chapter.

Chapter 9: Frequently Asked This chapter provides answers to frequently asked questions based on specific themes and topics (e.g., asset creation or transformations).

Chapter 10 summarizes the topics covered in this book and also provides useful information if you would like to progress further with this book series.

OceanofPDF.com

What you Need to Use this Book

To complete the project in this book, all you need is a free Roblox account and a computer!

Roblox supports both Windows and making it easy to get started no matter what kind of computer you have.

As for the technical details, don’t worry too much—most computers less than five years old should work just fine.

If you want to dive into the specifics like system requirements or graphic card details, you can check those out on the official Roblox website:

https://create.roblox.com/docs/studio/setting-up-roblox-studio

You don’t need any prior experience with programming or game development to follow along. The book is designed to be beginner-friendly, guiding you through simple tasks like downloading files, saving projects, and using drag-and-drop features in Roblox Studio. If

you’re comfortable with basic computer tasks, you’re all set to start creating amazing games!

OceanofPDF.com

Who this Book is for

OceanofPDF.com

If you can answer yes to these questions, then this book is for you:

Are you a total beginner in Roblox Studio, looking to create your first game?

Would you like to become proficient in Roblox's core functionalities, such as building, scripting, and game design?

Would you like to guide your students or children through the process of creating fun and interactive games?

Are you ready to learn how to build engaging 3D environments with levels and objectives?

Have you had some exposure to Roblox but want to deepen your understanding of Lua scripting and advanced game mechanics?

OceanofPDF.com

Who this Book is not for

If you can answer yes to these questions, then this book is not for you:

Can you already create 3D games in Roblox using built-in objects, controllers, cameras, and environments?
Are you seeking a detailed reference on advanced Roblox programming?
Are you an experienced or advanced Roblox user?

If so, you might prefer the next books in the series. To explore the content covered in those books, visit the official website at

OceanofPDF.com

How you will Learn from this Book

Because all students learn differently and have different expectations of a course, this book is designed to ensure that all readers find a learning mode that suits them. Therefore, it includes the following:

A list of the learning objectives at the start of each chapter so that readers have a snapshot of the skills that will be covered.
Each section includes an overview of the activities covered.
Many of the activities are step-by-step, and learners are also given the opportunity to engage in deeper learning and problem-solving skills through the challenges offered at the end of each chapter.
Each chapter ends-up with a quiz and challenges through which you can put your skills into practice, and see how much you know.
The book focuses on the core skills that you need. Some sections also go into more detail. However, once the concepts have been explained, links are provided to additional resources, if and where necessary.

OceanofPDF.com

Format of each Chapter and Writing Conventions

Throughout this book, and to make reading and learning easier, text formatting and icons will be used to highlight parts of the information provided and to make it more readable.

The full solution for the project presented in this book is available for download on the official website So if you need to skip a section, you can do so; you can also download the solution for the previous chapter that you have skipped.

OceanofPDF.com

Special Notes

Each chapter includes resource sections so that you can further your understanding and mastery of Roblox. These include:

A quiz for each chapter: these quizzes usually include 10 questions that test your knowledge of the topics covered throughout the chapter. The solutions are provided on the companion website.
Challenges: each chapter includes a challenge section where you are asked to combine your skills to solve a particular problem.

The author’s notes appear as described below:

Author’s suggestions appear in this box.

Checklists that include the important points covered in the chapter appear as described below:

OceanofPDF.com

How Can You Learn Best from this Book

Talk to your friends about what you are doing.

We often think that we understand a topic until we have to explain it to friends and answer their questions. By explaining your different projects, what you just learned will become clearer to you.

Do the exercises.

All chapters include exercises that will help you to learn by doing. In other words, by completing these exercises, you will be able to better understand the topic and you will gain practical skills (i.e., rather than just reading).

Don’t be afraid of making mistakes.

I usually tell my students that making mistakes is part of the learning process. The more mistakes you make and the more opportunities you have for learning. At the start, you may find the errors disconcerting, or that the engine does not work as expected until you understand what went wrong.

Export your games early.

It is always great to build and export your first game. Even if it is rather simple, it is always good to see it in a browser and to be able to share it with you friends.

Learn in chunks.

It may be disconcerting to go through five or six chapters straight, as it may lower your motivation. Instead, give yourself enough time to learn, go at your own pace, and learn in small chunks (e.g., between 15 and 20 minutes per day). This will do at least two things for you: it will give your brain the time to “digest” the information that you have just learned, so that you can start fresh the following day. It will also make sure that you don’t “burn-out” and that you keep your motivation levels high.

OceanofPDF.com

Feedback

While I have done everything possible to produce a book of high quality and value, I always appreciate feedback from readers so that the book can be improved accordingly. If you would like to give feedback, you can email me at

OceanofPDF.com

Improving the Book

Although great care was taken in checking the content of this book, I am human, and some errors could remain in the book. As a result, it would be great if you could let me know of any issue or error you may have come across in this book, so that it can be solved and the book updated accordingly. To report an error, you can email me with the following information:

Name of the book.
The page where the error was detected.
Describe the error and also what you think the correction should be.

Once your email is received, the error will be checked, and, in the case of a valid error, it will be corrected, and the book page will be updated to reflect the changes accordingly.

OceanofPDF.com

Supporting the Author

A lot of work has gone into this book, and it is the fruit of long hours of preparation, brainstorming, and finally writing. As a result, I would ask that you do not distribute any illegal copies of this book.

This means that if a friend wants a copy of this book, s/he will have to buy it through the official channels (i.e., through Amazon, lulu.com, or the book's official website:

If some of your friends are interested in the book, you can refer them to the book's official website where they can either buy the book, enter a monthly draw to be in for a chance of receiving a free copy of the book, or to be notified of future promotional offers.

OceanofPDF.com

1

The Benefits of Using Roblox

This chapter is an introduction to Roblox, explaining what Roblox is, its key features, and why it's a powerful platform for creating games. We will explore the basics of the Roblox ecosystem, including how it supports creativity and provides a platform for both playing and creating games.

After completing this chapter, you should be able to:

Understand what Roblox is and its purpose.
Know the key features of Roblox and how they benefit developers.
Recognize the potential of Roblox as a platform for both game creation and gameplay.



OceanofPDF.com

What is Roblox?

Roblox is a global online platform that allows users to play and create games. It was launched in 2006 and has since grown into one of the largest social gaming platforms, where users can build and share their own games using Roblox Studio. Roblox provides a vast, immersive world where millions of people connect daily to explore, interact, and create.

OceanofPDF.com

Key Features of Roblox

Roblox includes a wide range of features, some of which are also found in game engines:

User-Generated Content The heart of Roblox is its UGC, where users create games and experiences. This feature has made Roblox one of the most diverse gaming platforms, with over 40 million games created by its community.
Roblox A free, powerful tool that allows users to create games and experiences with ease. It offers an intuitive interface and a wide array of assets and scripts that enable both beginners and advanced developers to bring their ideas to life.
Cross-Platform Roblox supports gameplay across various devices, including PCs, smartphones, tablets, and consoles, allowing players to access games anywhere, anytime.
Virtual Roblox has a thriving virtual economy, where developers can monetize their games through in-game purchases, advertisements, and game passes. The virtual currency, Robux, can be used to buy items, and developers can convert Robux into real money.

Social Roblox is not just a gaming platform but also a social platform. It includes features like chat, friends lists, and groups, making it easy

for players to connect and collaborate.

OceanofPDF.com

Benefits of Using Roblox

Roblox includes a wide range of benefits for those who want to create their own games, and possibly learn to code in the process; one of its most interesting feature include:

Roblox is accessible to users of all skill levels. Whether you are a beginner or an experienced developer, Roblox Studio provides tools and resources that cater to your needs.
Roblox With millions of active users and a robust developer community, you can find plenty of resources, tutorials, and forums to help you learn and improve your game development skills.
Monetization Roblox provides various ways to earn money through your games, making it a viable platform for aspiring game developers to turn their hobby into a career.

Given all these features, it’s easy to see how Roblox has become one of the most popular online game engine and game creation platforms.

OceanofPDF.com

The Evolution of Roblox

Since its inception, Roblox has continuously evolved, introducing new features and expanding its platform. Initially designed as a basic building tool, it has grown into a comprehensive development platform that supports complex games and experiences.

Key Milestones in Roblox's History:

Roblox is launched, offering basic building tools and physics-based gameplay.
Introduction of the Developer Exchange (DevEx) program, allowing developers to convert Robux into real currency.
Roblox Studio becomes more advanced with the addition of complex scripting tools and physics engines.
Roblox hits 48 million monthly active users, highlighting its growing popularity.
Introduction of the Roblox Education initiative, providing resources and curricula for educators to teach coding and game design using Roblox.
Roblox surpasses 150 million monthly active users, and the platform continues to expand with virtual concerts, educational experiences, and branded content.

Roblox stands out due to its emphasis on user-generated content and its community-driven ecosystem. Unlike traditional game development platforms, Roblox democratizes game creation by providing all the necessary tools for free and fostering a collaborative environment where users can learn from each other.

OceanofPDF.com

How Roblox Empowers Creators

Roblox empowers creators by providing a platform where imagination is the only limit. Whether you want to create a simple obstacle course or a complex multiplayer game, Roblox Studio offers the tools you need to bring your vision to life.

Roblox Studio includes some very useful features that make it easy and straight-forward to create and share games; these include::

Drag-and-Drop Users can easily create and modify game objects without needing advanced technical skills.
Lua, a lightweight programming language, helps to add interactivity and complex mechanics to games created with Roblox Studio.
Asset Roblox Studio provides access to a vast library of free models, textures, and sounds to enhance your game.
Collaborative With Roblox, you can work with other developers in real-time to create and improve your games.
Roblox also makes it possible for you to track your game's performance and user engagement with built-in analytics tools.

In addition to Roblox Studio, it is possible to generate money from your game using the following:

Game You can sell exclusive items or abilities that give players special privileges in your game.
In-Game Players can buy virtual items sing Robux.
You can display ads in your games to generate additional revenue.

OceanofPDF.com

The Future of Roblox

Roblox continues to innovate and expand, with plans to introduce even more advanced tools and features in the future. The platform aims to remain at the forefront of the gaming industry by constantly evolving and adapting to the needs of its users.

Upcoming Features and Trends:

Virtual Reality Roblox is increasingly supporting VR experiences, allowing users to create and explore immersive environments.
Educational As part of its educational initiative, Roblox is being integrated into more school curricula worldwide, teaching students coding and game design.
Global Roblox is expanding its reach into new markets, making the platform accessible to even more users around the globe.
Enhanced Social Roblox plans to introduce new social features that will make it easier for users to connect, collaborate, and share their creations.

Roblox is more than just a gaming platform; it's a creative ecosystem where users can play, create, and share their own games. With its powerful tools, supportive community, and endless possibilities,

Roblox offers a unique opportunity for aspiring game developers to hone their skills and bring their ideas to life.

OceanofPDF.com

Summary

This chapter introduced you to the world of Roblox, covering its key features, the benefits of using the platform, and how it empowers creators. You also learned about the history and evolution of Roblox, as well as the exciting opportunities it offers for game developers. With this foundation, you are now ready to dive deeper into the specifics of game development in Roblox, starting with the next chapter on building basics.

OceanofPDF.com

2

Getting Started with Roblox Studio

In this section, we will cover the installation and initial setup of Roblox Studio, explore the main interface components, learn basic navigation and tool usage, and customize the workspace to fit personal preferences. So, after completing this chapter, you will be able to:

Install Roblox Studio (the tool we will use to create your game) on your computer.
Navigate and understand the Roblox Studio interface.
Use basic tools and navigation techniques.
Customize your workspace for a personalized setup.
Create your first project and understand basic components.
Perform simple object manipulations.
Utilize the Toolbox for pre-made assets.
Write and run your first script in Lua.

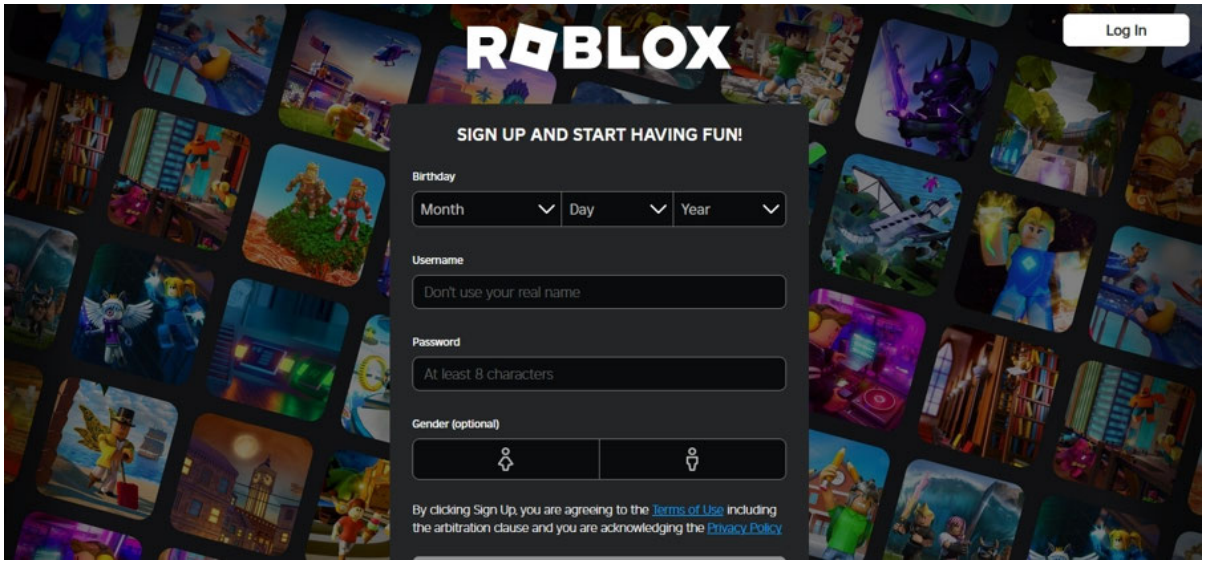
OceanofPDF.com

Installing Roblox Studio

In this section, I will guide you through the steps to install Roblox Studio on your computer, ensuring that you have all the necessary tools to start your game development journey.

Let's start with the installation process:

Visit the Roblox website and navigate to the Roblox Studio download page: Open your web browser and go to

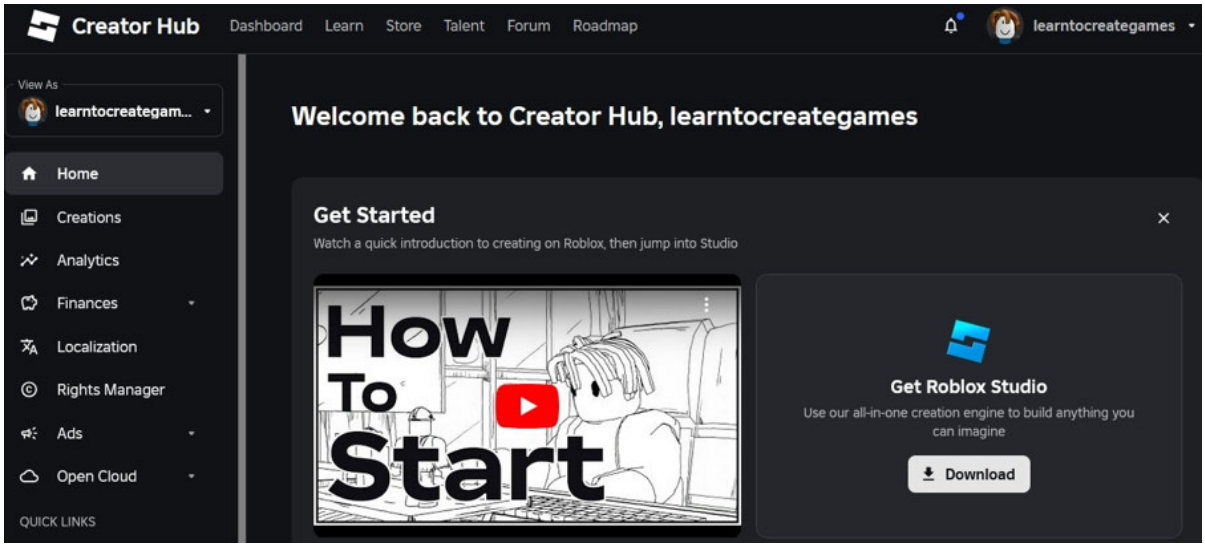


If you already have a Roblox account, you can press the button, otherwise, please go through the registration process and follow the instructions.

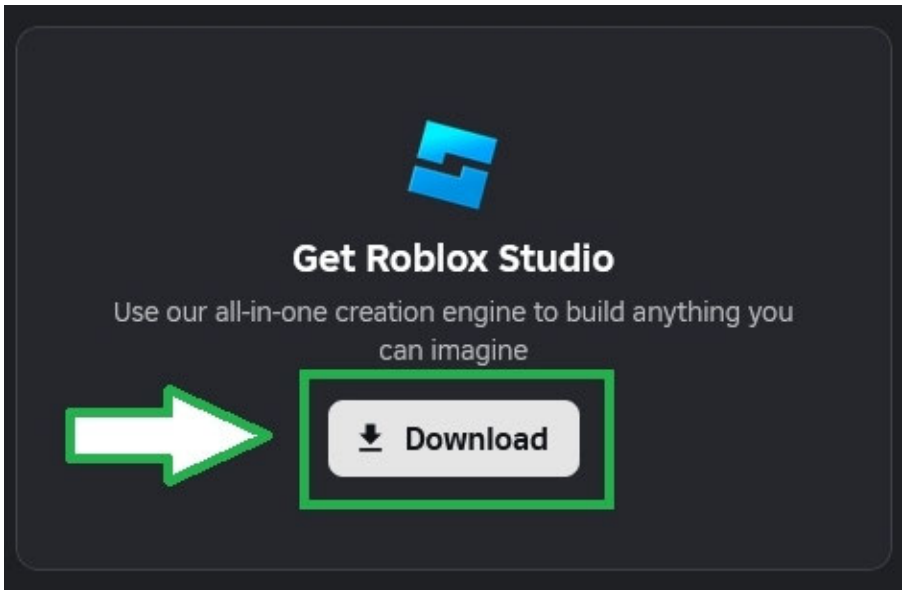
Once this is done, you can start to create your first content: Click on the Create button at the top of the page.



A new window will appear, as illustrated in the next figure.

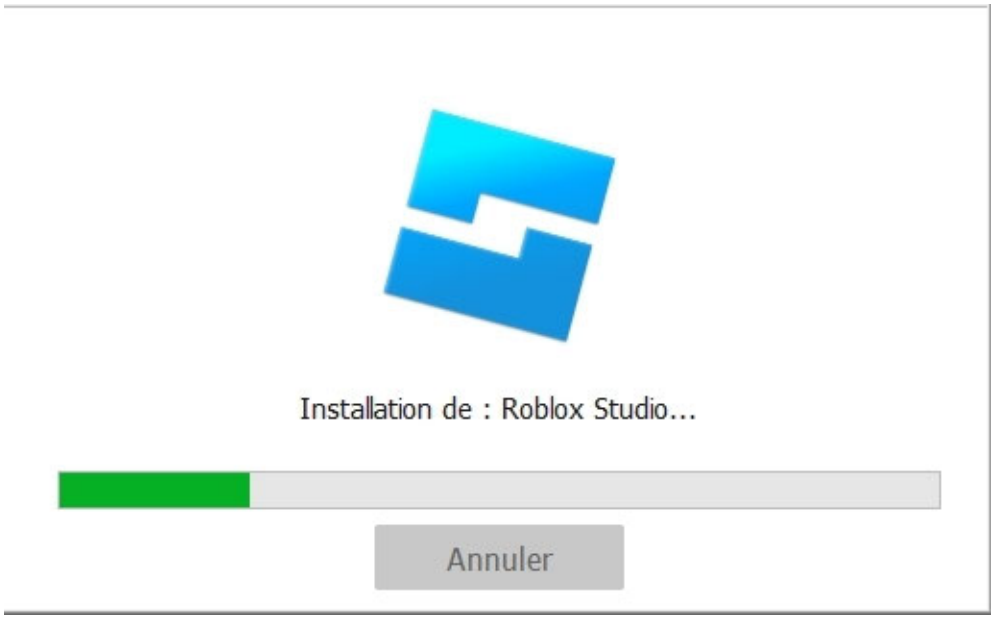


Download the Roblox Studio installer appropriate for your operating system (Windows or macOS): Click the button in the section Roblox



Save the installer file to your preferred download location on your computer.

Run the installer and follow the on-screen instructions to complete the installation: Locate the downloaded installer file and double-click it to run the installer.

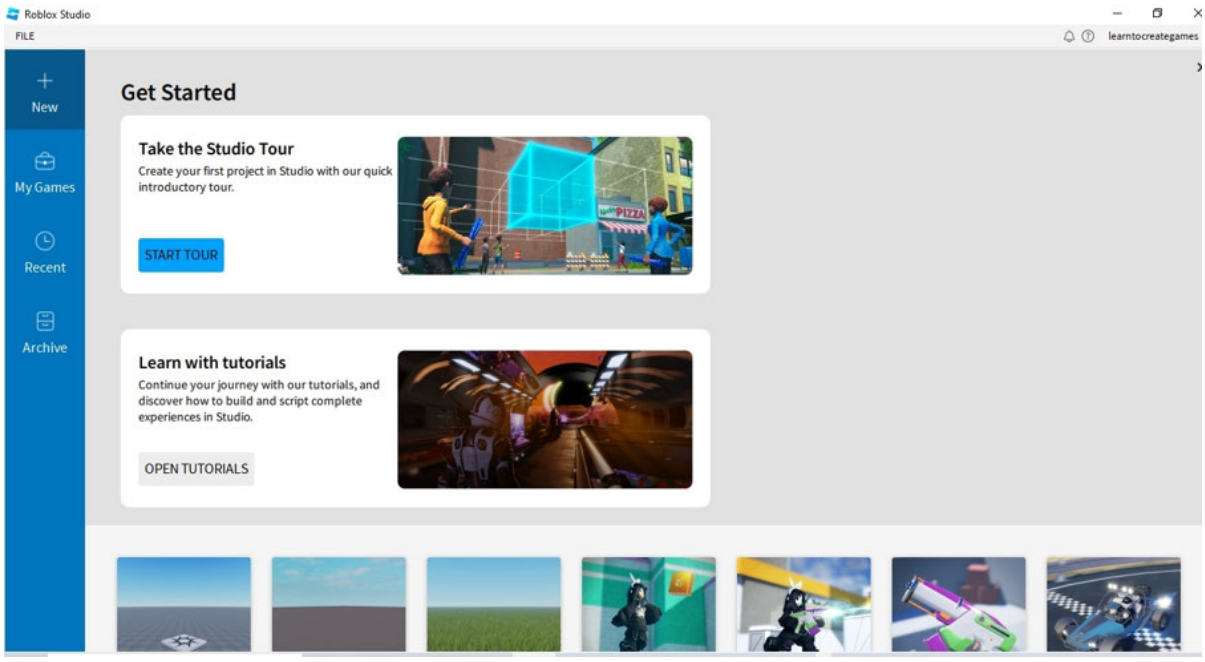


Follow the on-screen prompts to install Roblox Studio on your computer.

Once installed, launch Roblox Studio and sign in with your Roblox account. If you don't have an account yet, you will need to create one.

Open Roblox Studio from your desktop or start menu.
Enter your Roblox username and password, then In (same information as for your online account on
If you don't have an account, Up and follow the instructions to create a new account.

After completing these steps, you should see that Roblox Studio is installed and ready to use on your computer, as per the next figure.



OceanofPDF.com

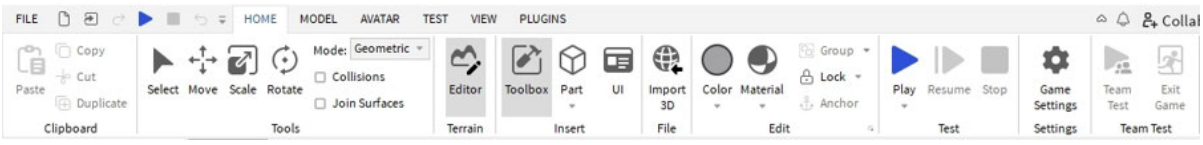
Exploring the Interface

In this section, we will explore the main components of the Roblox Studio interface to help you become familiar with the environment.

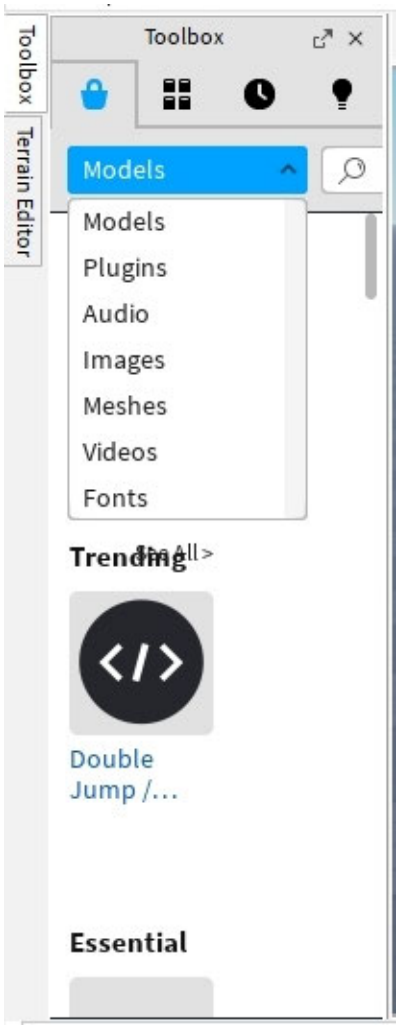
Please select: File | this will open a new project where you can see the Roblox Studio interface.

The Roblox Studio interface consists of several key elements:

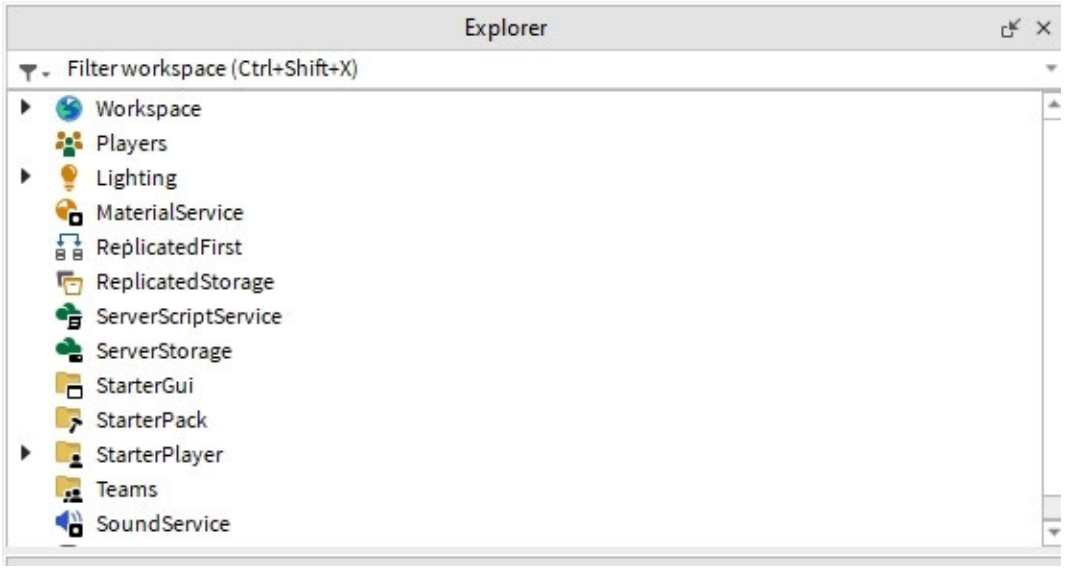
Toolbar: Located at the top, it includes options such as File, Edit, View, and Plugins.



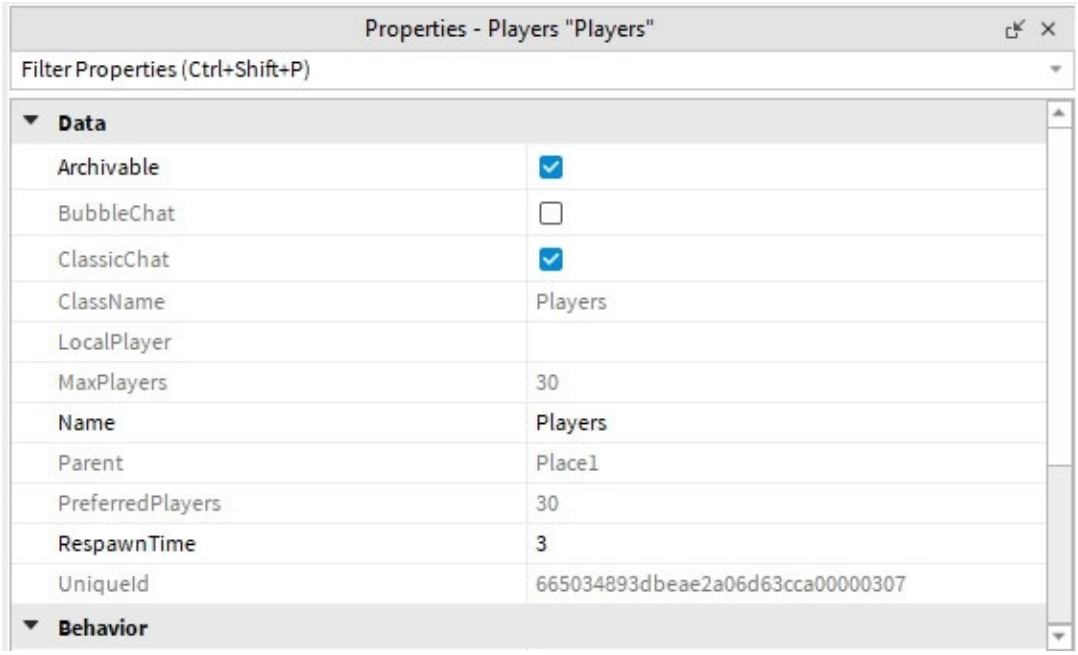
Toolbox: Found on the left side, it provides access to various models, scripts, and other assets.



Explorer: This panel, usually on the right, shows the hierarchy of objects in your game.



Properties: Below the this panel displays the properties of the selected object.



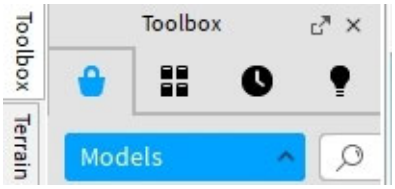
Let's examine each component in more detail:

The Toolbar: You can use the Toolbar to access different functions such as saving your project | editing objects, and changing the view.



File Menu: Provides options to create, open, and save projects.
View Tab: Allows you to toggle different panels and tools on and off.
Plugins Tab: Enables you to manage and use various plugins.

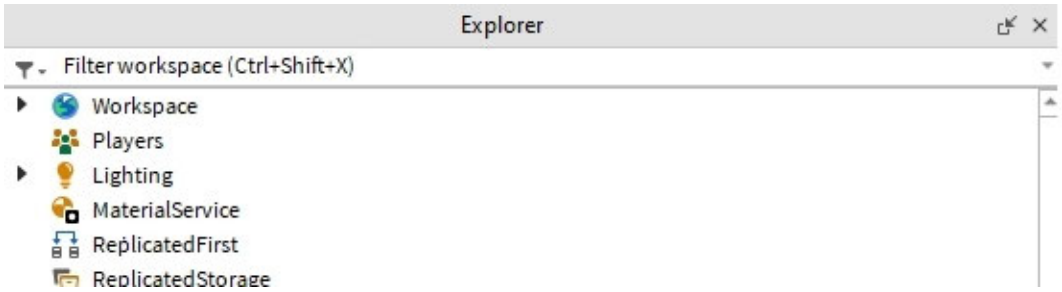
Toolbox: you can browse through the toolbox to find pre-made models and assets that you can use in your game.



To use the toolbox, you can do the following:

Click on the Toolbox tab on the left side of the screen.
Search for assets using the search bar at the top of the toolbox.
Drag and drop assets from the toolbox into the game workspace.

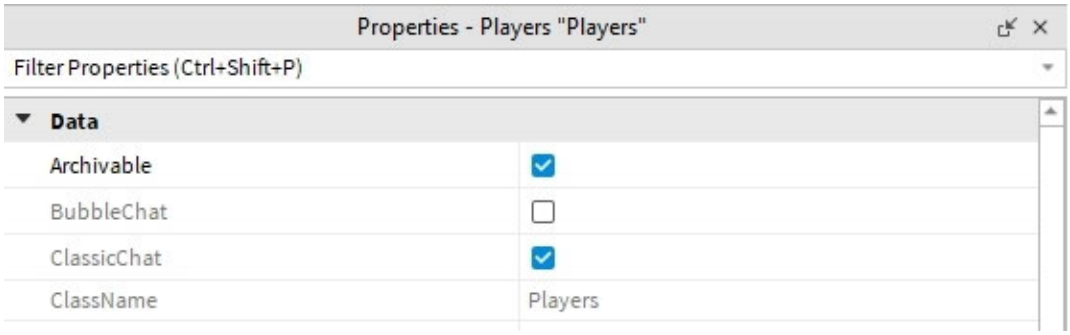
The Explorer: The Explorer helps you manage all the objects in your game scene.



To use the toolbox, you can do the following:

Click on the Explorer tab on the right side of the screen.
Expand and collapse objects to see their components.
Right-click on objects to access additional options like renaming and deleting.

Properties: You can adjust the properties of selected objects, such as size, color, and position.



To use the Properties tab, you can do the following:

Click on an object in the Explorer and then see its properties in the Properties panel.

Modify properties by typing in new values or using the provided controls.

You should now be familiar with the basic components of the Roblox Studio interface. This familiarity is important as it will allow you to efficiently navigate and utilize the tools available for creating and managing your game projects.

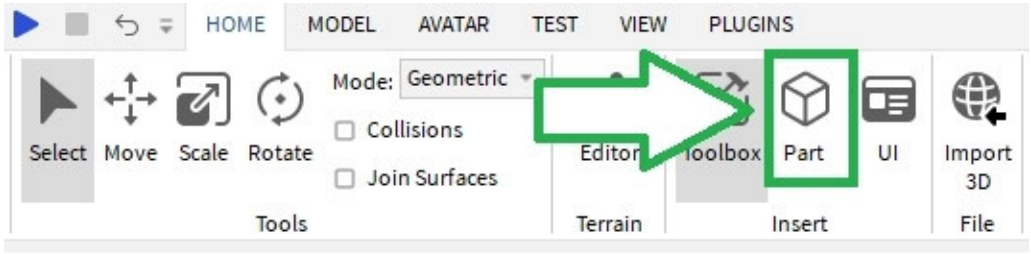
OceanofPDF.com

Basic Navigation and Tools

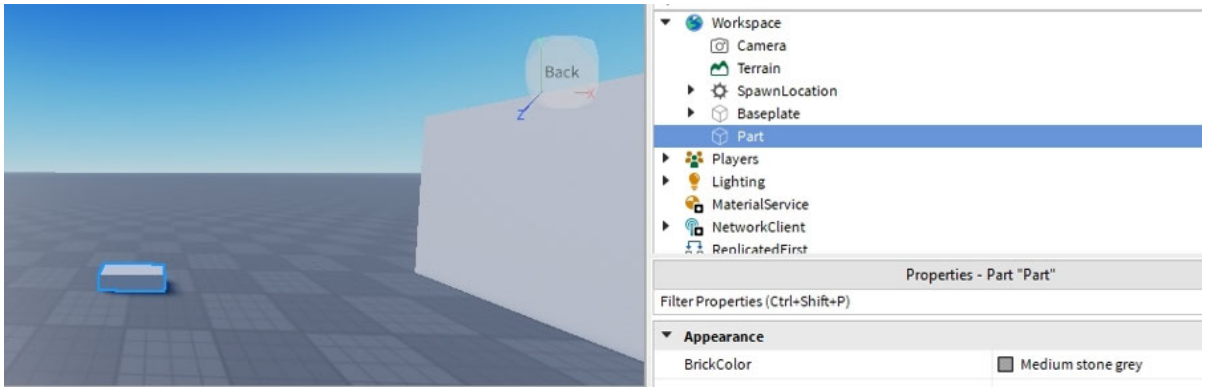
In this section, we will cover basic navigation techniques and essential tools within Roblox Studio to help you move around and manipulate objects in your game.

Let's explore the basic navigation tools:

Please create a new object (File |
Click on the Home tab from toolbox and select: Part |



This will create a new object called Part that you can see both in the and in the 3D



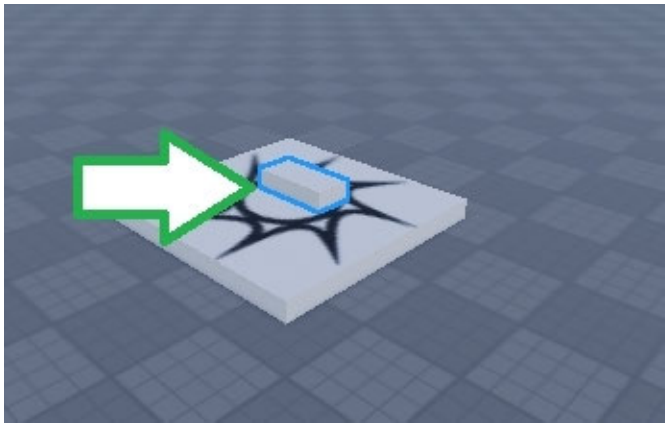
Now that we have created this object, we will operate a series of transformation on same:

Select Tool:

Click the Select Tool from the Home tab at the top of the screen, as per the next figure.

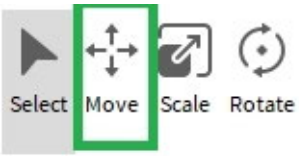


Click on the object called Part in the 3D Viewport to select it.
Drag the object to move it to a new location. As you do, you should see a hand on this object, symbolizing the fact that you can drag the object.

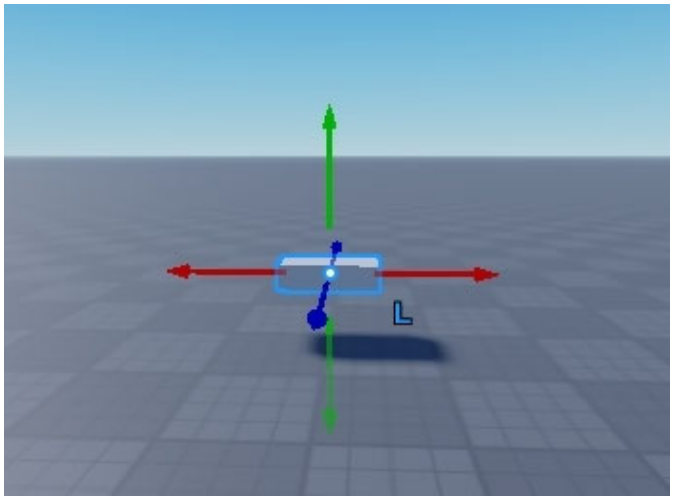


Move Tool to Move Objects:

Click the Move Tool from the Home tab.



Click on the object called
Drag the arrows that appear around the object to move it along the corresponding axis: the green axis to move the object along the y-axis (up or down), the red axis to move the object along the x-axis (left or right), and the blue axis to move the object along the z-axis (forward or back).

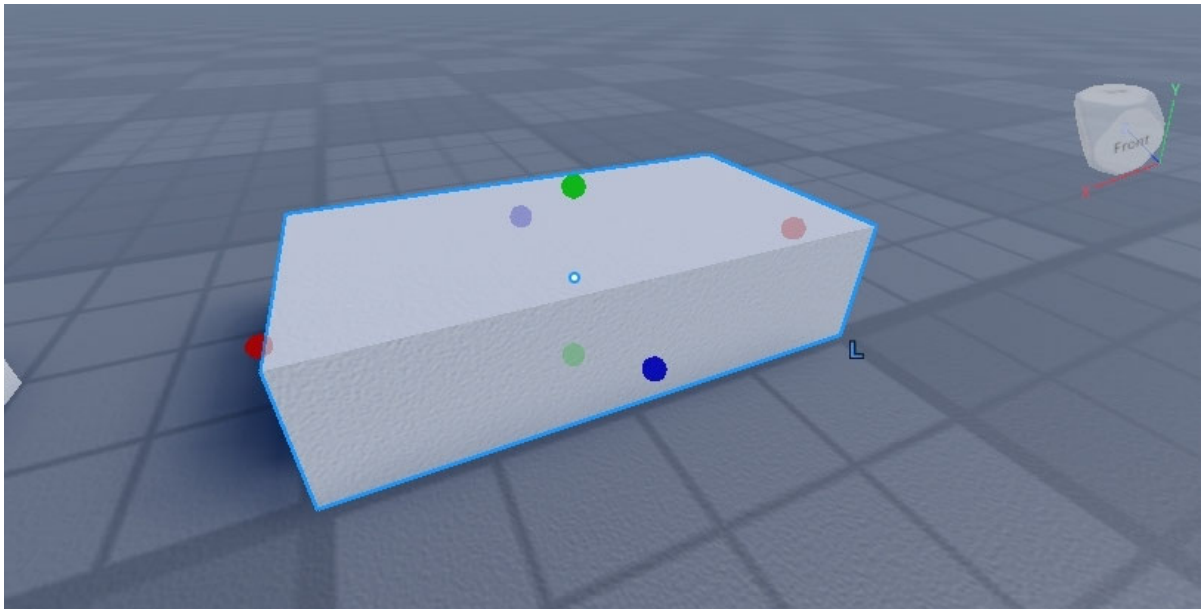


Scale Tool: Resize objects using the Scale tool.

Click the Scale Tool from the Home tab.

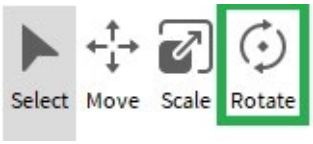


Click on the object called
Drag the circles that appear around this object to resize it: the green dots to scale the object along its y-axis (up or down), the red dots to scale the object along its x-axis (left or right), and the blue dots to move the object along its z-axis (forward or back).

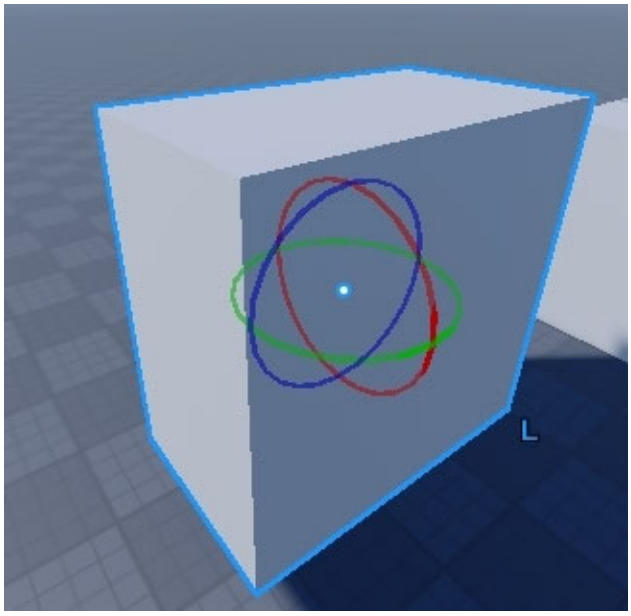


Rotate Tool objects and adjust their

Click the Rotate Tool from the Home tab.



Click on the object called
Drag the circles that appear around this object to resize it: the green circle to rotate the object along its the red circle to rotate the object along its and the blue circle to rotate the object along its



Now that we have managed to get to grips with basic transformation tools, we will look into how to navigate the workspace.

To navigate the workspace:

Once the object is selected (in the 3D Viewport or the press the key this will focus the camera on that object.

Right-click and drag the mouse to change your view angle: Hold down the right mouse button and move your mouse to look around.
Press the Mouse Wheel and drag the mouse to Pan the

Use the scroll wheel on your mouse to zoom in and out of the workspace.

Use the arrow keys or W, A, S, D keys to move around the 3D to move forward, S to move backward, A to move left, and D to move right.

After completing these steps, you should see that you can easily navigate and manipulate objects within Roblox Studio. This ability is essential for creating and editing game environments and objects, allowing you to position and adjust them accurately.

OceanofPDF.com

Keyboard Shortcuts

Keyboard shortcuts are essential tools that can significantly speed up your workflow in Roblox Studio. By using shortcuts, you can quickly access tools and commands without having to navigate through menus.

In the next steps, we will learn some of the most used keyboard shortcuts in Roblox Studio. These shortcuts will help you work more efficiently by reducing the time spent on repetitive tasks.

Please do the following:

Move Tool Press CTRL + 2 on your keyboard to quickly select the Move tool. This tool allows you to reposition parts in the workspace.
Scale Tool Press CTRL + 3 on your keyboard to switch to the Scale tool. This tool allows you to resize parts in any direction.
Rotate Tool Press CTRL + 4 on your keyboard to access the Rotate tool. Use this tool to change the orientation of parts.
Duplicate Select an object and press Ctrl + D to duplicate it. This is useful for quickly creating multiple copies of an object.
Undo Press Ctrl + Z to undo your last action. This is helpful if you make a mistake and need to revert to the previous state.

Now that you've learned these essential keyboard shortcuts, you'll be able to work more efficiently in Roblox Studio. In the next section, we'll explore the resources and communities available to you as a Roblox developer.

OceanofPDF.com

Customizing Your Workspace

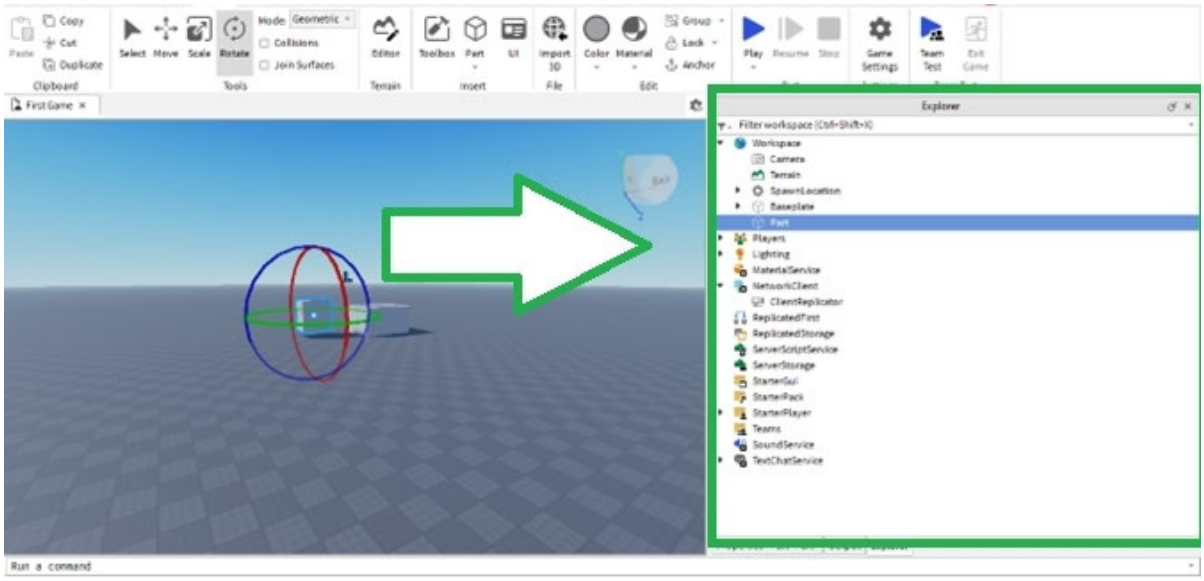
In this section, we will learn how to customize your workspace to create a comfortable and efficient working environment.

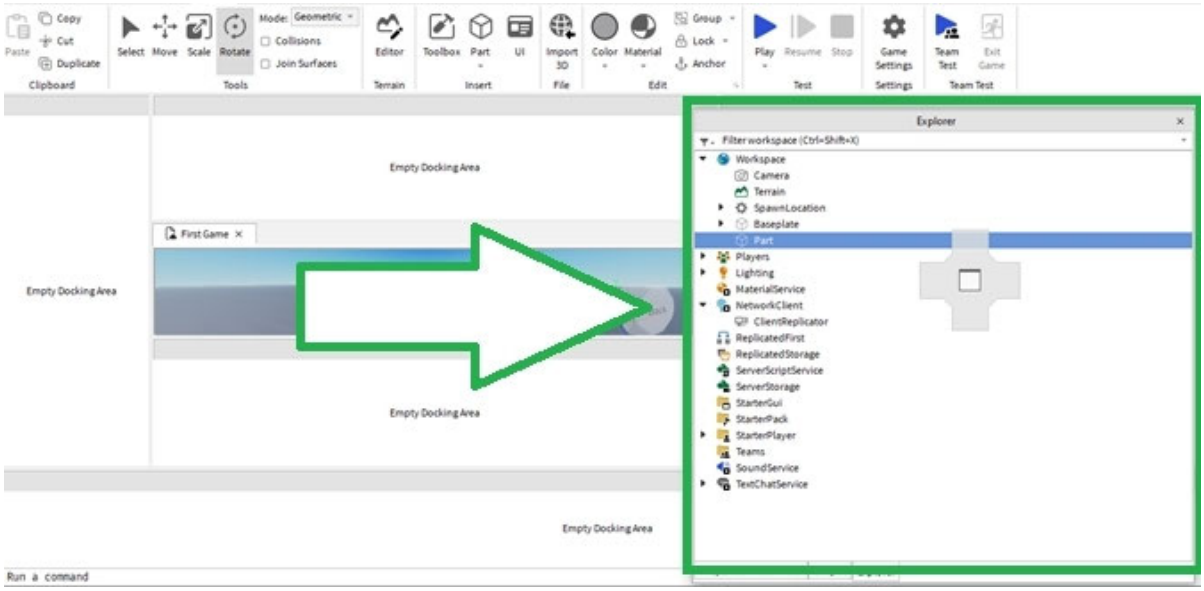
There are several ways to customize your environment in Roblox, and these include: adjusting panels, adding or removing panels, and saving your layout.

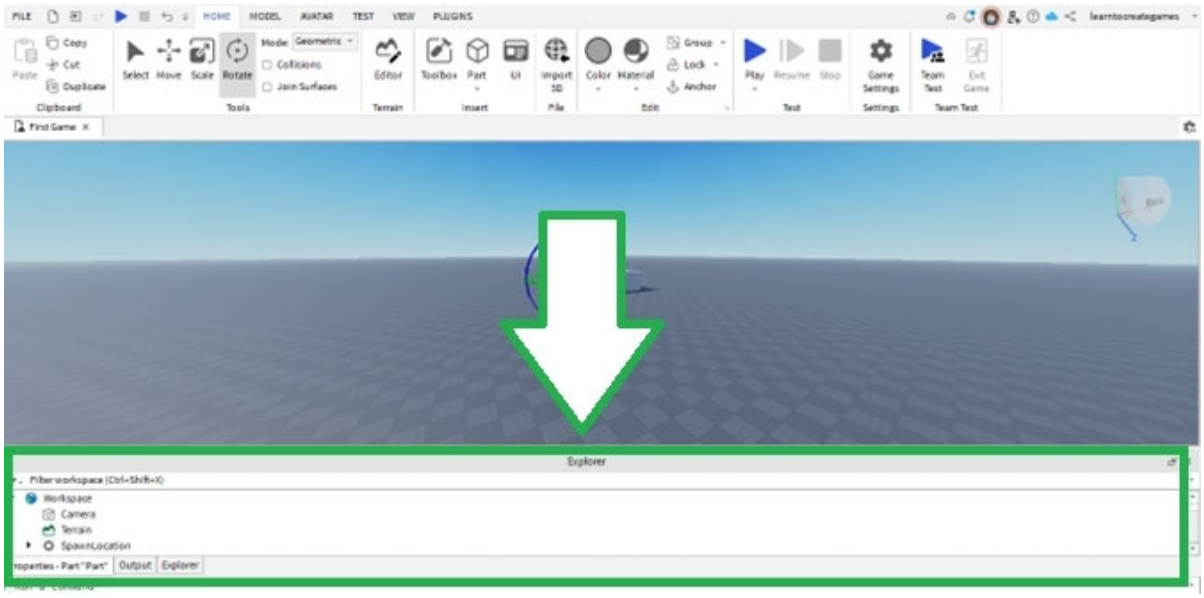
Let’s look at each of these features:

Adjusting Panels: To adjust a panel, you can drag and drop it to rearrange it as needed. For example, you could do the following:

Click and hold the title bar of the
Drag this panel to a new location (for example at the bottom of the screen) and release the mouse button to drop it, as per the next figures.

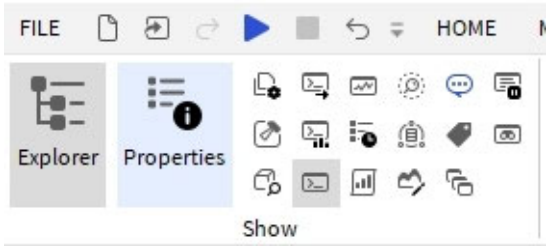

Copy
Cut
Duplicate
Paste
Clipboard
Select
Move
Scale
Rotate
Mode: Geometric
Collisions
Join Surfaces
Tools
Editor
Terrain
Toolbox
Part
UI
Insert
Import 3D
File
Color
Material
Group
Lock
Anchor
Edit
Play
Resume
Stop
Test
Game Settings
Settings
Team Test
Exit Game
Team Test
First Game
Explorer
Filter workspace (Ctrl+Shift+X)
Workspace
Camera
Terrain
SpawnLocation
Baseplate
Part
Players
Lighting
MaterialService
NetworkClient
ClientReplicator
ReplicatedFirst
ReplicatedStorage
ServerScriptService
ServerStorage
StarterGui
StarterPack
StarterPlayer
Teams
SoundService
TextChatService
Run a command


Copy
Cut
Duplicate
Paste
Clipboard
Select
Move
Scale
Rotate
Mode: Geometric
Collisions
Join Surfaces
Tools
Editor
Terrain
Toolbox
Part
UI
Insert
Import 3D
File
Color
Material
Group
Lock
Anchor
Edit
Play
Resume
Stop
Test
Game Settings
Settings
Team Test
Exit Game
Team Test
Empty Docking Area
First Game
Empty Docking Area
Empty Docking Area
Empty Docking Area
Explorer
Filter workspace (Ctrl+Shift+X)
Workspace
Camera
Terrain
SpawnLocation
Baseplate
Part
Players
Lighting
MaterialService
NetworkClient
ClientReplicator
ReplicatedFirst
ReplicatedStorage
ServerScriptService
ServerStorage
StarterGui
StarterPack
StarterPlayer
Teams
SoundService
TextChatService
Run a command

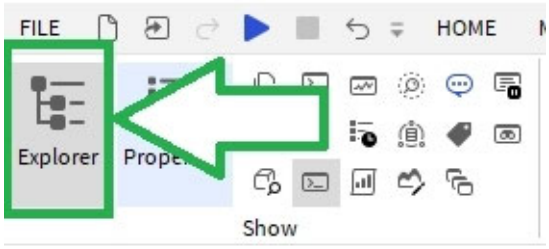


Add or Remove Panels: To add or remove a panel you can select View | Toolbox | Show and then select the panels that you want to display/hide in your workspace. For example:

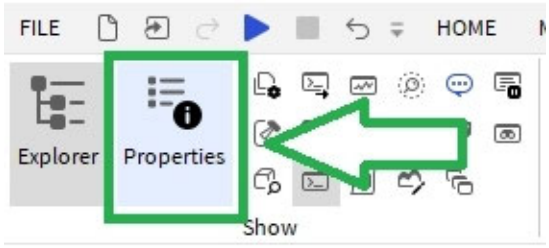
Select View | Toolbox from the top of the screen.
You should then see a section called Show that lists all the panels that can be displayed/hidden in your workspace.



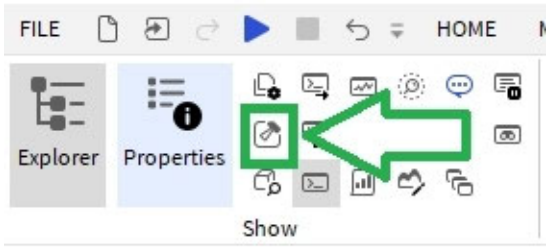
Click on the Explorer panel and check that it displays/hides in the workspace.



Repeat this step for the panels



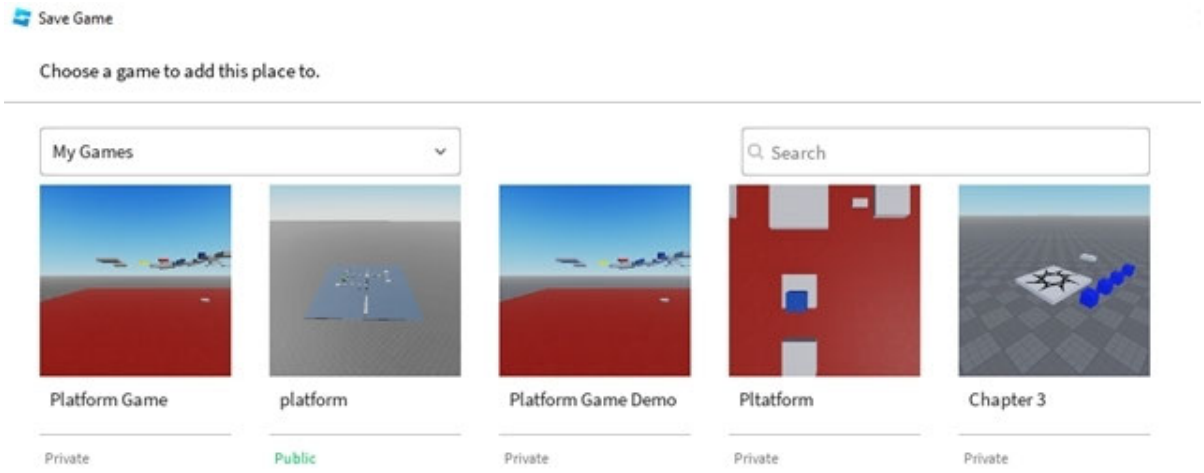
Repeat this step for the panels



After completing these steps, you should see that your workspace is tailored to your personal preferences. A customized workspace enhances productivity by keeping frequently used tools and panels easily accessible.

You can now name and save your project:

Click on File | Save to Roblox As.
In the new window, press Create New

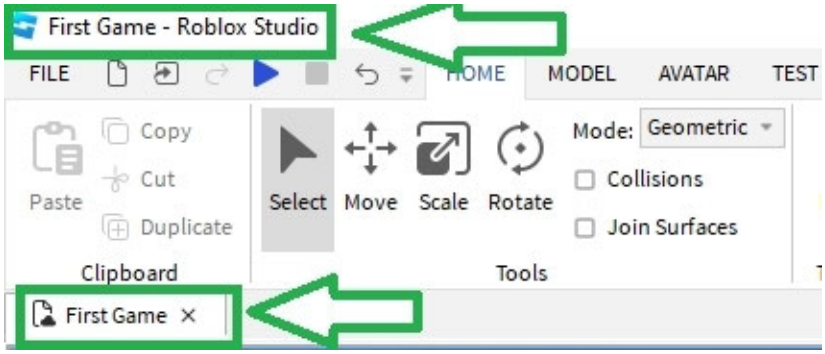


This will open a new window where you can include more information about your project.



We will be able to change this information later, but for now you can set its name to leave the rest of the information as default and press the button.

As the game is saved and Roblox Studio appears, you should notice that the name that you have defined appears both at the top of the window, and also on the tab for your 3D as per the next figure.



OceanofPDF.com

Understanding Basic Components

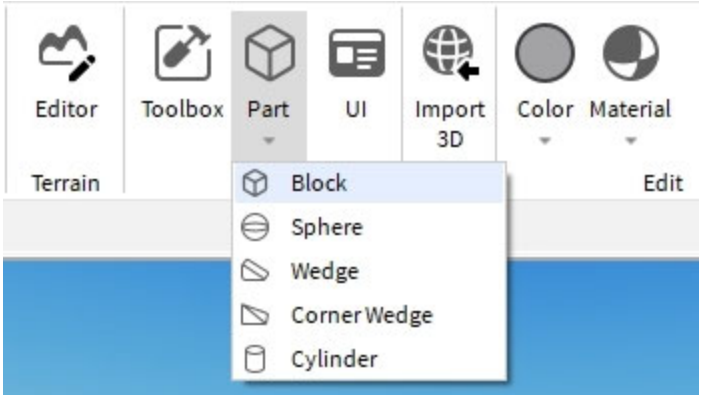
While you are now familiar with the interface of Roblox, we will dive deeper into the basic components of a Roblox project, such as parts, models, and scripts.

Roblox provides what are called Basic which are basic shapes that you can add to your project and combine.

Let's explore the basic components:

Parts: parts are the simplest building blocks of any game, and they can be created very easily.

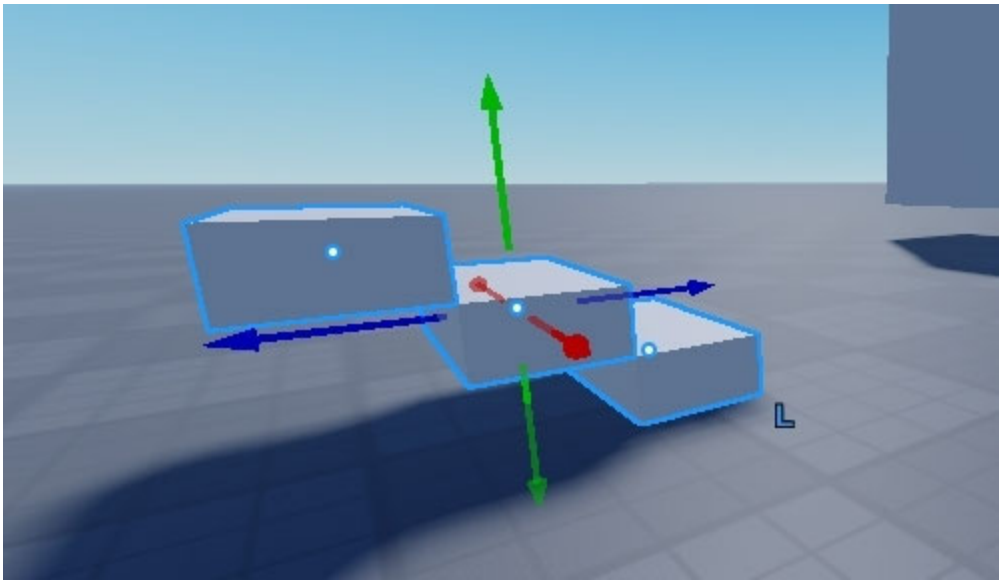
Part | Block from the Home tab.



This will create a new block that you should see both in the viewport and the
Rename this object In the right-click on this object, select the option and type



Please repeat the previous steps to create two more boxes that you will rename box2 and please also move them apart using the Move tool so that they look like a staircase put together, as illustrated in the next figure.



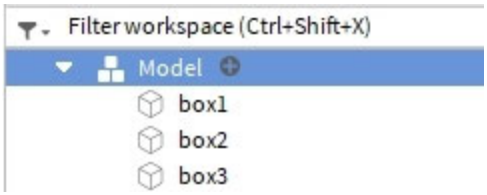
Note that, to make things easier, you can also duplicate the first box by selecting that object and pressing CTRL + D (duplicate) twice.

Models: In Roblox, it is possible to group different parts to form a complete object and this is done through models. To create a model (i.e., a new object that includes individual parts), for example to create an object called that includes all three boxes created earlier, you can do the following:

Select the three boxes by holding the CTRL key and clicking on each box.
Right-click on one of the boxes and select Group as a Model from the contextual menu to create a model, as per the next figure.

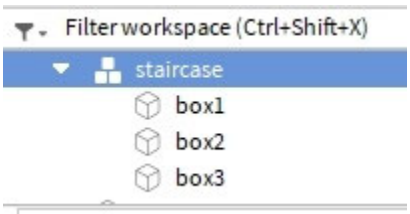


You should then see in the Explorer that a new object called Model has been created, and that it includes the three boxes that we have created earlier.



We can say, in that case that Model is the parent of the objects and or that these 3 boxes are the children of the object

You can rename the object Model as staircase: Select the object press F2 (or right-click and select and type the new name.



Scripts: scripts are pieces of code that add functionality to your game, and they can be added to objects by selecting that object, right-clicking on that object, and selecting the option Insert Object | We will experiment with that feature in the next sections.

After completing the previous steps, you should see that you can add and manipulate basic components in Roblox Studio. These components are fundamental for building and scripting your game.

OceanofPDF.com

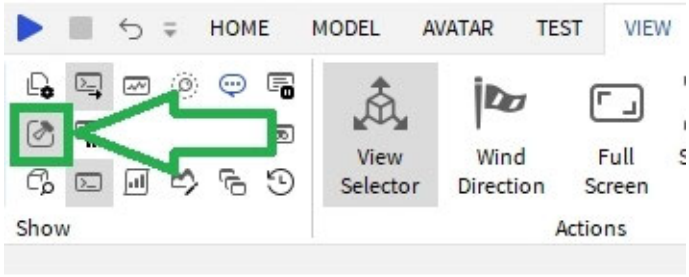
Using the Toolbox

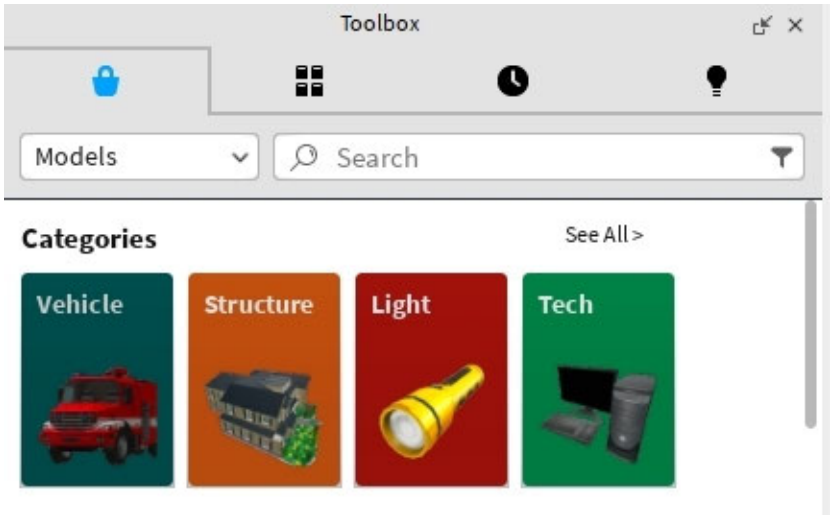
In this section, we will learn how to use the Toolbox to find and utilize pre-made assets in your game.

Let's use the Toolbox:

Open the Toolbox from the left side of the screen.

If the Toolbox is not visible, go to View | Show to enable it.



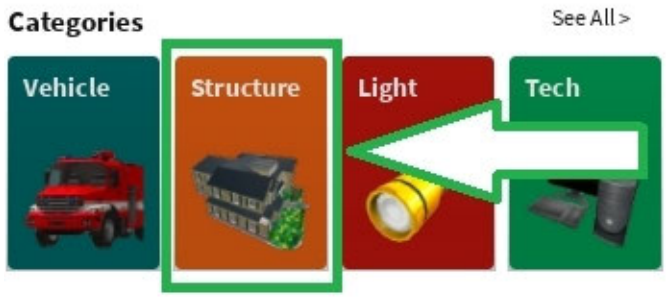


Search for assets using the search bar at the top of the

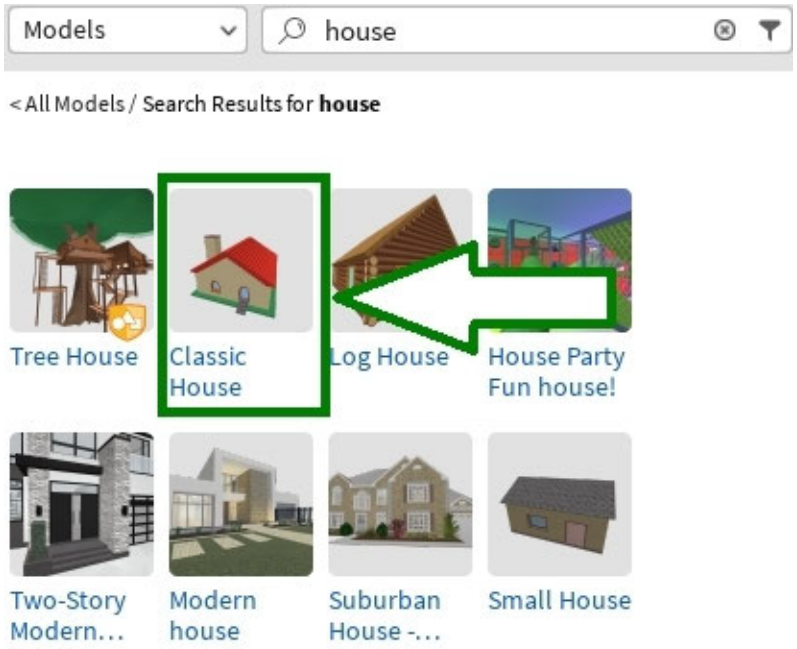
Select Models from the drop-down menu located at the top of that panel.



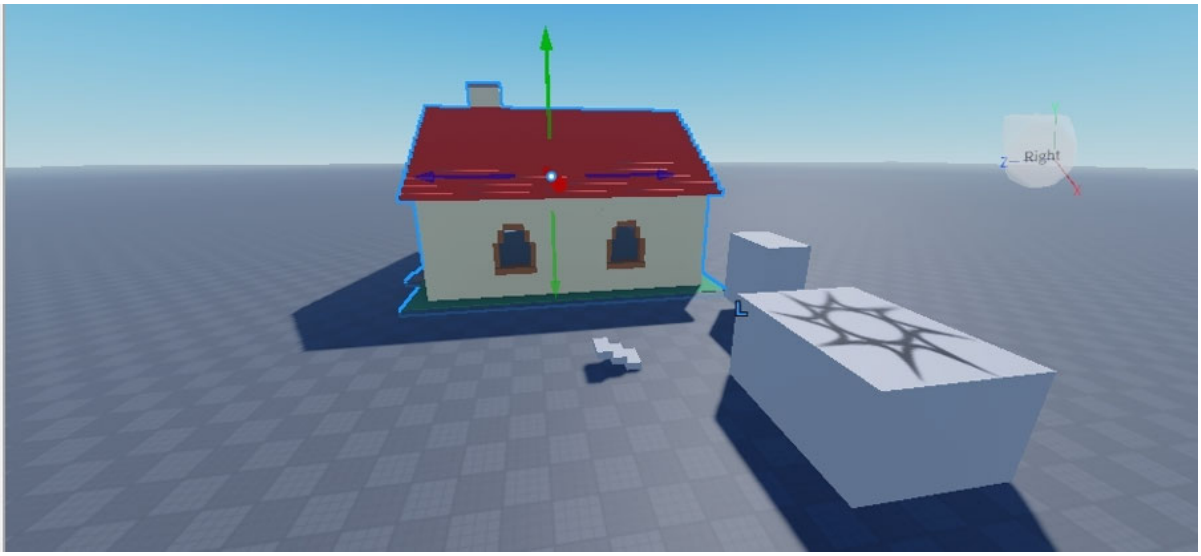
Click on the icon labelled as per the next figure.



Type the keyword house in the search field.
Browse through the results and select the result called Classic House as per the next figure.

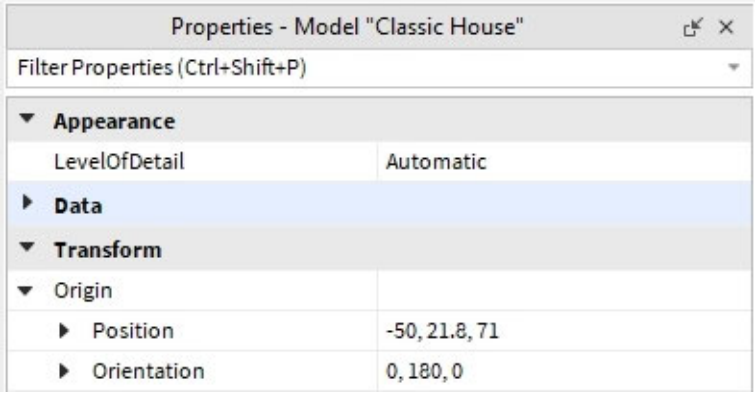


Drag and drop it to your scene (i.e., the 3D You should then see that the house has been added to your scene.



Once this is done, you can look at the Properties window to see some of the attributes of the asset that you have just imported into your game

You can adjust some of these properties, either using the transformation tools (e.g., etc.) or the Properties window.



OceanofPDF.com

Exploring Your New Scene

Now that you are comfortable with transforming objects and adding assets, it is time to play our scene and see how it will look when played by you and other players.

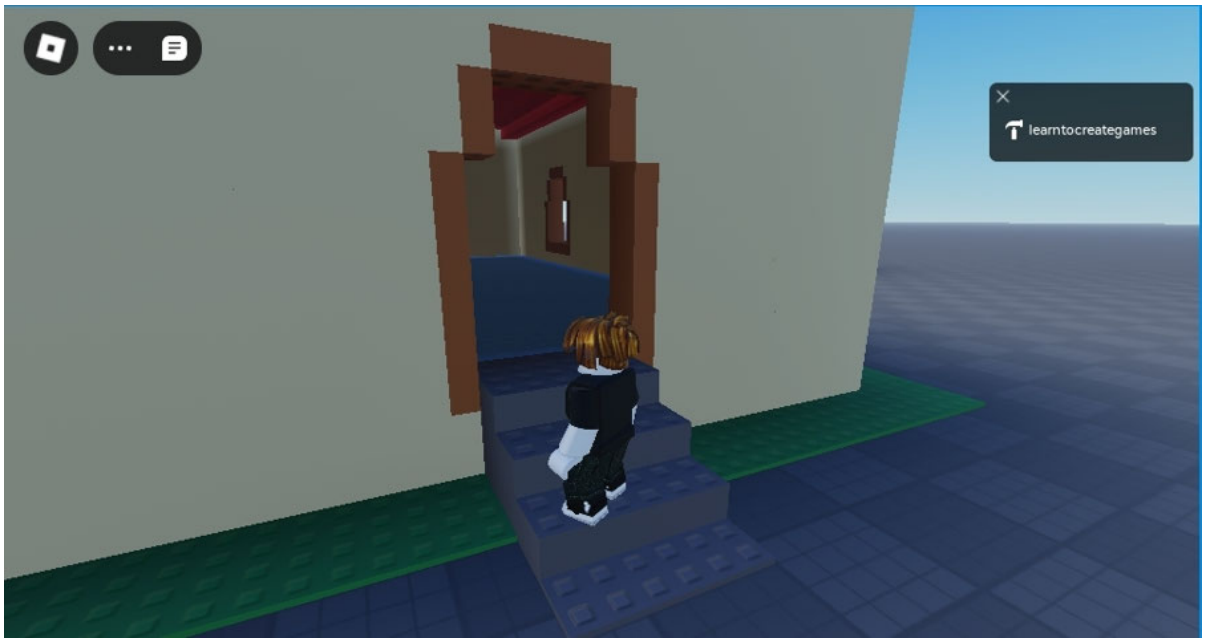
Please press the Play button located at the top of the window (blue triangle).



You should see, as per the next figure, your main character and all the elements that you have added, including: a scaled box, a staircase, and a house.



You can then use the arrow keys on your keyboard, along with the space bar to navigate this scene.



You can also enter the house and explore inside.

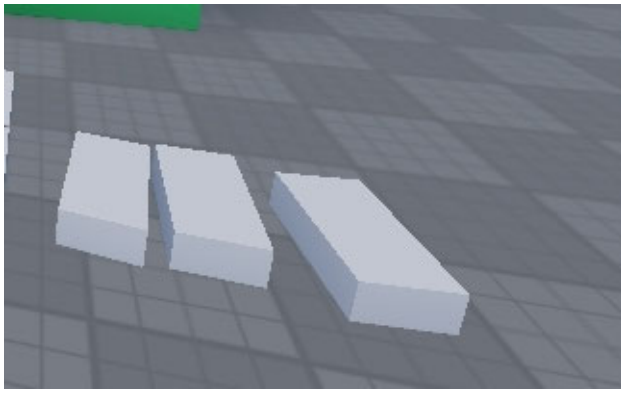


When you want to stop playing the scene, you can press the red square located at the top of the screen or press SHIFT +



One last thing, you may notice that the staircase doesn’t appear correctly, as per the next figure.





This is because all the boxes that we have created, because they are subject to gravity, have fallen to the ground; to fix this, we can specify that these boxes should not be subject to gravity, and to anchor them; so please do the following:

Using the select each box.



Using the Properties window, tick the option Part |



As you play the game, you should see that the staircase appears properly this time.



Note that the Properties and Toolbox tabs may be displayed under the same window, and you will need to click the relevant tab to display the corresponding information.

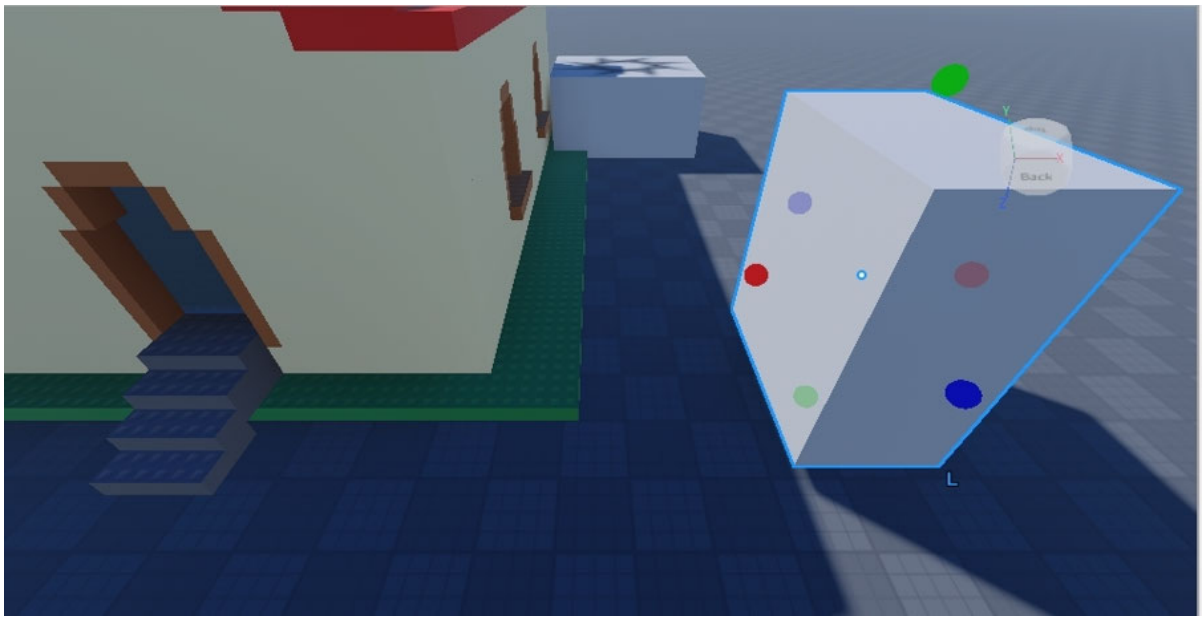


OceanofPDF.com

Basic Scripting with Lua

In this section, we will introduce basic scripting with Lua, the programming language used in Roblox Studio and write your first script. With this new script attached to an object, we will ensure that every time the player clicks on that object, that its color changes to a random color.

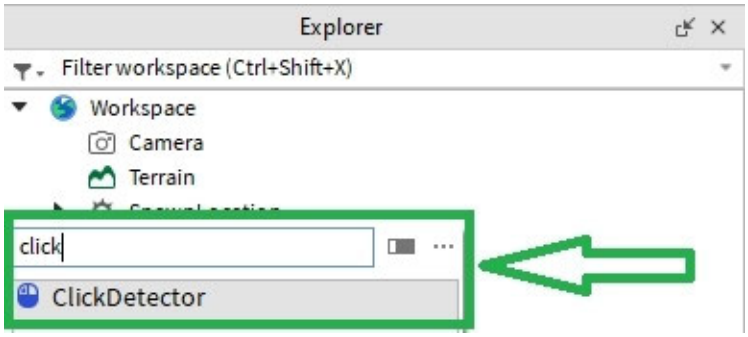
Please add a new box to the scene: Select Part |
Rename this new object
Resize this box along the y axis so that it is easier to click.



Add a click detector to this object: using the Explorer window, right-click on the object then select Insert



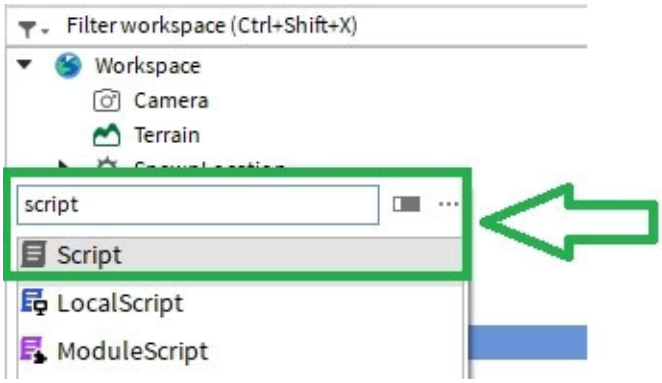
In the new contextual window, type the text in the search field, then select the result as per the next figure.



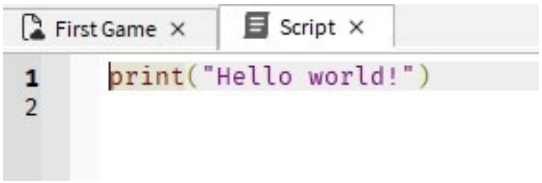
This will add a new component called to your object and ensure that any click on that object is detected and processed.

Now that we have added the ability for this box to detect clicks, we just need to create a script that will handle the clicks by changing the color of the box whenever this happens.

Using the Explorer window right-click on the object then select Insert Object
Type the word in the contextual search field.
Select the result called “Script” as per the next figure.



This will create a new script that is attached to the object, and automatically open in the script editor.



This new script that we are about to write will ensure that we change the color of the part whenever it is clicked.

Note that you can reopen the script later by double-clicking on it in the Explorer panel.

Now that the script has been created, we will modify it so that it produces the behavior that we need (change of color upon clicking).

Replace the default script with the following code:

```
local part = script.Parent

local clickDetector = part.ClickDetector

local function onClick()

= BrickColor.Random()

end

clickDetector.MouseClick:Connect(onClick)
```

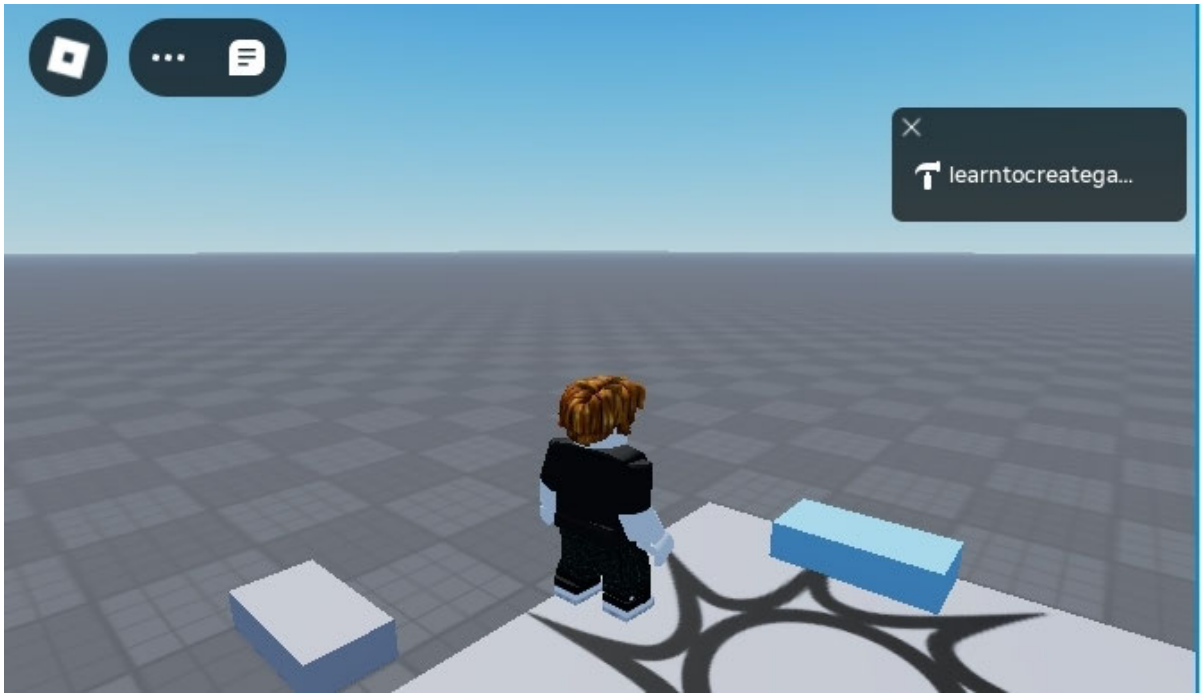
In the previous code:

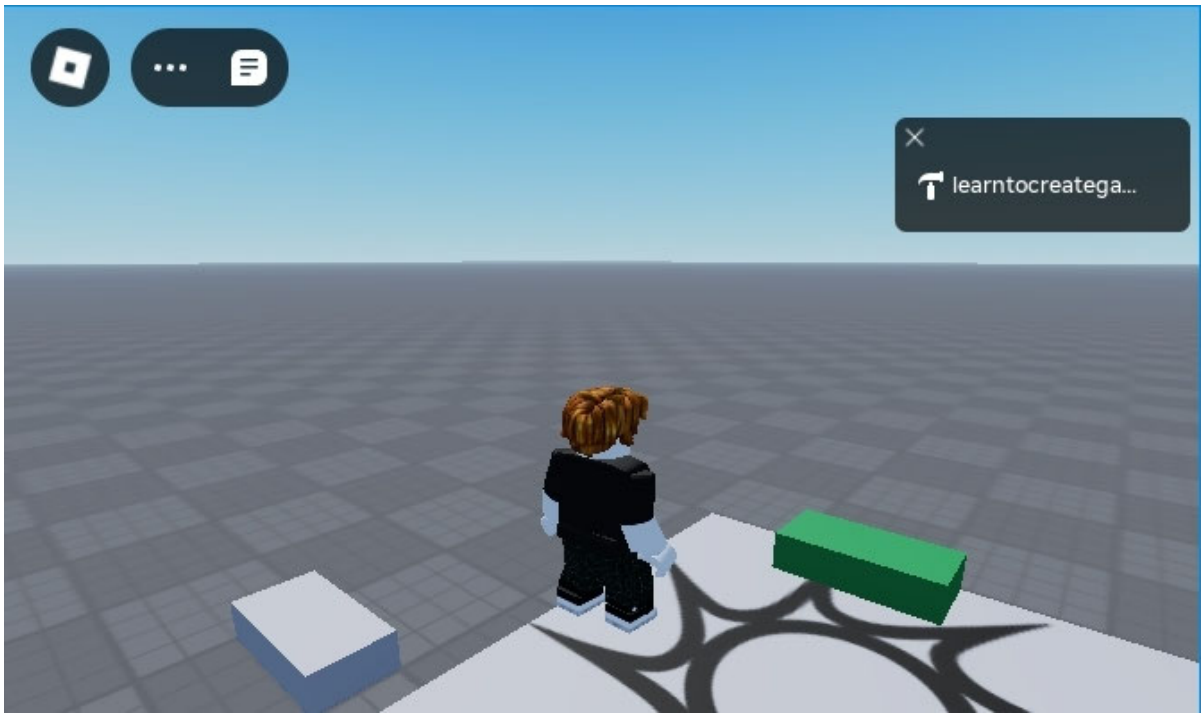
We declare a variable called part that is the parent of (or the object attached to) the script.
We then create a function called onClick that, when called, will change the color of the box to a random color.
Finally, we specify that every time the box is touched (i.e., clicked), we call the function onClick

Now that the script has been created, we can save it and play the game to see how it works.

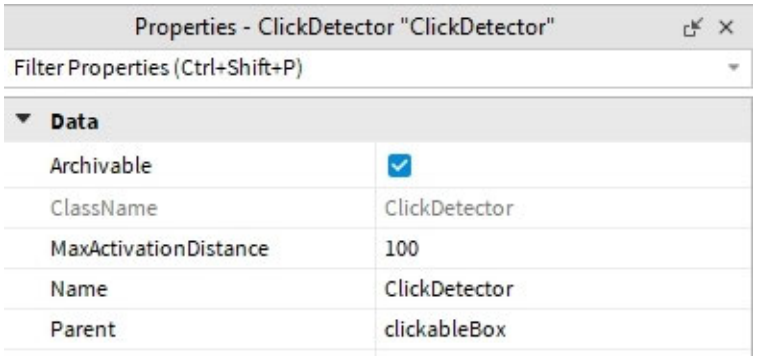
Save the script and test it (CTRL +

Click Play at the top of the screen to start the game.
Click the part in the game to see the color change.





Note that if the clicks don’t work, you can increase the attribute called MaxActivationDistance for the Click Detector linked to the box, as per the next figure:



As you've seen in the last few steps, basic scripting with Lua allows you to add interactivity and functionality to your game objects. With just a few lines of code, you've managed to bring your scene to life. As we continue through the book, Lua will play an essential role in enhancing your

game’s interactivity and mechanics. We’ll explore more advanced Lua features in the upcoming chapters to help you further develop your game.

OceanofPDF.com

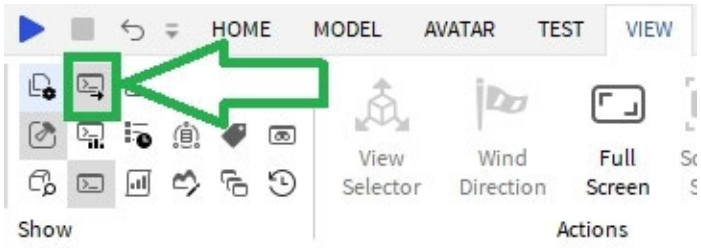
Testing and Debugging Your Script

In this section, we will learn how to test and debug your game to ensure it runs correctly and fix any issues that arise.

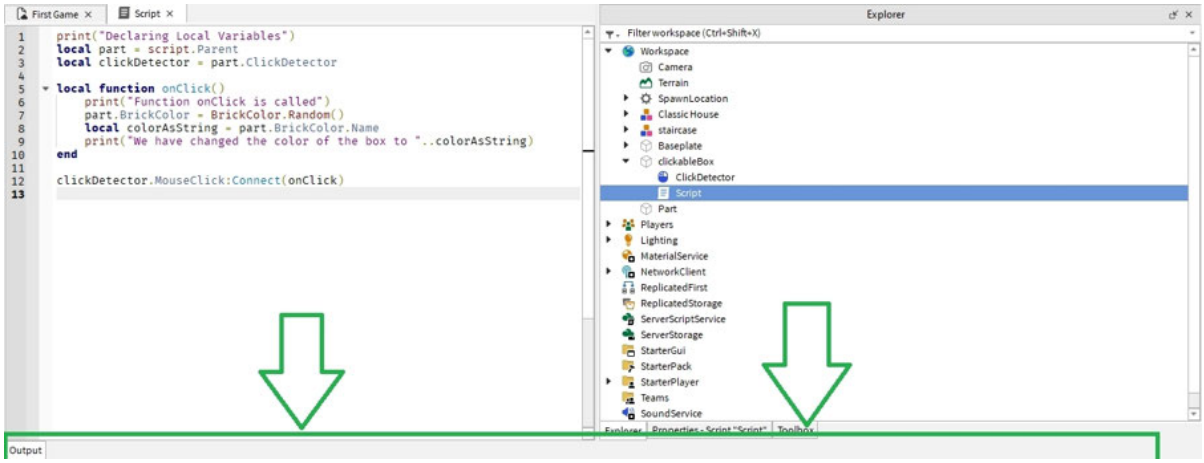
Whenever you write your code, it is often useful to write messages that will be displayed in the Output window, and that will indicate what part of the code has been executed; this way, if there are any issues with the code, you are better equipped to fix it.

The Output window is where messages can be displayed from the code; you can activate this window (if it is not active yet) by selecting it in the View tab | It usually appears at the bottom of the screen by default (after clicking on the Output tab.

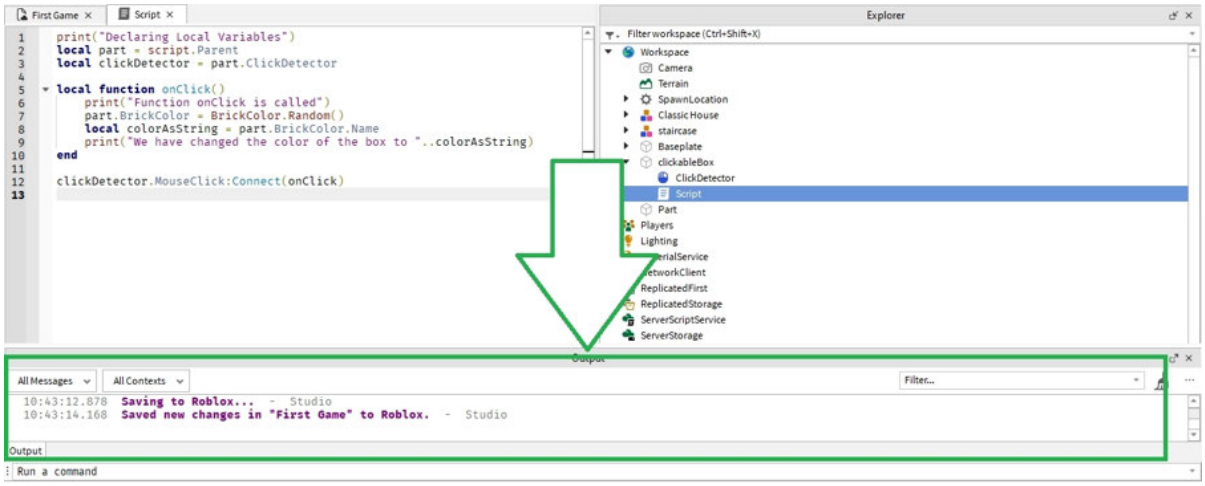
For this you will need to activate the Output window from the View panel as per the next figure.



You should then see the Output tab at the bottom of the screen



After clicking on that tab, you should see the messages, as per the next figure.



Now that that its clearer as to how to display the Output window, let's modify our code:

Please open the script that you have just created.
Modify the code as follow (new code in bold).

---

print("Declaring Local Variables")

local part = script.Parent

local clickDetector = part.ClickDetector

local function onClick()

onClick is called")

= BrickColor.Random()

colorAsString = part.BrickColor.Name

have changed the color of the box to "..colorAsString)

end

clickDetector.MouseClick:Connect(onClick)

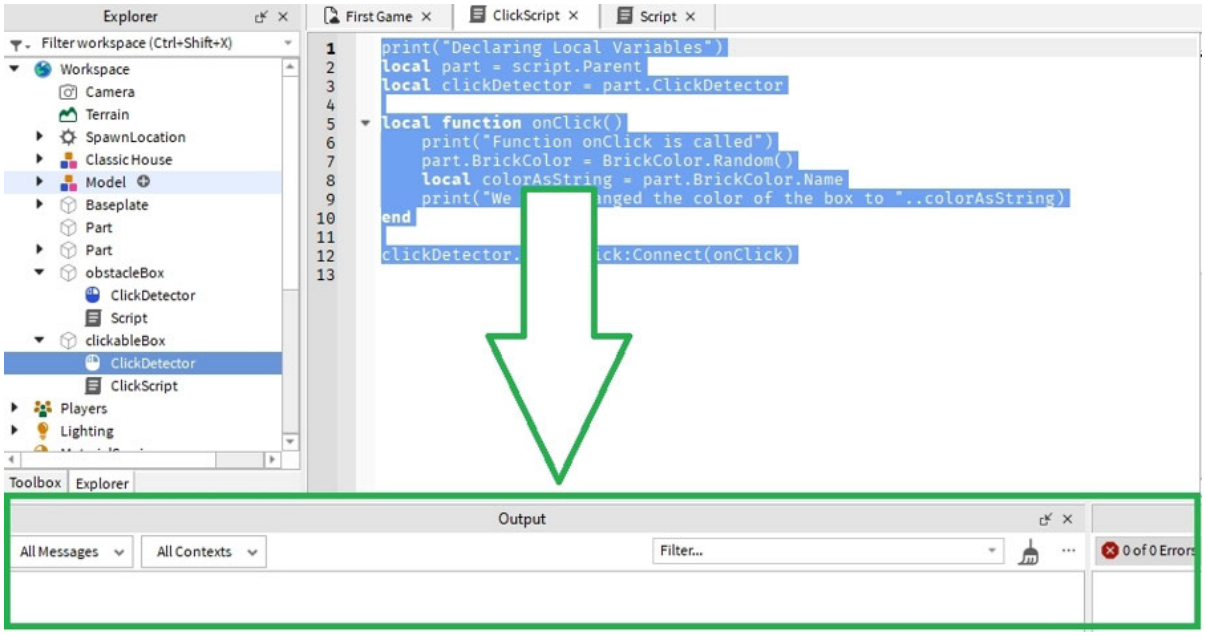
In the previous code:

We write statements that will be displayed in the Output window.
The first statement tells us that we have declared variables
The second print statement tells us that we have called the function onClick
We then declare a local variable that stores the color of the object, as a string
We then display a message that specifies the color of the object

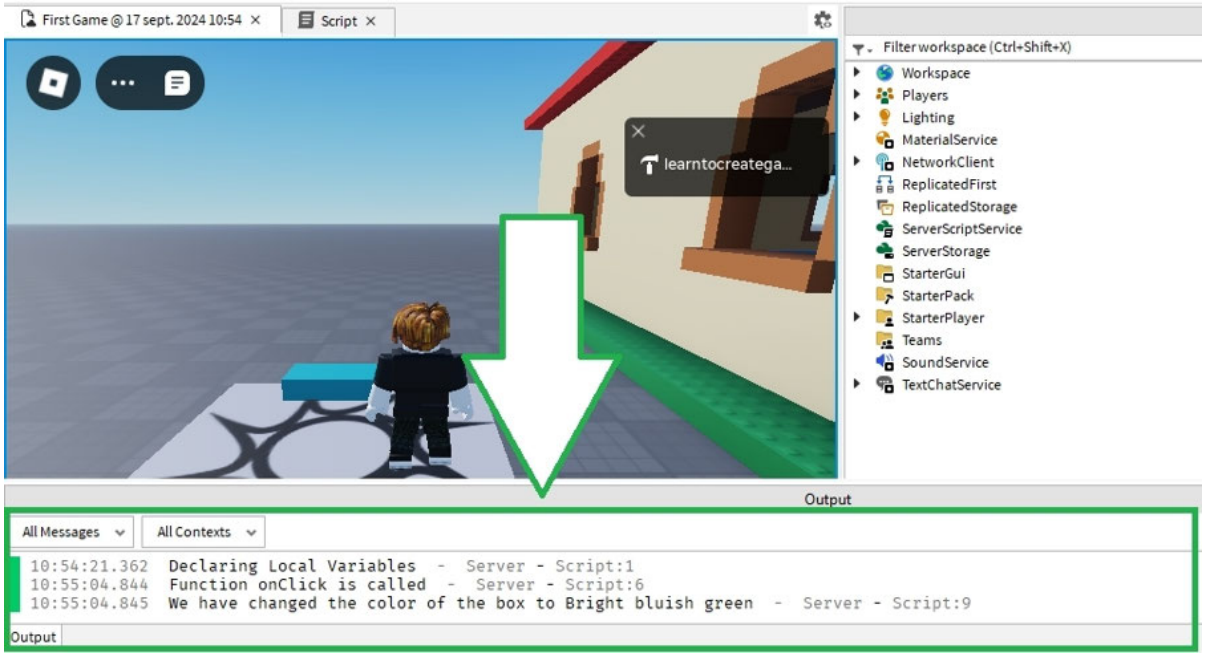
Now that you have changed your code, you can save it + and ensure that the Output window is active through the View Panel.



If you have decided to display your Output window at the bottom of the screen, it should look like the following figure:



You can now play your scene, and click on the box, and you should see the following messages in the Output window:



After completing these steps, you can see that it is possible to include very simple messages in your code that will help you, through the Output window, to debug your code, if and when needed.

OceanofPDF.com

Saving and Loading Projects

In this section, we will cover the proper way to save your work and how to load projects in Roblox Studio.

Let's save and load your projects:

Save your project regularly to prevent data loss.

Click on File | Save To Roblox or File | Save To Roblox
If you choose the second option to Roblox you will need to select a name for your project.

You can also load an existing project by opening it from Roblox Studio.

Click on File |
Browse to the location of your saved project and click

OceanofPDF.com

Overview of Roblox Community and Resources

Roblox has a vast and supportive community of developers, along with a wealth of resources to help you learn and grow as a developer. Whether you're looking for tutorials, forums, or developer tools, the Roblox community has something to offer.

In this section, we will explore some of the key resources available to you, including the Roblox Developer Hub, community forums, and official documentation. These resources will be invaluable as you continue your journey in Roblox game development.

Please do the following:

Explore the Roblox Developer Hub: Visit the Roblox Developer Hub at This site offers a wide range of tutorials, API references, and tools to help you develop your games.
Join the Community The Roblox Developer Forum is a great place to connect with other developers, ask questions, and share your knowledge. You can access the forums at devforum.roblox.com.
Use the Official The Roblox API Reference is an essential tool for understanding how to use different features and functions in Roblox Studio. You can find it at

By taking advantage of these resources, you can continue to learn and grow as a Roblox developer. Remember, the community is a powerful tool for gaining knowledge and support as you develop your games

OceanofPDF.com

Summary and Review

In this chapter, we covered the installation and setup of Roblox Studio, explored the main interface components, learned basic navigation and tool usage, customized the workspace, created and saved your first project, understood basic components, performed simple object manipulations, used the Toolbox for pre-made assets, wrote and tested your first Lua script, created a simple game, and debugged it. That's a long way from where we started, so congratulations on completing this first chapter! In the next chapter, we'll dive into creating your first game objects and understanding their properties.

OceanofPDF.com

Quiz

Please guess whether the following statements are True or False (the solution is in the section Answers to at the end of the book).

Can Roblox Studio be installed on both Windows and macOS?
Is the Menu Bar located at the top of the Roblox Studio interface?
Can you find pre-made models in the Toolbox?
Is the Explorer panel used to manage game objects?
Does the Properties panel allow you to change object properties?
Can the Select tool be used to move objects?
Are the arrow keys used to navigate the workspace?
Can you customize the layout of your workspace?
Is it possible to save your custom workspace layout?
Does the Move tool resize objects?

OceanofPDF.com

3
Building Basics

In this chapter, we will focus on the core skills of building in Roblox Studio. This includes creating and manipulating parts, understanding part properties, grouping objects, and more. By the end of this chapter, you'll be able to create basic structures and maps for your games.

So, after completing this chapter, you will be able to:

Create and manipulate parts in Roblox Studio.
Understand and modify part properties.
Group and ungroup parts for better organization.
Use anchoring and collision properties effectively.
Apply materials and textures to parts.
Create simple structures and maps.

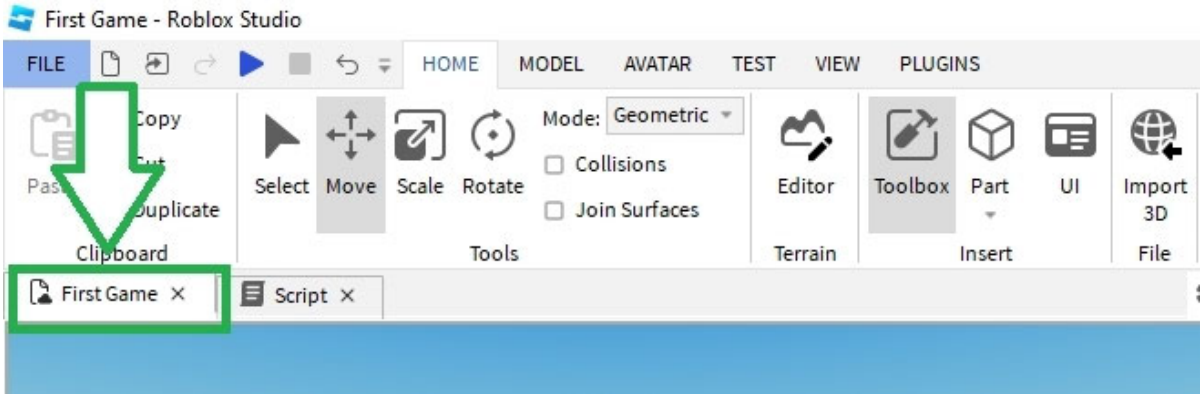
OceanofPDF.com

Understanding Part Properties

In this section, we will explore the different properties of parts that you can modify to customize them. Part properties allow you to change the appearance and behavior of your parts, such as their colour, material, and size.

In the next steps, we will select a part and explore its properties in the Properties panel. We will then modify some of these properties to see how they affect the part.

Click on the tab for the 3D Viewport (the tab called First so that you can see the scene, as per te next figure.

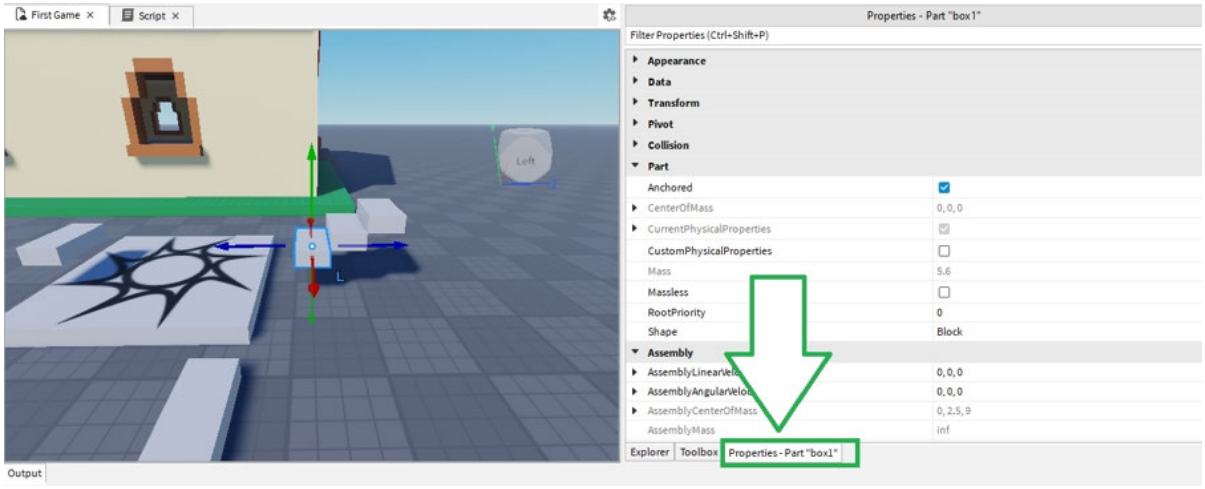


In the Explorer window, locate the model called staircase and the children objects within.



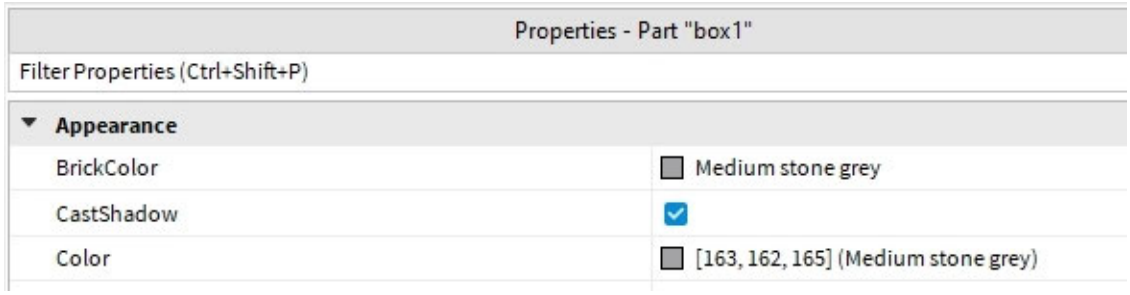
Click on box1 to select it.

Open the Properties panel by clicking on View | Properties if it's not already open or by clicking on the Properties tab, as per the next figure.

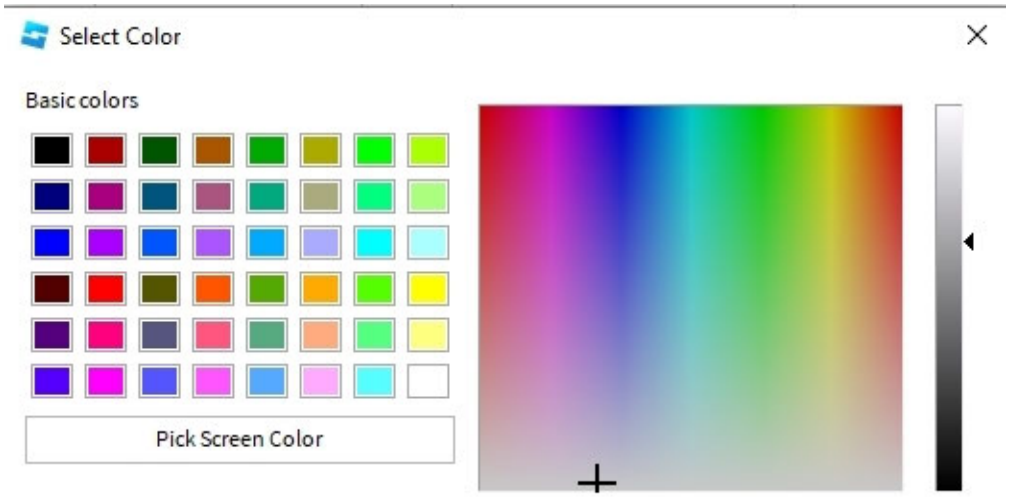


We can now modify the properties of the box, including its color or its texture; let's start with the color:

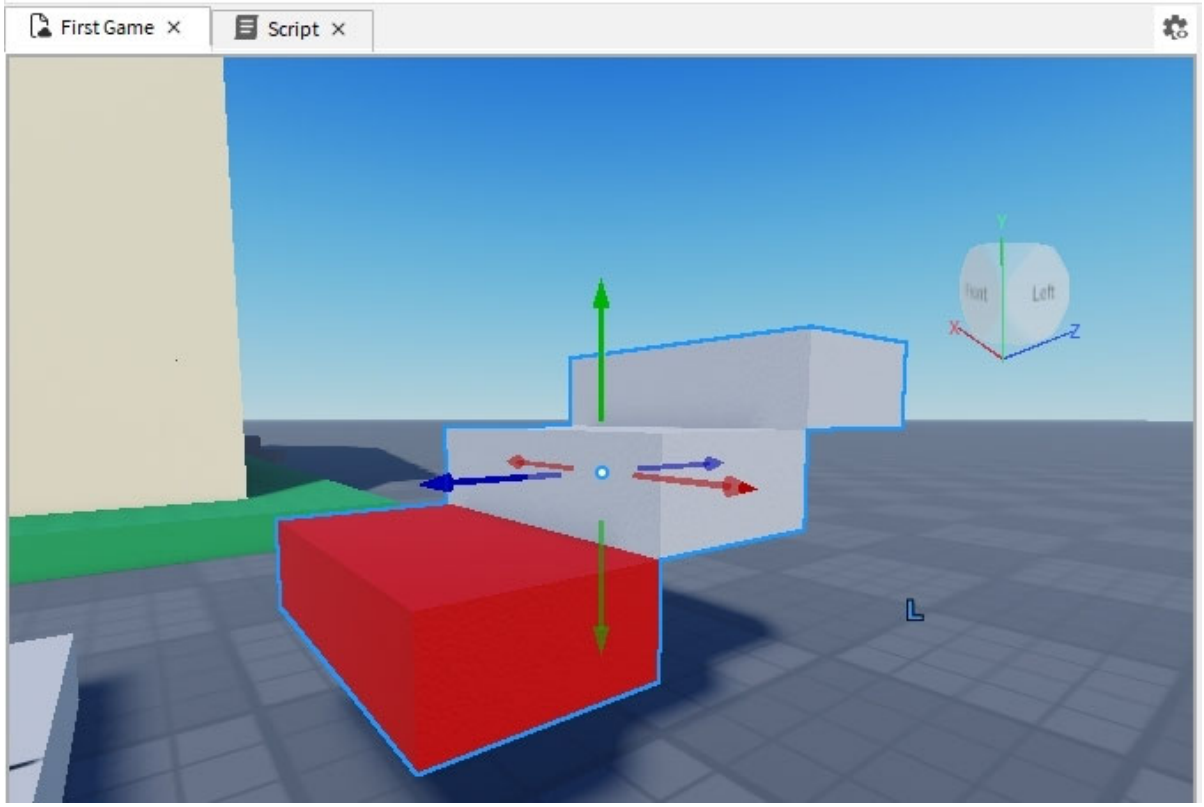
In the Properties panel, find the Color property (Appearance |



Click on the color box next to the Color property to open the color picker.

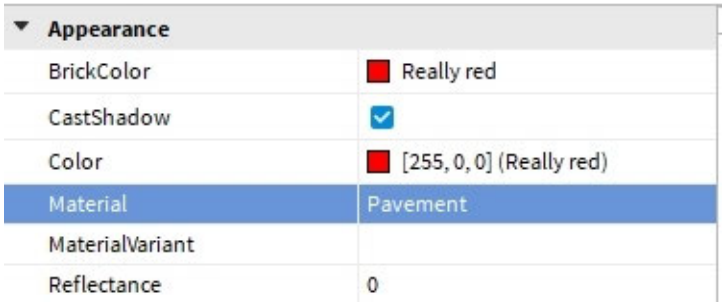


Choose a new color for your part (for example After selecting the color, the part in the workspace should update to reflect this change.

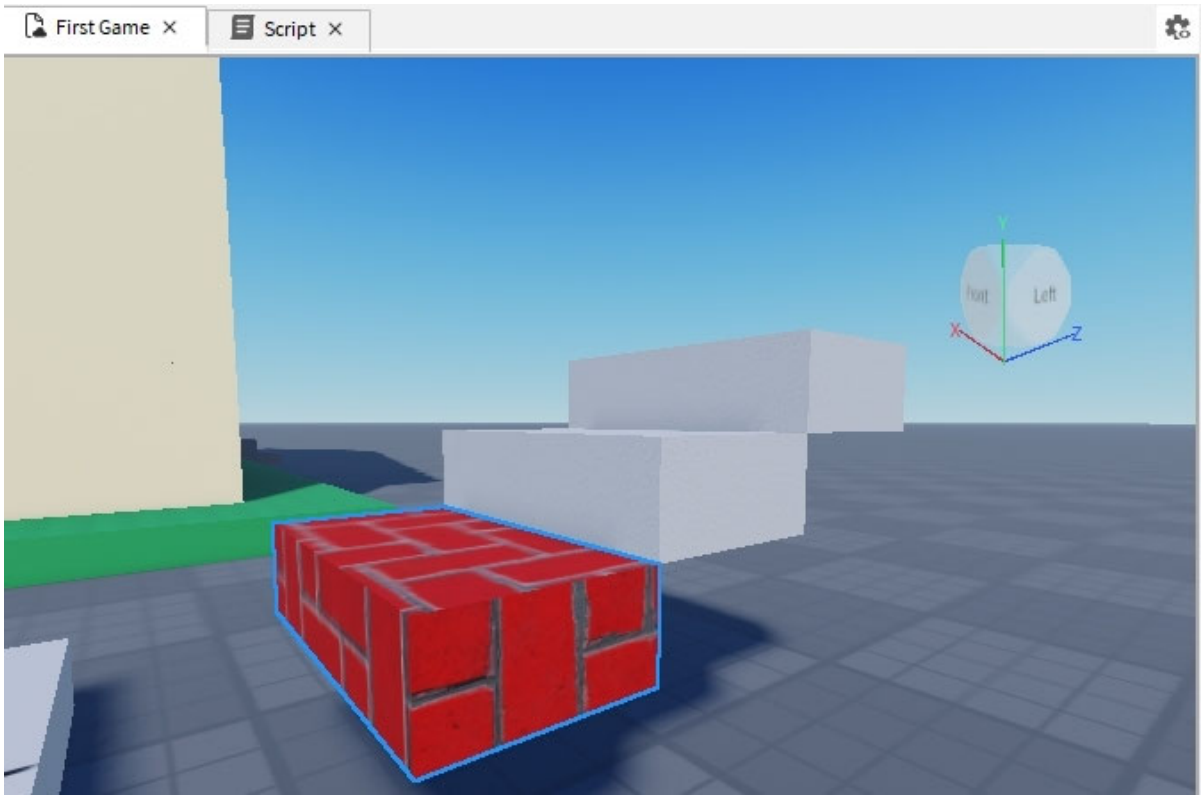


Now that we have modified the color, let’s modify the texture of this box:

In the Properties panel, locate the Material property (Appearance |
Click the dropdown menu next to Material and select a different material (e.g.,

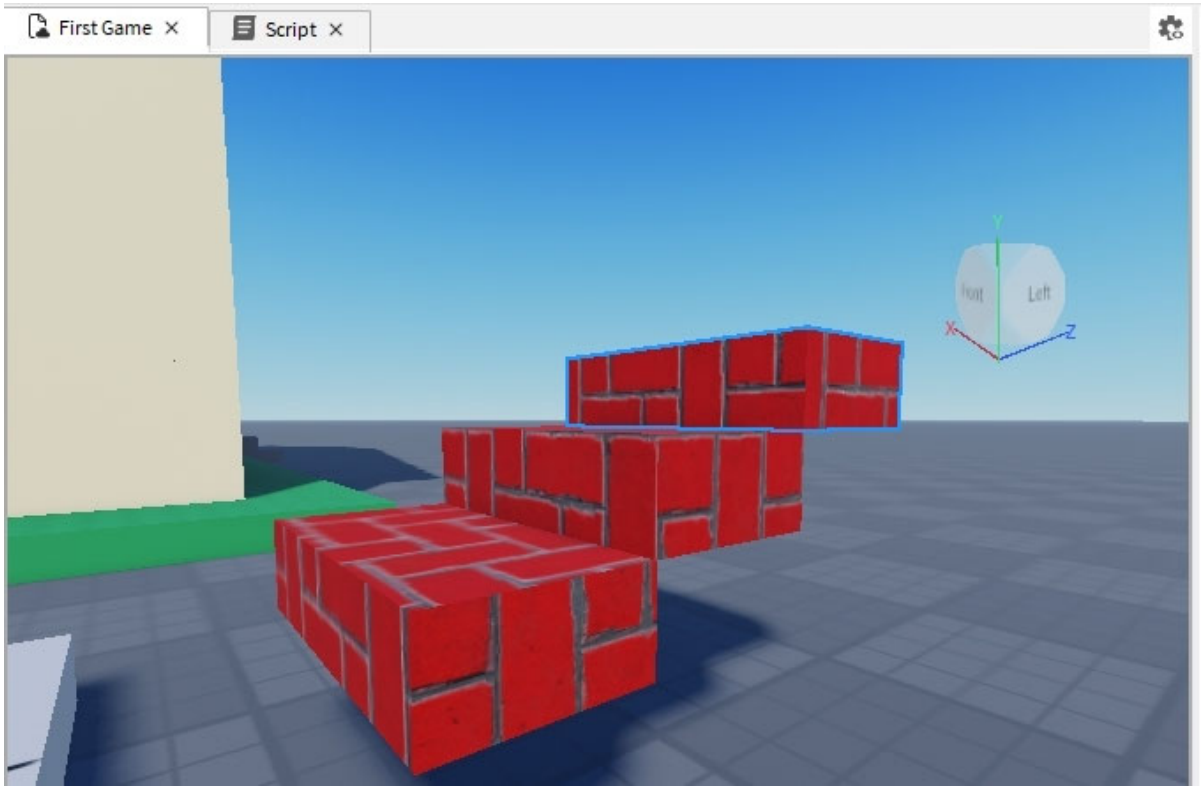


The part's appearance should change to match the selected material, as per the next figure.



As you can see, it’s very easy to modify the appearance of a part using the Properties window; now that you know how it’s done, please apply the same changes to the two other steps named box2 and

After applying these changes, your staircase, should look like the following figure.

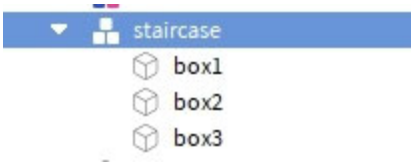


OceanofPDF.com

Grouping and Ungrouping Parts

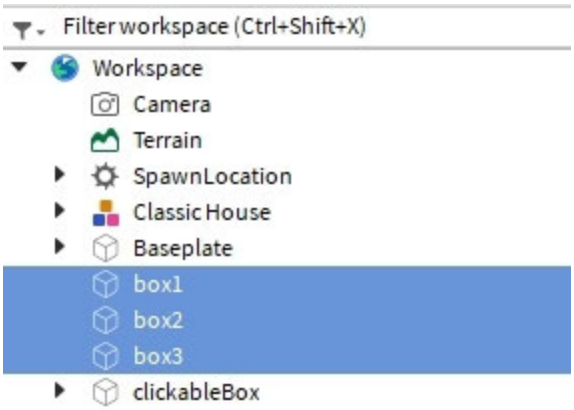
Grouping parts together allows you to manage multiple parts as a single unit, which is particularly useful when building complex structures.

As we have seen in the previous sections, the objects box1, box2, and box3 are grouped and they are the children of a model called staircase. This means, amongst other things that any transformation applied to the object staircase will also be applied to its children: and



This being said, and as you may have guessed, while you can group parts, you can also ungroup them, in a similar way by, right-clicking on the group in the Explorer panel and choose Alternatively, you can press Ctrl + U to ungroup them. After doing so, the parts will return to their individual state.

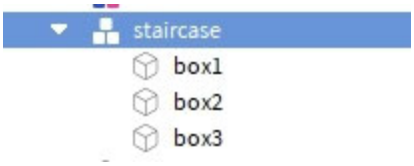
Please click right-click on the object and select the option you should see that the boxes are no longer listed under the object and that the object staircase has disappeared.



Please regroup these objects again: select all three boxes, right-click on one of them and select as

This will create a new model that you can rename

You should now see that the boxes are again the children of the object



Grouping and ungrouping parts is a powerful feature that allows you to manage your game elements more effectively. Next, we will explore anchoring and collision properties to ensure that our parts behave as expected in the game environment.

OceanofPDF.com

Using the Anchor and Collision Properties

Anchoring and collision properties determine how parts interact with the environment and with each other. Anchoring prevents parts from moving, while collision properties determine whether parts will collide with other objects.

Anchoring an object in Roblox makes sure it stays exactly where you put it (as we have seen in the previous sections), without moving or falling when the game starts. This is really important for keeping your buildings and other creations stable and looking the way you want them to.

Anchoring also helps your game run more smoothly because it reduces the amount of work the game has to do to keep track of moving objects. Overall, it helps ensure that everything in your game behaves as expected, making your game more reliable and fun to play.

Collision properties in Roblox determine whether objects can pass through each other or if they will bump into and block each other. If an object’s collision is turned on, it will act like a solid object, preventing players or other objects from moving through it. If collision is turned off, the object will let things pass through as if it isn’t there. Adjusting collision properties is important for controlling how players interact with different parts of your game, like making sure they can walk on a floor or pass through a doorway without getting stuck

You may remember, from a previous section, that we needed to use the Anchored attribute to make sure that our staircase appeared just the way we wanted to, and you can check this by looking at the properties of each box, you will see that the option Anchored is set to true.



You may have noticed in the previous figure that I typed in the search field at the top of the Properties window. This search field allows you to display only properties that include a specific word in their name. In this case, I typed 'ancho' because I was looking for the Anchored attribute. Using this search feature makes it much quicker and more efficient to find the specific properties you're looking for.

You can also play the scene, to check that the staircase looks like it should during the game.

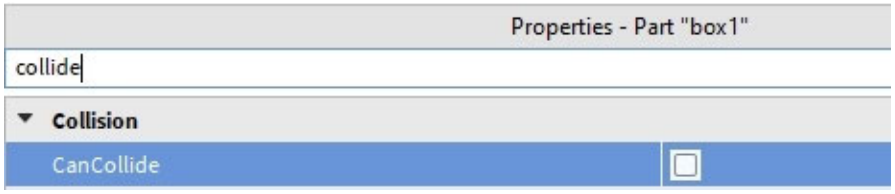


As you move the character, you should be able to climb the stairs because each box has the CanCollide attribute set to allowing collisions so your character can stand on them. If we were to set this attribute to False (it’s True by default), your character would walk right through the stairs.

Let’s verify that

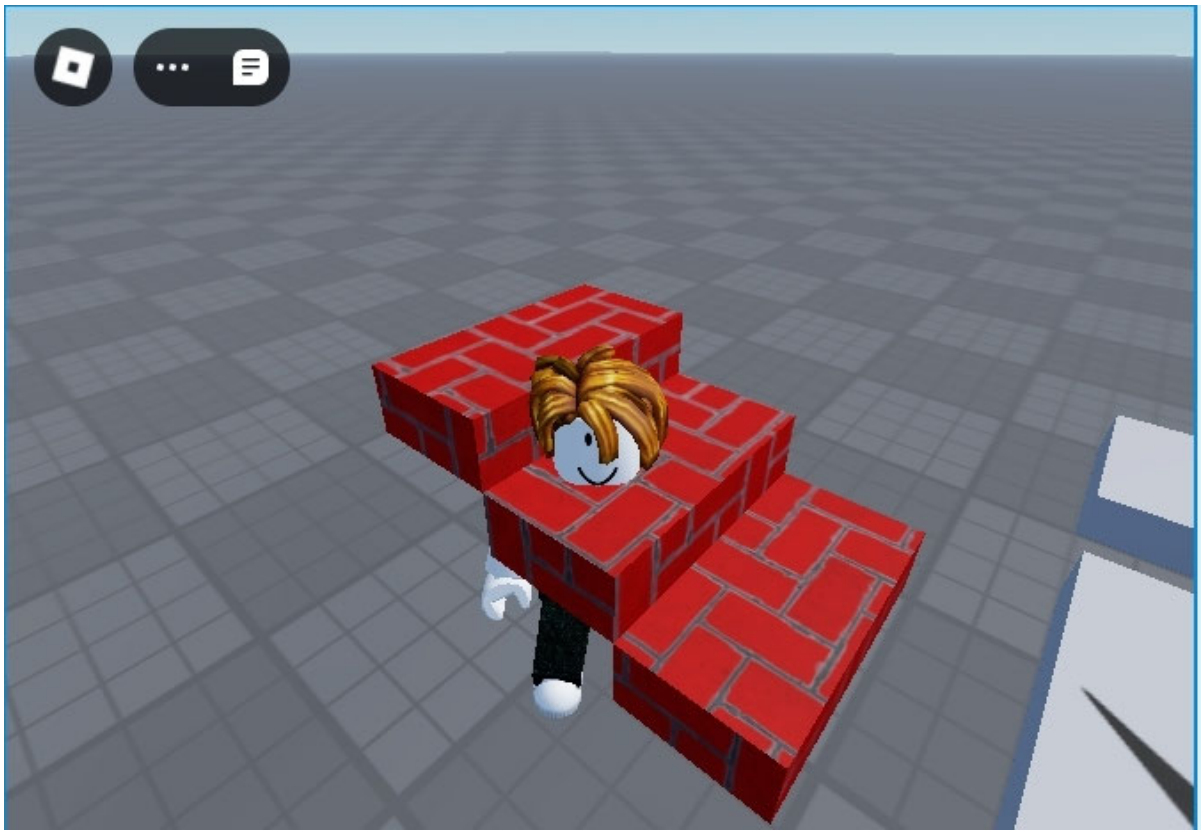
Stop the game.

Select each box and, using the Property window, set the attribute CanCollide to False (i.e., uncheck the box).

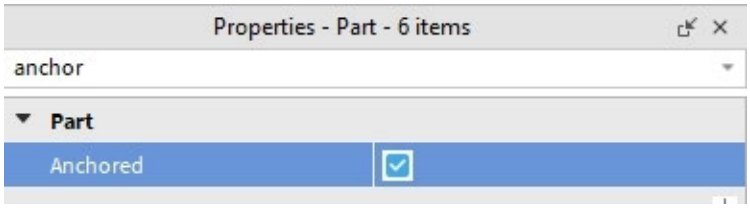


To do this faster, you can also select all three boxes in the Explorer and then display the Properties window; by unticking the CanCollide box, this attribute will be set to False for all three boxes.

Play the game and try to climb the stairs.
You can test the collision behavior by playing the game and observing how the part interacts with other objects. You should see that this time, the player character can literally walk through the steps.



You can now set the collision property back to true for all the stairs.



OceanofPDF.com

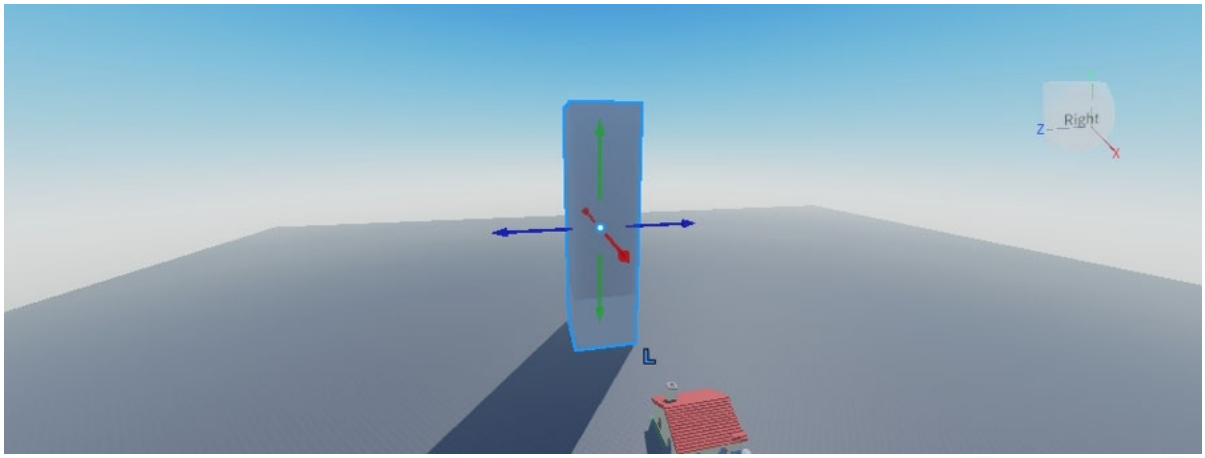
Creating Simple Structures and Maps

While we have learned to modify the properties of parts in the previous section, in this section, we will combine the skills we've learned so far to create basic structures and maps. This is an essential part of game development in Roblox, as it forms the environment where players will interact.

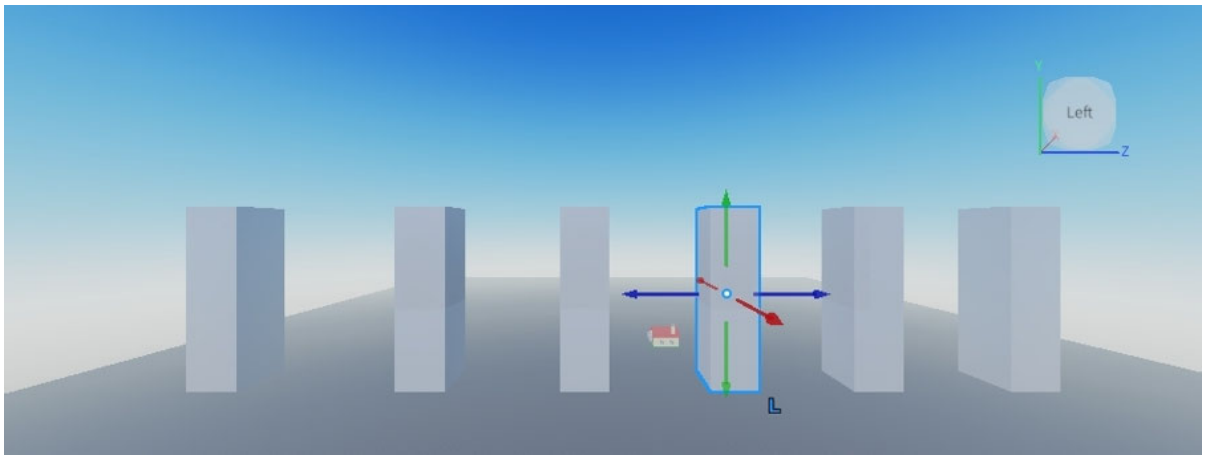
In the next steps, we will create a simple building using the parts we've manipulated, and then we will arrange these buildings to form a basic map layout.

Let's start by creating a building:

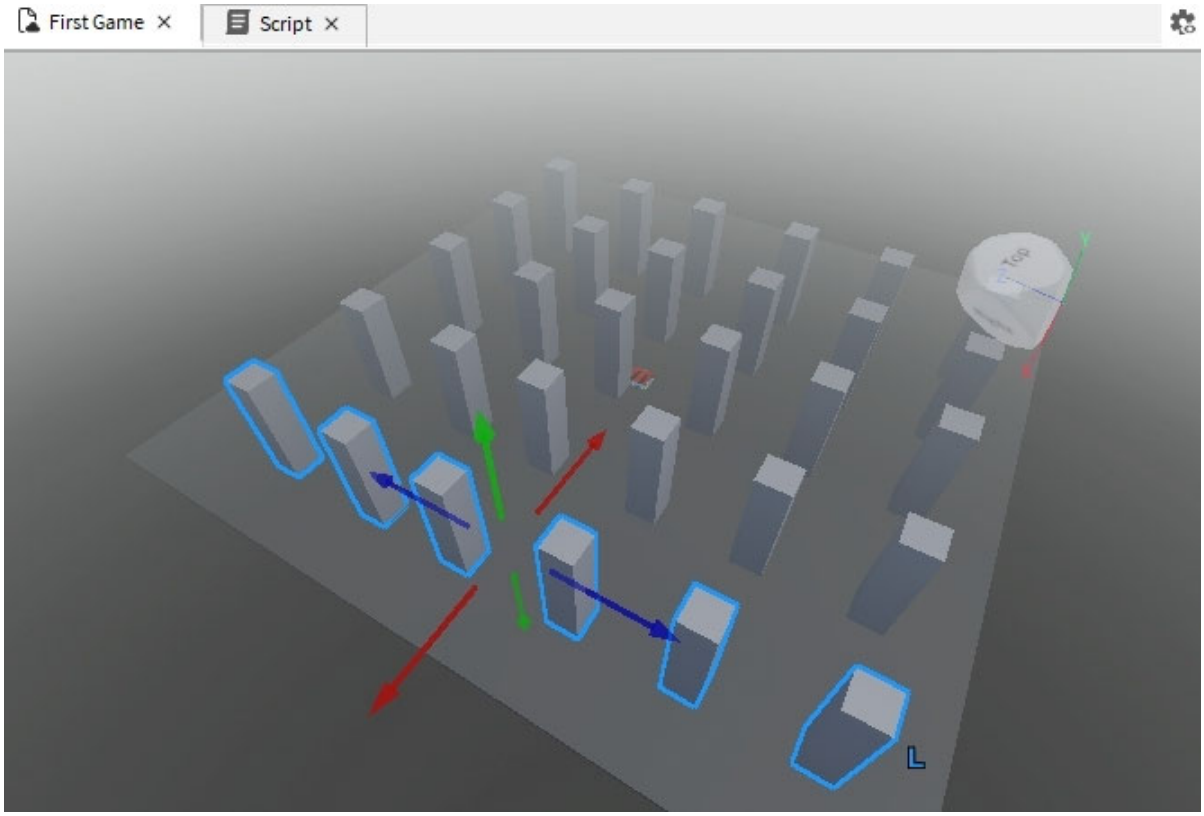
Create a new Box Part | and rename it
Scale this box on all three axes so that it looks like a building, for example: 300,



Duplicate this building 5 times and create a row, as per the next figure:



Duplicate this row 5 times (select all 6 building and press CTRL + D) and move the duplicate to obtain a layout similar to the next figure

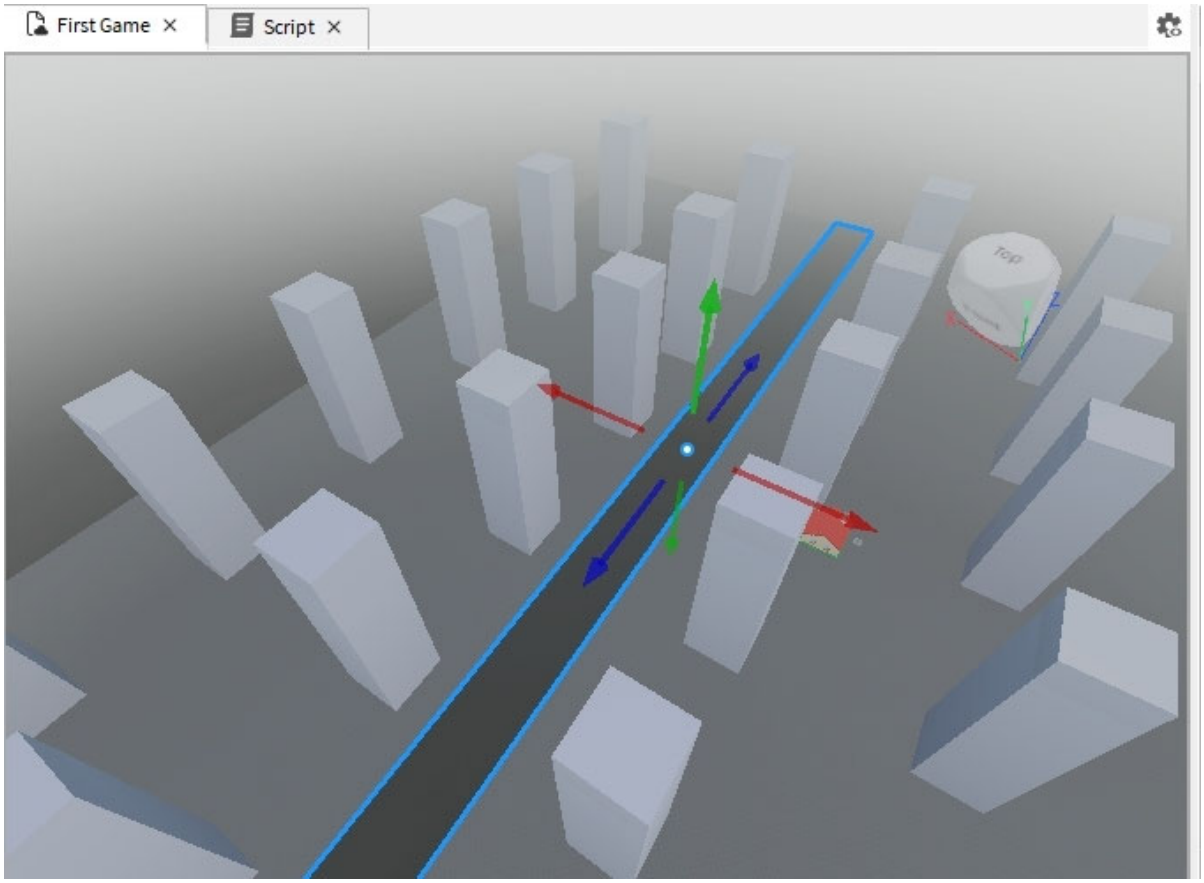


Next, we can create roads:

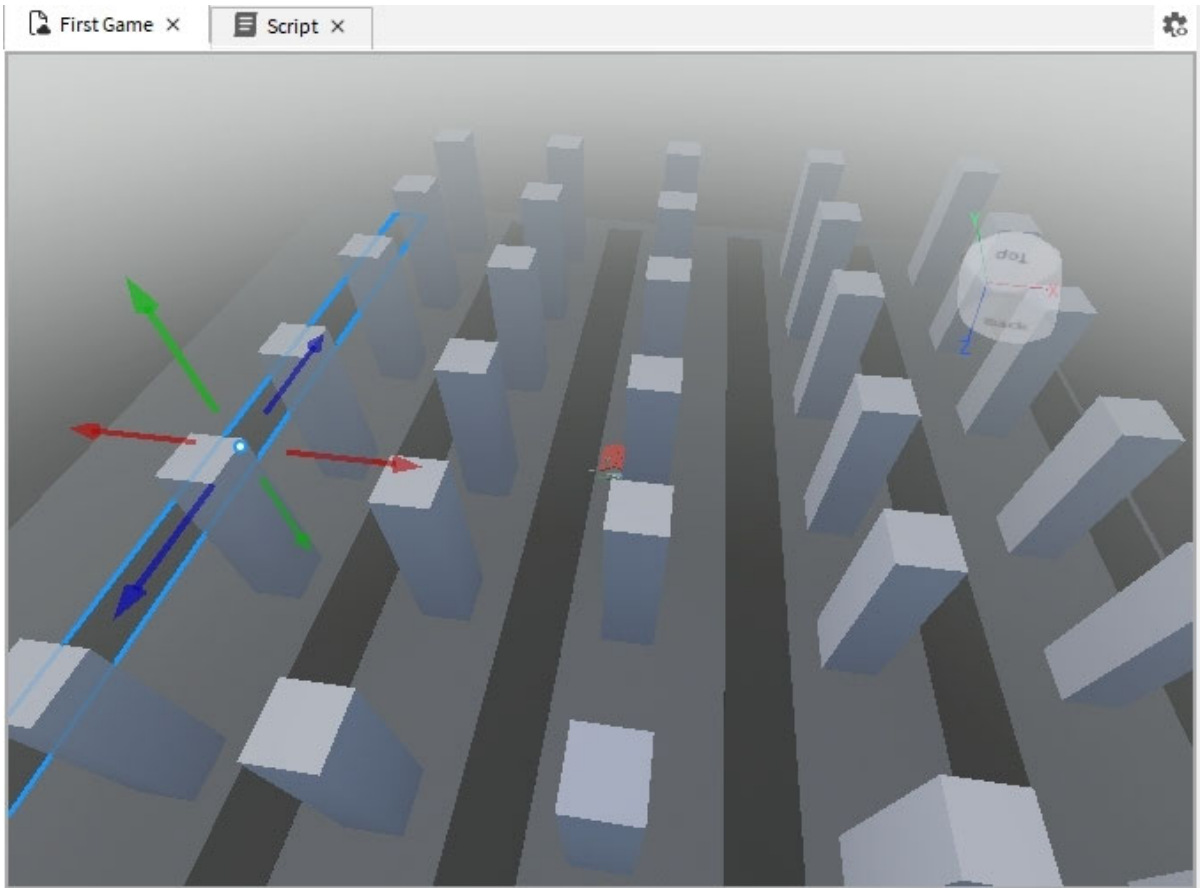
Create a new part of type Block and change its size to (100, 1,

Size	100, 1, 2047
X	100
Y	1
Z	2047

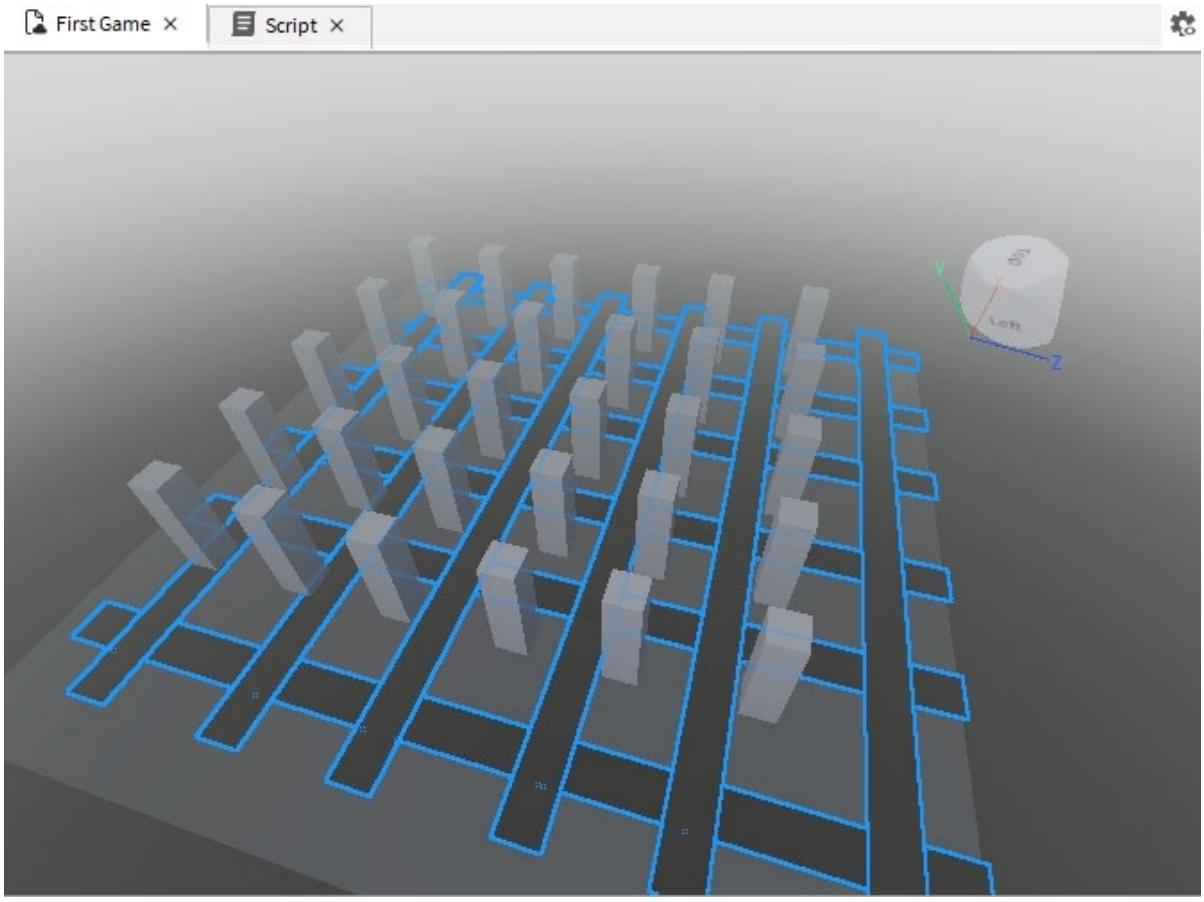
Change and its y-coordinate | to
Change its color to black and its Material to sandstone (or any other material of your choice).



Duplicate it 6 times along the x axis.



Select the 6 new objects, duplicate them, rotate them around the y axis (90 degrees) and place them to obtain the layout illustrated in the next figure.

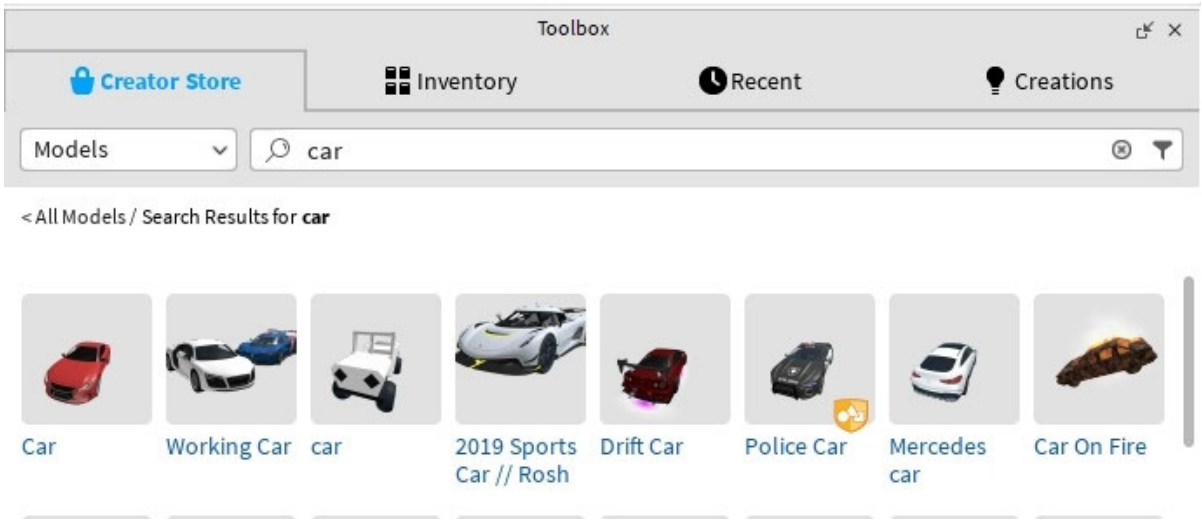


You can play the scene and see how the game looks.

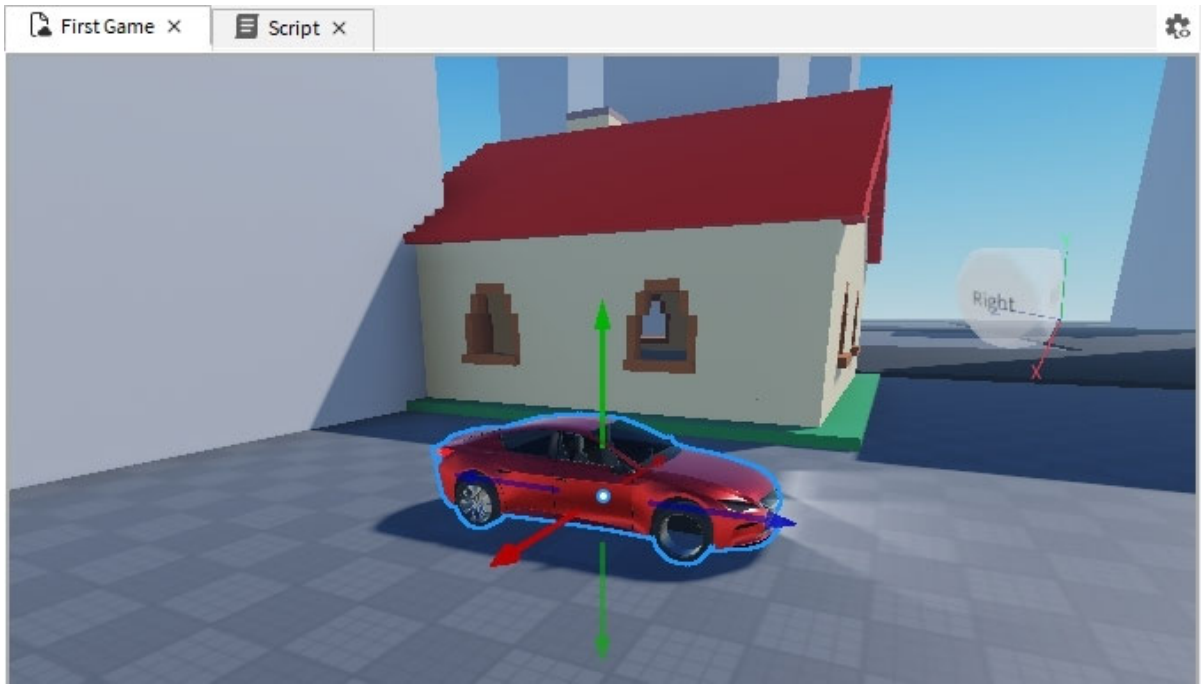


Last but not least, we will add a car so that the player character can navigate this environment faster:

In the Toolbox window, type car in the search field:



Select a car of your choice, (I have selected the object and drag and drop it to the scene.



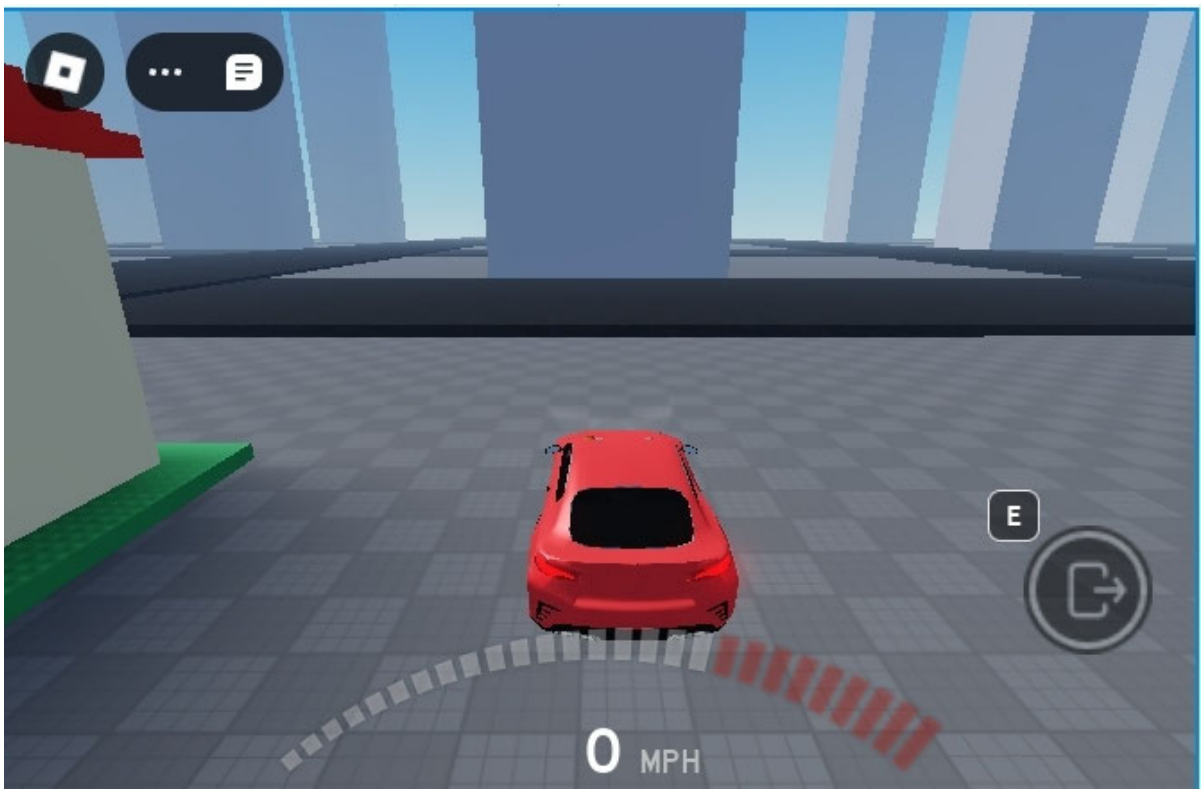
You can play the scene.



Move the character near the car until you can see the letter E along a stirring wheel, indicating that you need to press the letter E on your keyboard to drive the car.



Press the letter E on your keyboard and you should be able to drive the car



OceanofPDF.com

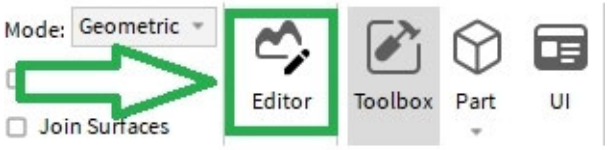
Using the Terrain Editor: Basics

The Terrain Editor in Roblox Studio allows you to create realistic landscapes such as mountains, valleys, and rivers. This tool is very useful for designing natural environments in your games.

In the next steps, we will use the Terrain Editor to create a simple hill and valleys, helping us understand how to shape the land in our game.

Please do the following:

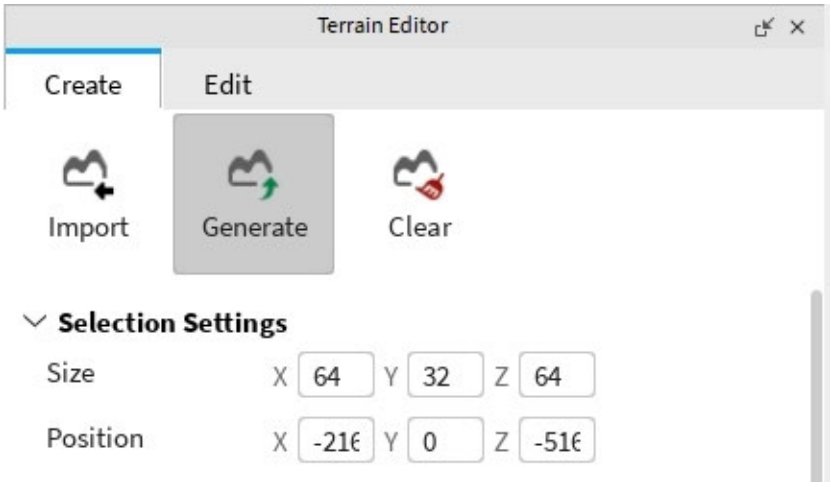
Open the Terrain Click on the Terrain tab in the toolbar.



In the Terrain select the Create tab and click on

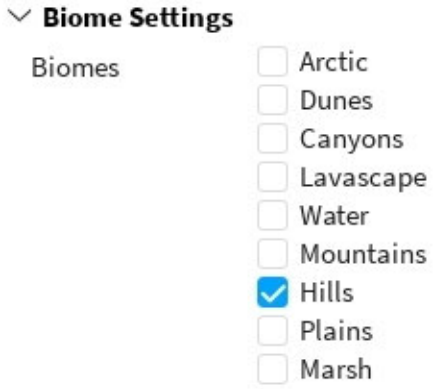


You should then see all the parameters of the terrain to be created, as per the next figure.

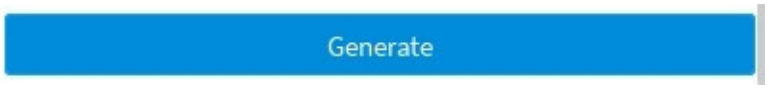


Change the size to (300, 40,

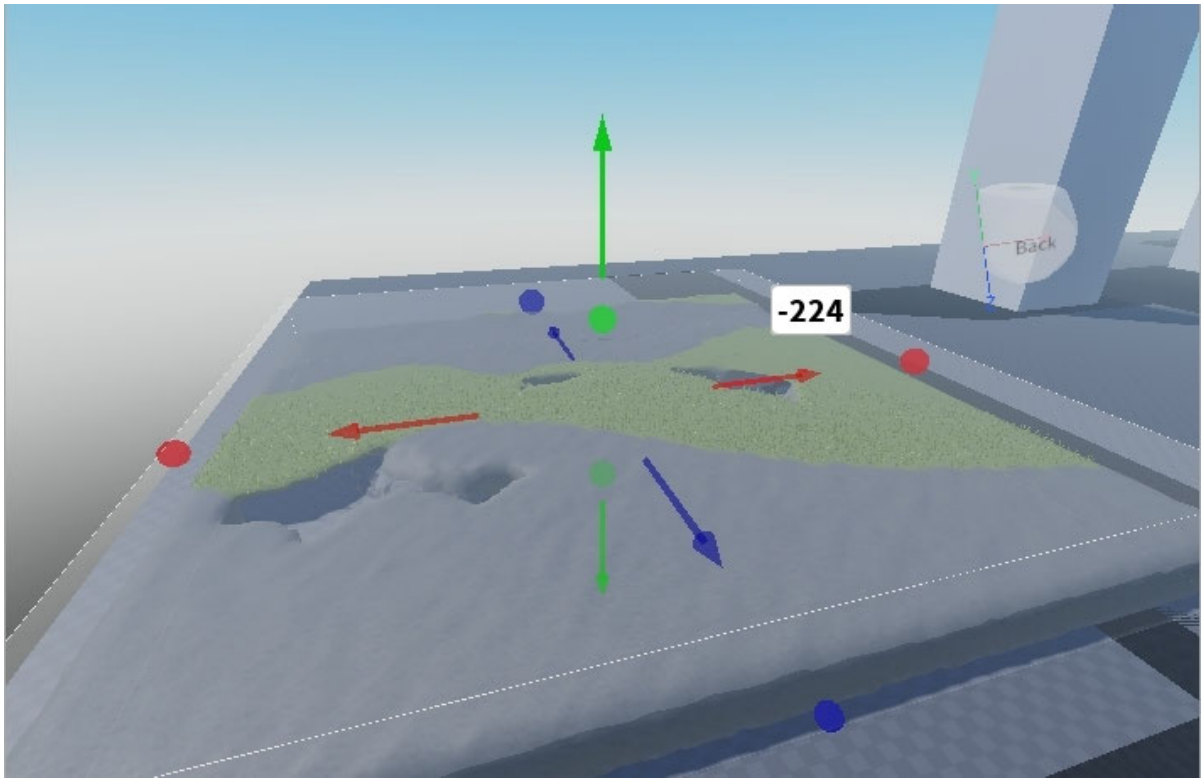
In the option Biome settings, only check the box and leave the other options empty for now.



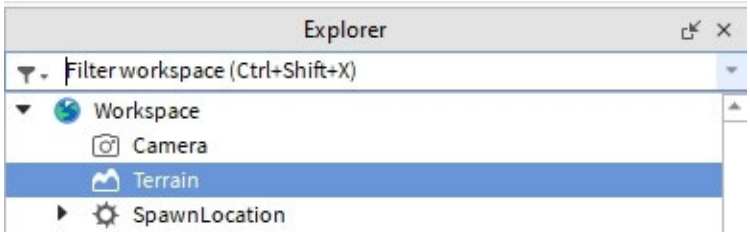
Leave all the other options as default and press the buttons.



This will create a new terrain that has been randomly generated.



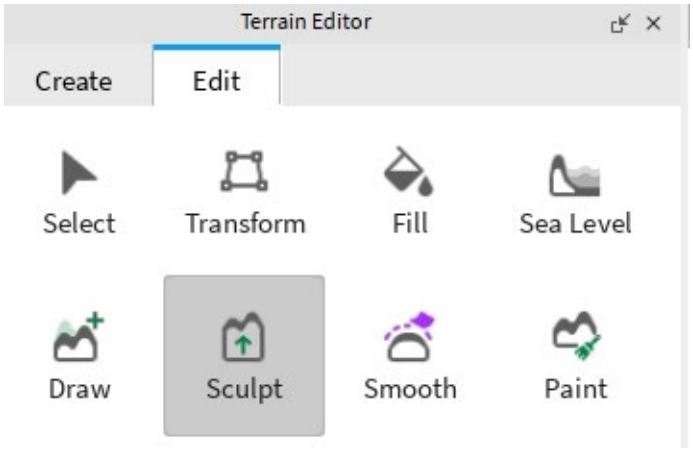
You may also notice an asset called Terrain in the Explorer window.



So, at this stage, the foundation for our terrain has been created, and we can carve and shape it as we wish to give it the appearance that we want.

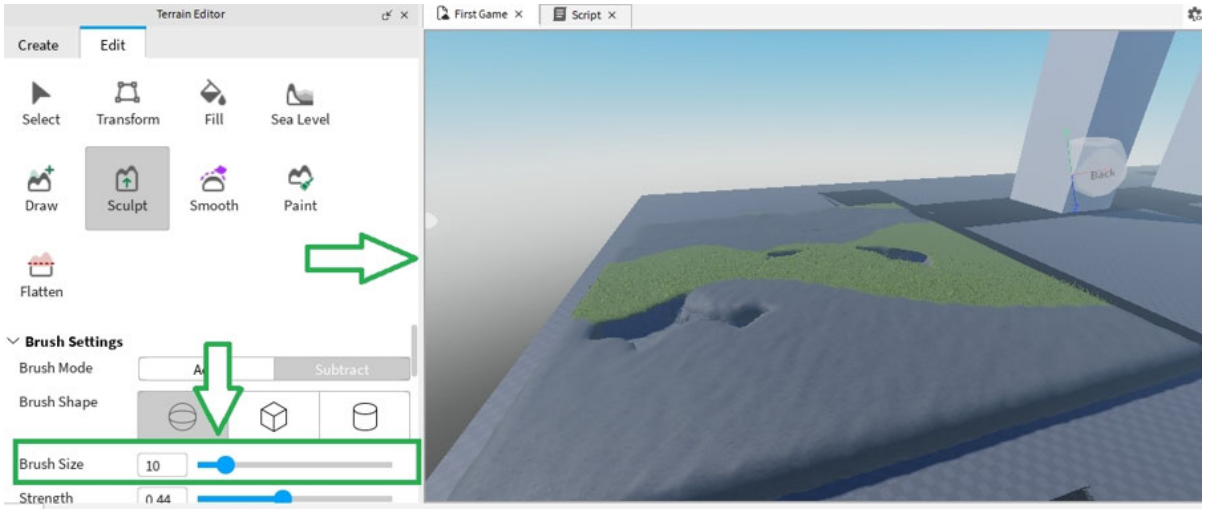
Using the Move tool, move this terrain to a place away from the building in your scene.

Click on the Edit tab, within the Terrain tool and select the option

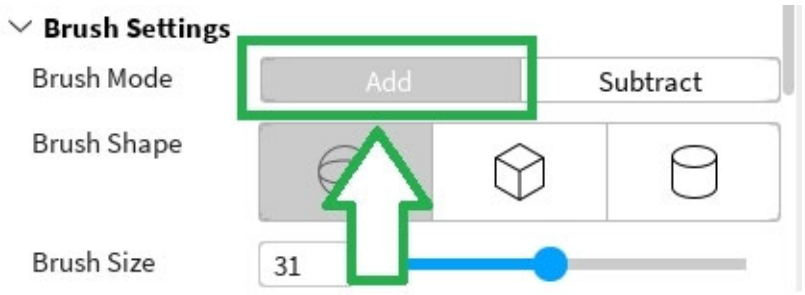


Set the Brush size to

If you can’t see the Brush Size option, you may need to expand the Terrain Editor tab, as per the next figure.



Make sure that the Brush Mode is set to as we want to add height to the terrain.



Scroll down to the section Material Settings.

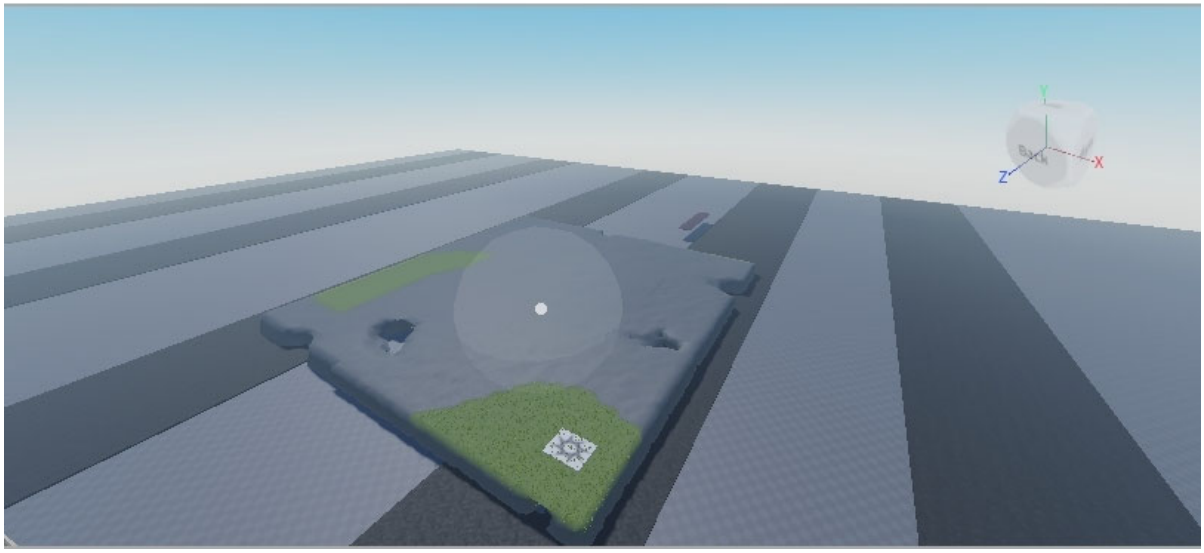


Choose (i.e., click on) the material called

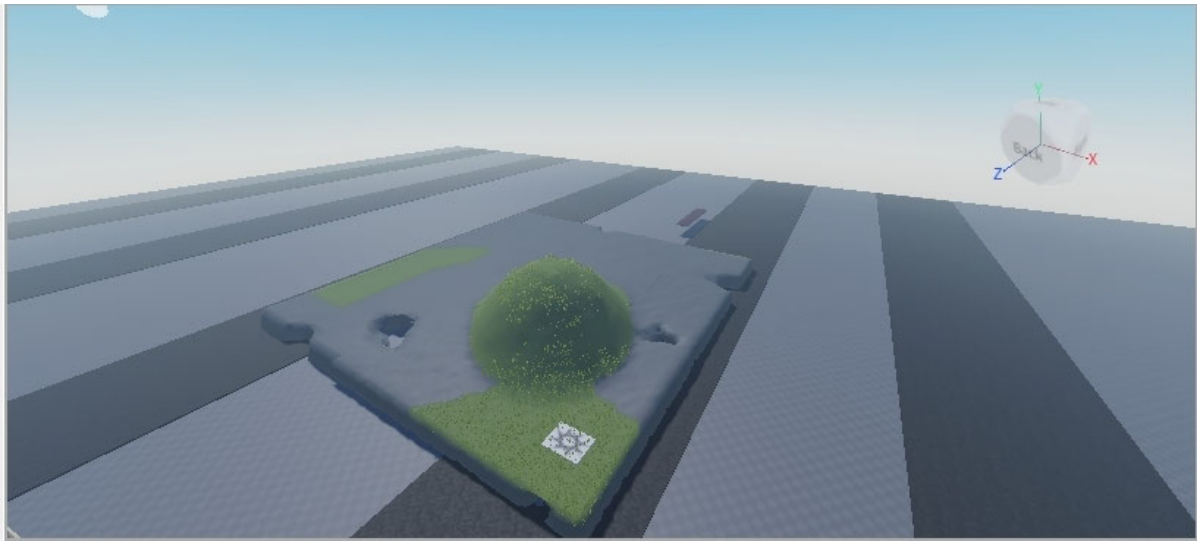


Once this is done, we can proceed with creating a hill:

Press the left mouse button in the middle of the terrain for about 5 seconds, as per the next figure.



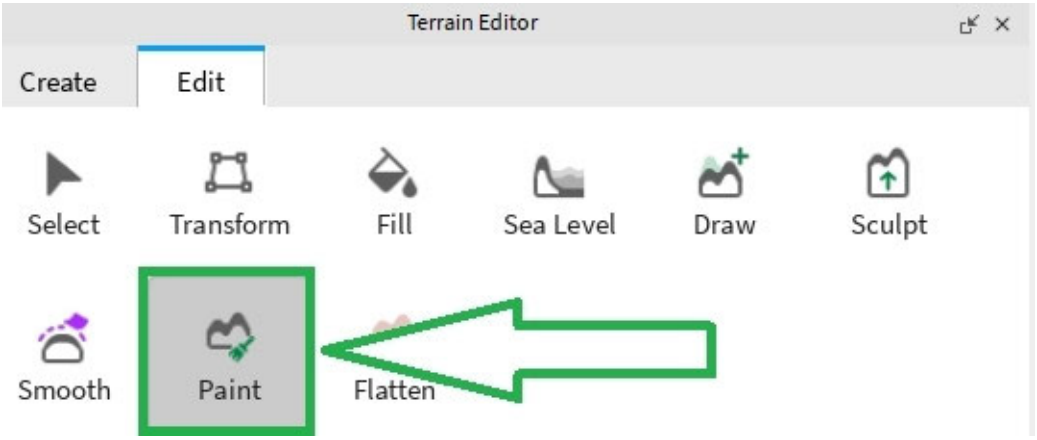
You should see a small hill formed after the 5 seconds.



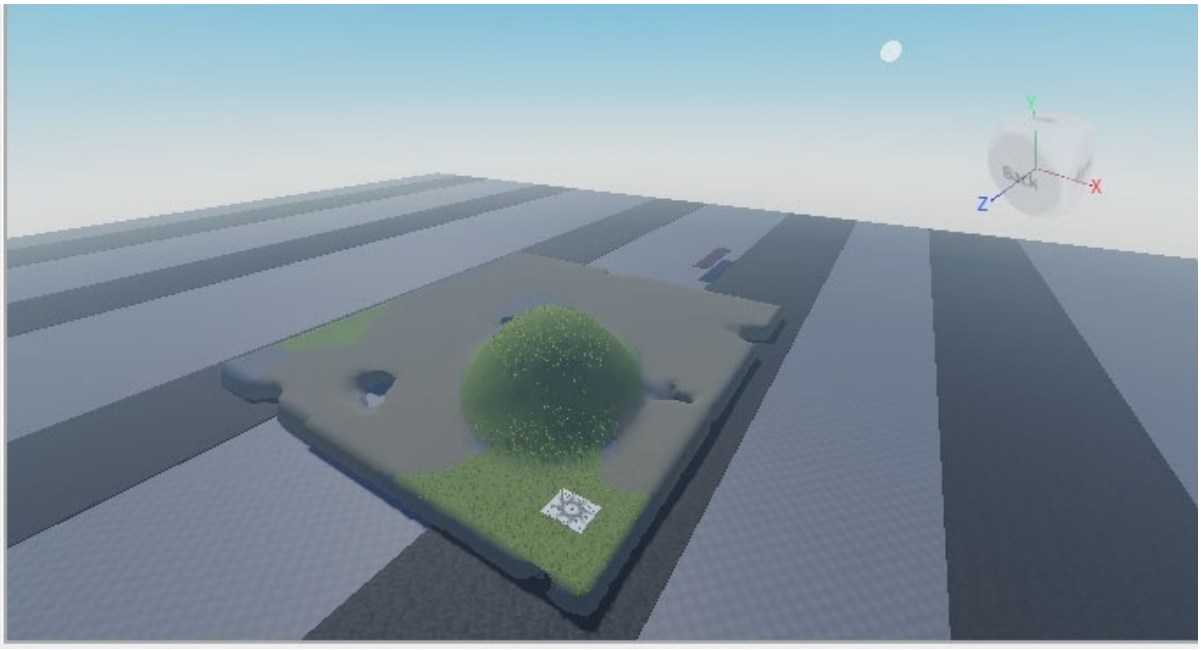
Let’s try a different color: please choose the material called



Select the Paint Mode (so that we can just paint and not raise the terrain as we have done previously).

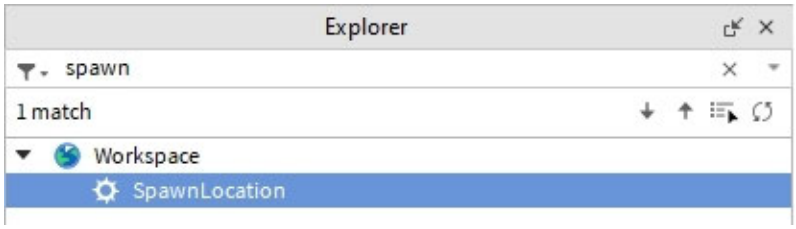


You can now paint the rest of the island: drag your mouse over the parts of the terrain that you want to pain with that texture.
Your terrain should now look like the next figure.

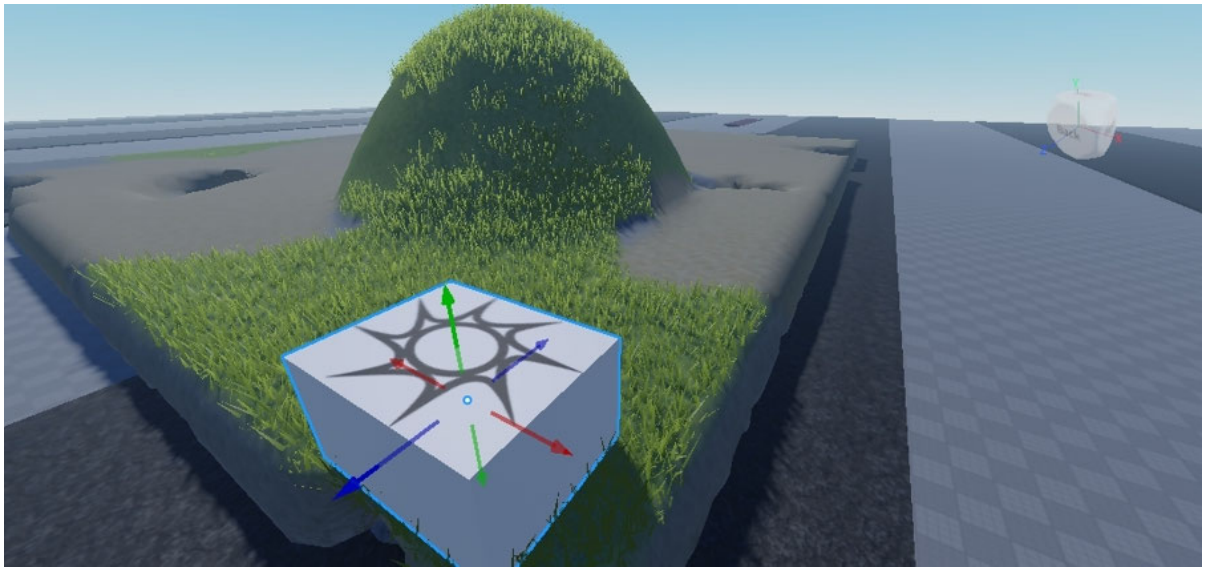


Once this is done, it’s time to see how the terrain looks like in the actual game. Before we play the scene, we may bring the player character (or the place where its spawned) closer to the terrain so that you don’t have to navigate the full scene to get to the terrain, so please do the following:

In the locate and select the object called SpawnLocation; you can use the search field within the Explorer, as per the next figure.



Using the Move tool, move that object near (or on) the terrain as per the next figure.

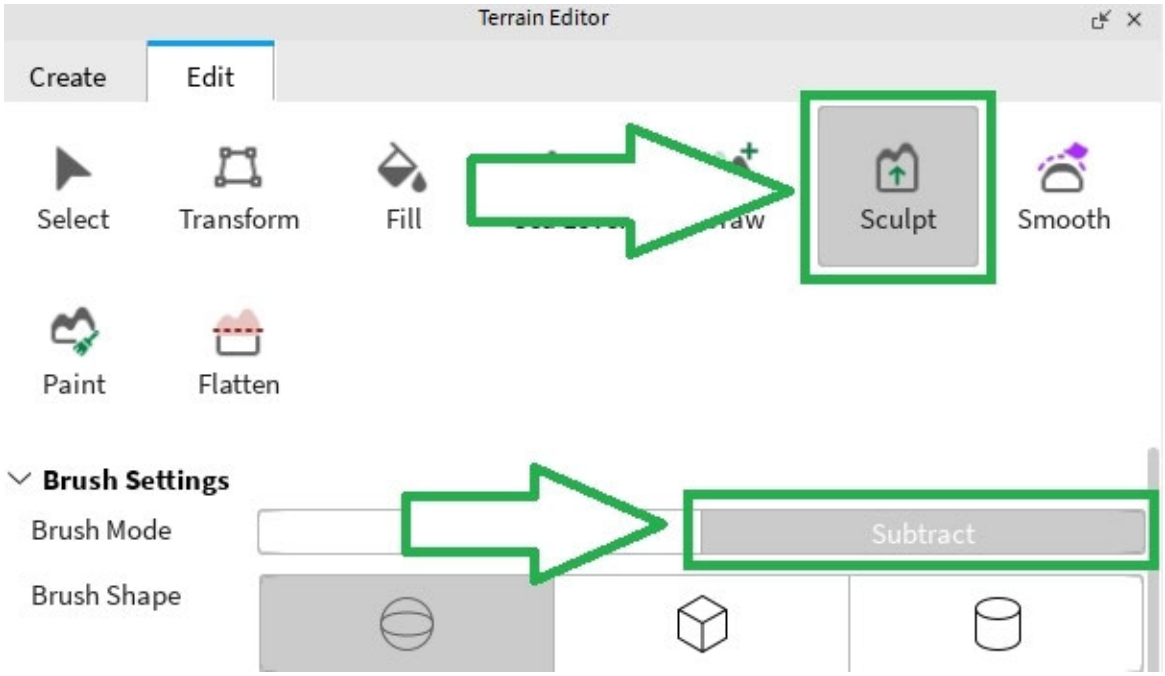


You can now play the scene; the player should spawn just on (or near) the terrain and you will be able to navigate the terrain accordingly.

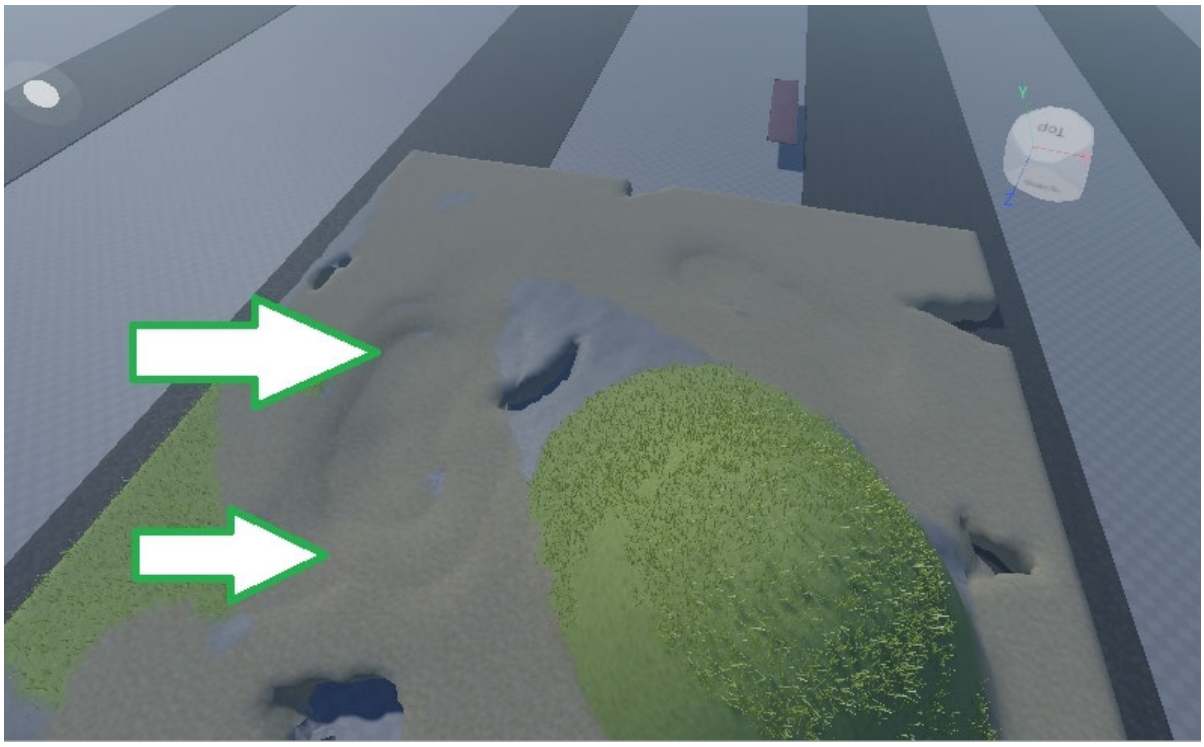


As you have seen in this section, it is relatively easy to create a hill, and we could use a similar technique to create a valley:

In the Explorer, select the Terrain object.
Using the Terrain Editor window, Select the Sculpt tool and the Subtract option in the Brush Settings (i.e., Brush



Adjust the brush size to 10 and the strength to
Click and drag in the workspace to carve out a valley. The terrain will be lowered where you drag the brush, as per the next figure.



Now that we have created basic terrain features like hills and valleys, let’s explore how to add water and more vegetation.

OceanofPDF.com

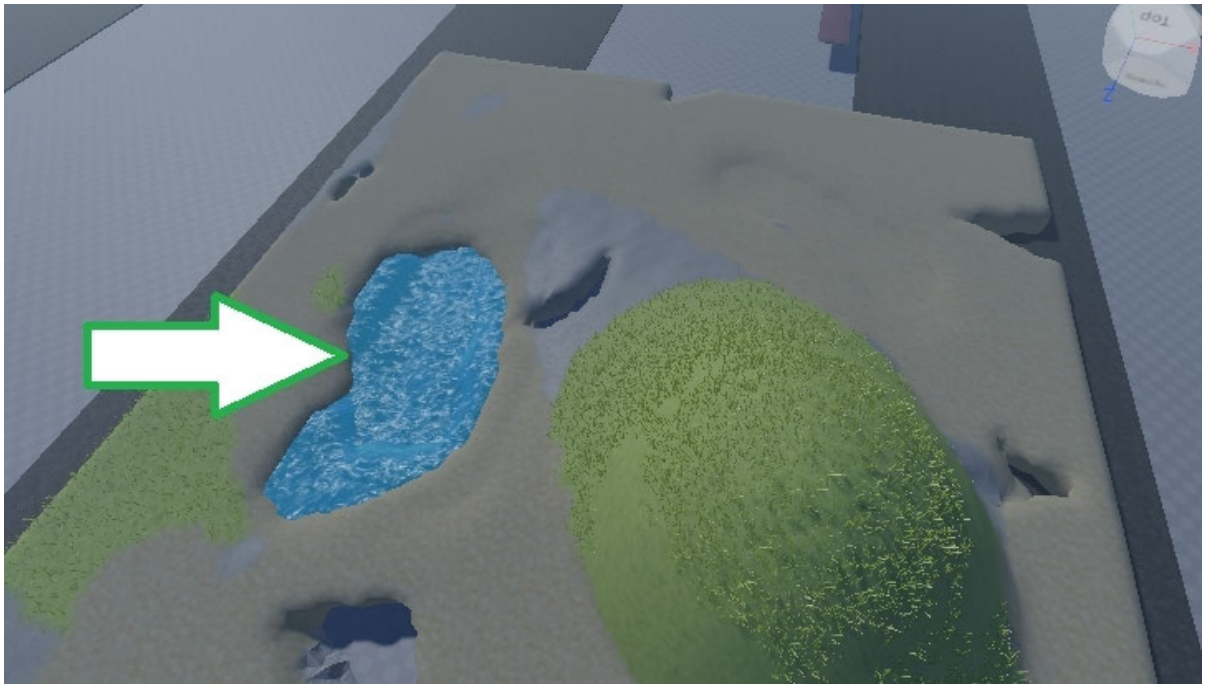
Adding Water and Vegetation

Adding water and vegetation to your terrain can bring it to life, making it more dynamic and visually interesting. This section will guide you through these additions.

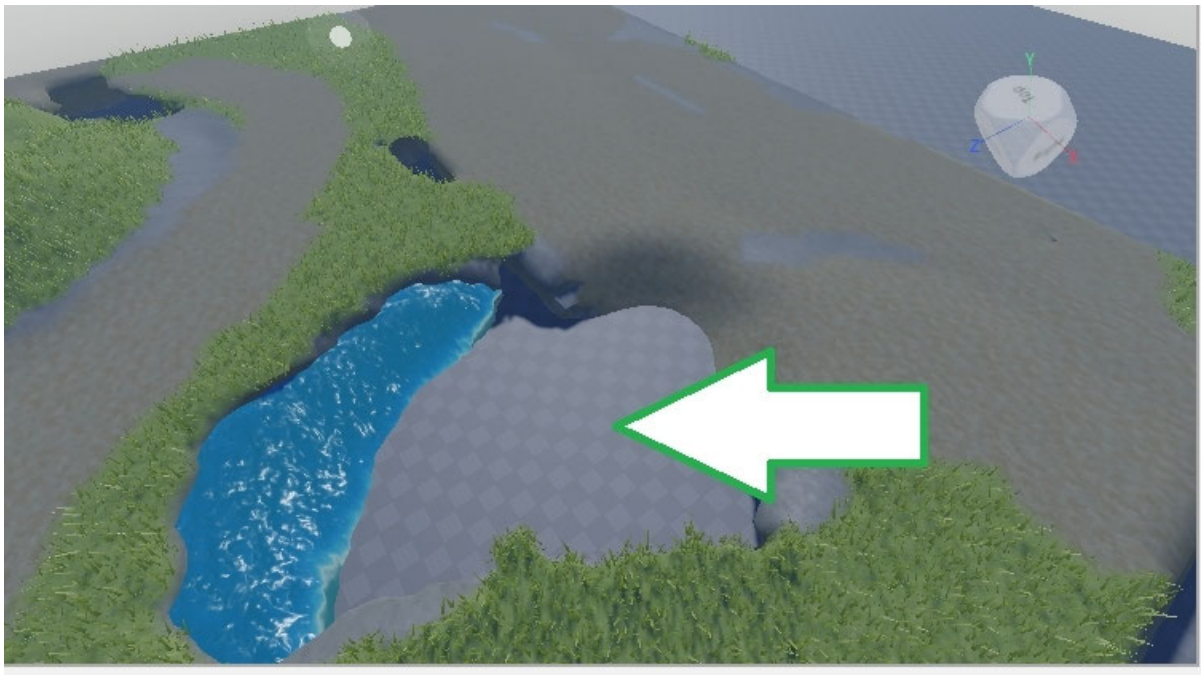
So, in the next steps, we will add a body of water and some trees to our terrain, which will help in creating a livelier and more natural environment.

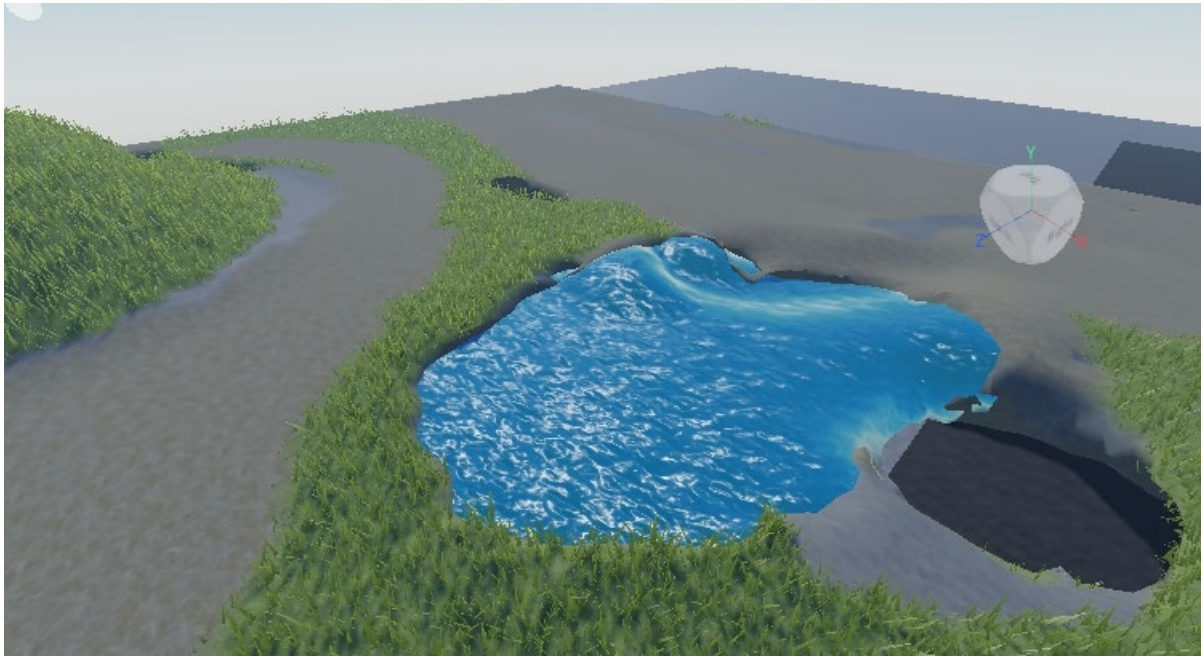
Let’s start by adding water:

In the Terrain select the Paint tool.
From the material section, choose
Click and drag to add water to your terrain, where you have created a valley earlier.

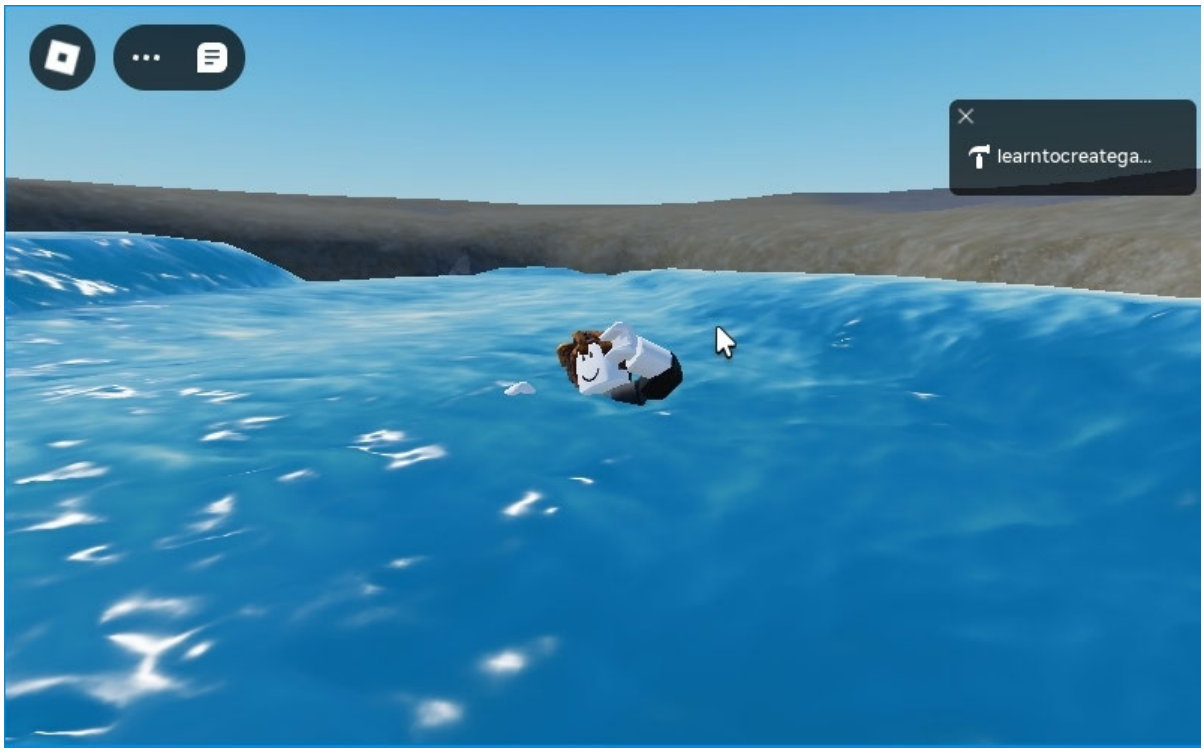


If the water doesn’t appear it is possibly because you have carved too much earlier, as per the next figure. In that case, use the Sculpt tool again to raise the terrain.



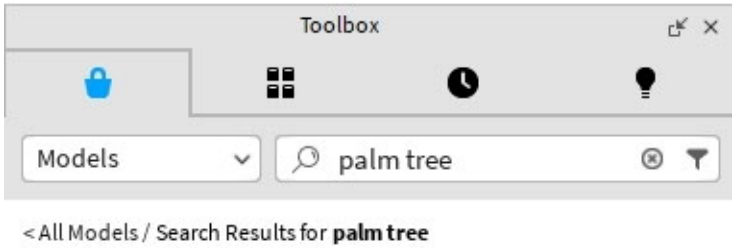


As you test your scene, and move the player character in the water, you should hear a splashing sound and see the player character swimming as you move forward in the water.



Now that we have added some water, we could also add plants to add realism:

Open the Toolbox from the View tab and search for trees or plants, for example a palm tree.

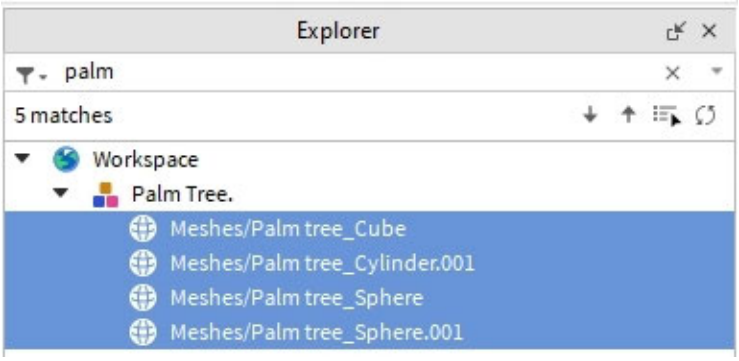


Select any of the results; I have chosen the asset called Palm

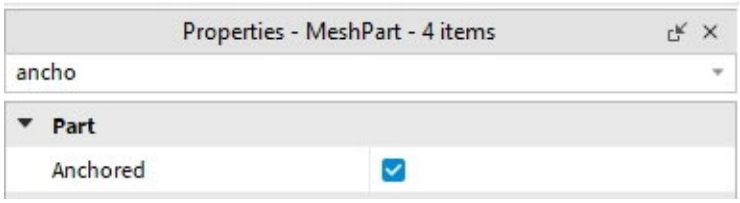


Drag and drop the palm tree onto your terrain

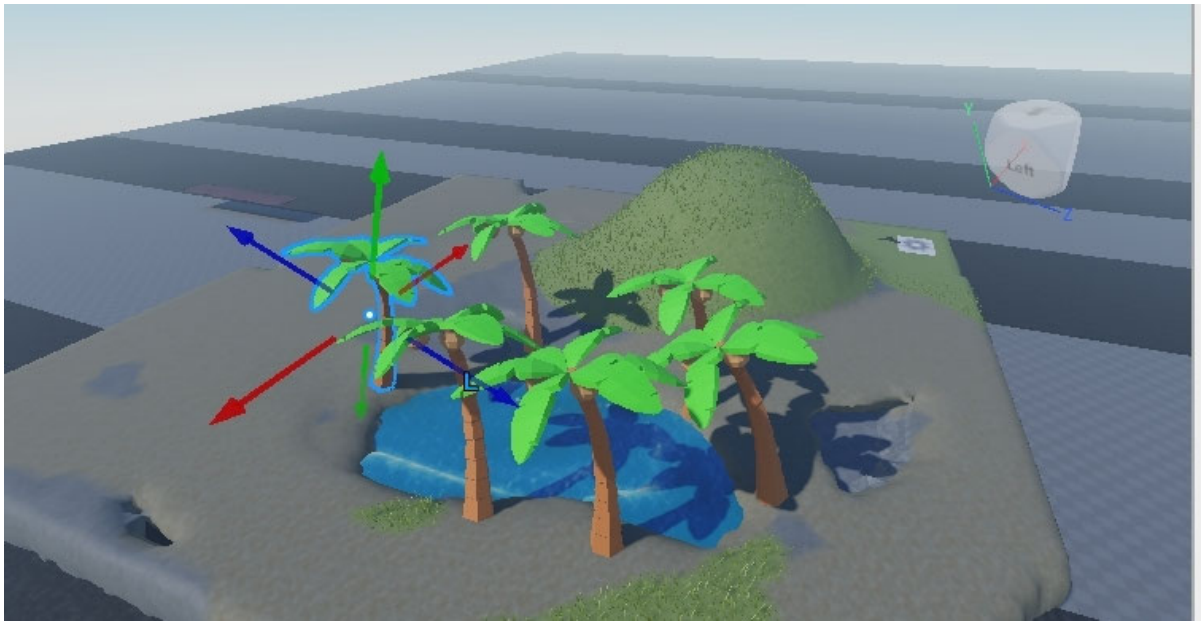
Select this object in the you should see that it includes several meshes.



Select all these meshes (and/or the part, depending on the asset). If the asset doesn't have meshes, select the asset; if it does, select the meshes within, and then switch to the Property window and tick the attribute called



Once this is done, you can duplicate this object (the palm tree) several times if needed.



Please test your scene and check that the palm trees appear properly in the game.



With water and vegetation added, your terrain should now look vibrant and full of life. Now that we have included this addition, we will explore the basics of lighting in Roblox Studio, which is important for setting the mood and visibility in your game.

OceanofPDF.com

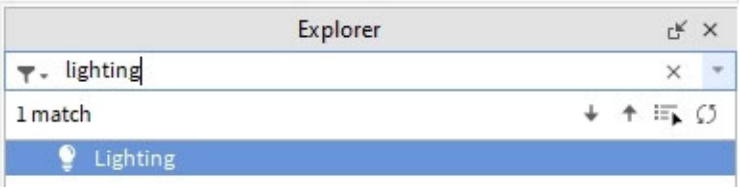
Lighting Basics

Lighting is very important for setting the mood and making sure everything is visible in your game. In this section, we'll walk through the basics of adding and adjusting lights in Roblox Studio to bring your scenes to life.

In the next steps, we will add different types of lights to our game and adjust their properties to achieve the desired effect.

Before we get to work with lights, we just need to set our world to darkness, so that we can better see the impact of the different types of light on our game.

In the select the element called



Using the Properties tab, set the attribute called Time of The Day to

Data	
Archivable	☑
ClassName	Lighting
ClockTime	23
GeographicLatitude	0
Name	Lighting
Parent	First Game
TimeOfDay	23:00:00

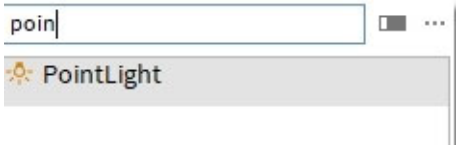
You should now see that the scene is much darker.



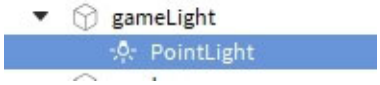
Now that the scene is darker, we will start to add lights of different types.

Create a new Sphere part and rename it
Using the Properties window, set its Transparency to 1 (i.e., fully transparent).
Set its Anchored attribute to

Add a Point Light to that object: using the right-click on the object select the option Insert and then select Point Light (or type Point Light in the search field).



This will add a Point Light as a child of the object



Select the PointLight object, and, using the Property window, set its Range attribute to

Move the object gameLight just above the water that you have created



Play the scene and check that the water area is illuminated.



You can also change the brightness of the Point Light to 9 and see how it affects the scene.



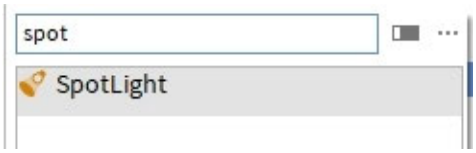
Now that we have seen how to create a Point we can create a Spot this type of light focuses on a specific spot allowing the light to be concentrated on a specific area; so it doesn’t shine in all directions like a point light; instead, its light focuses on a particular spot and emits light in the form of a cone.

Please do the following:

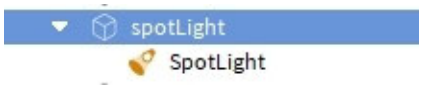
Create a new Sphere part and rename it spotlight (you can also duplicate the object gameLight and remove the object PointLight from the duplicate).
Move it apart from the previous object
Using the set its Transparency attribute to 1 (i.e., fully transparent).

Set its Anchored attribute to

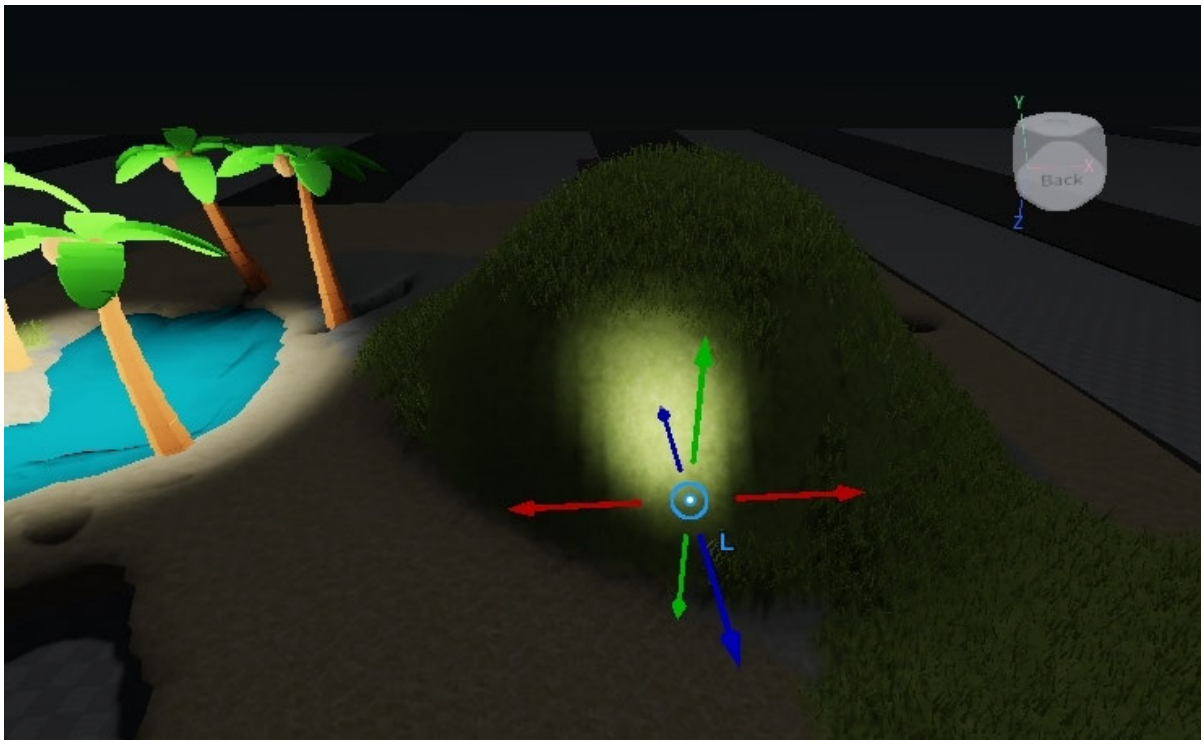
Add a Spot Light to that object: using the right-click on that object (i.e., and select the option Insert then select Spot Light (or type Point Light in the search field).



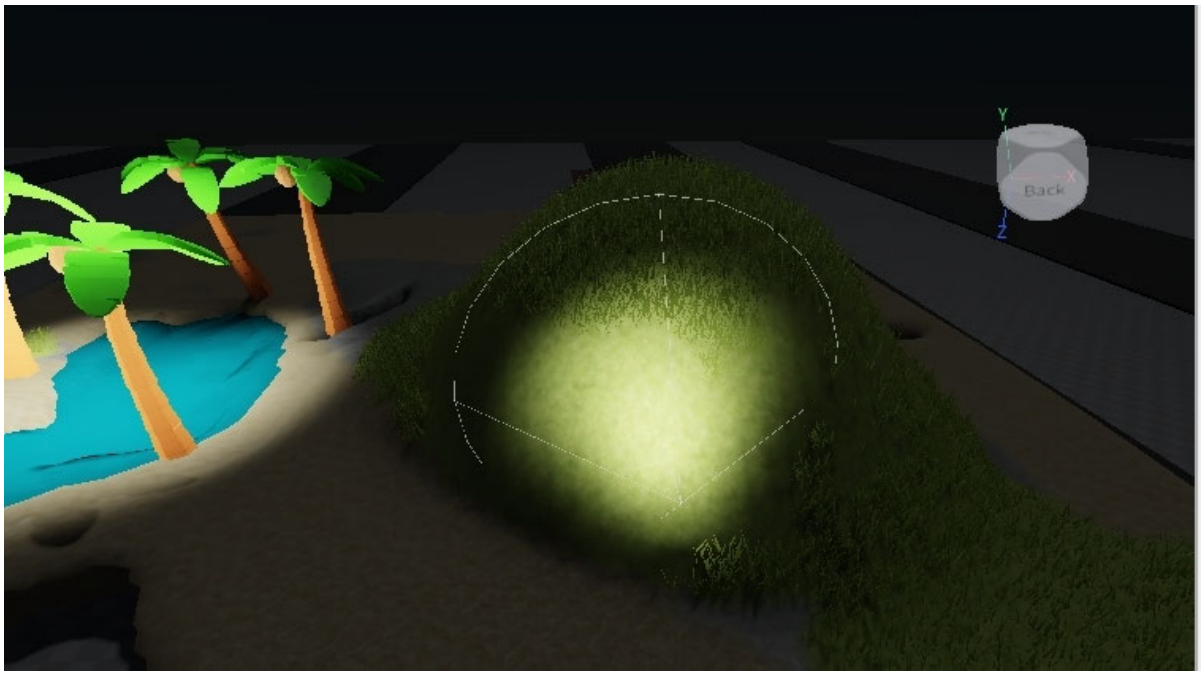
This will add a point light as a child of the object



Select the SpotLight object, and, using the Property window, set its Range attribute to 40, its Brightness to and its Face attribute to
Move the object spotLight near the hill that you have created so that it effectively shines light on the hill.



Modify the attribute called Angle and see how the range covered by the light changes.



Play the scene and check that the character is illuminated as it passes near the Spot

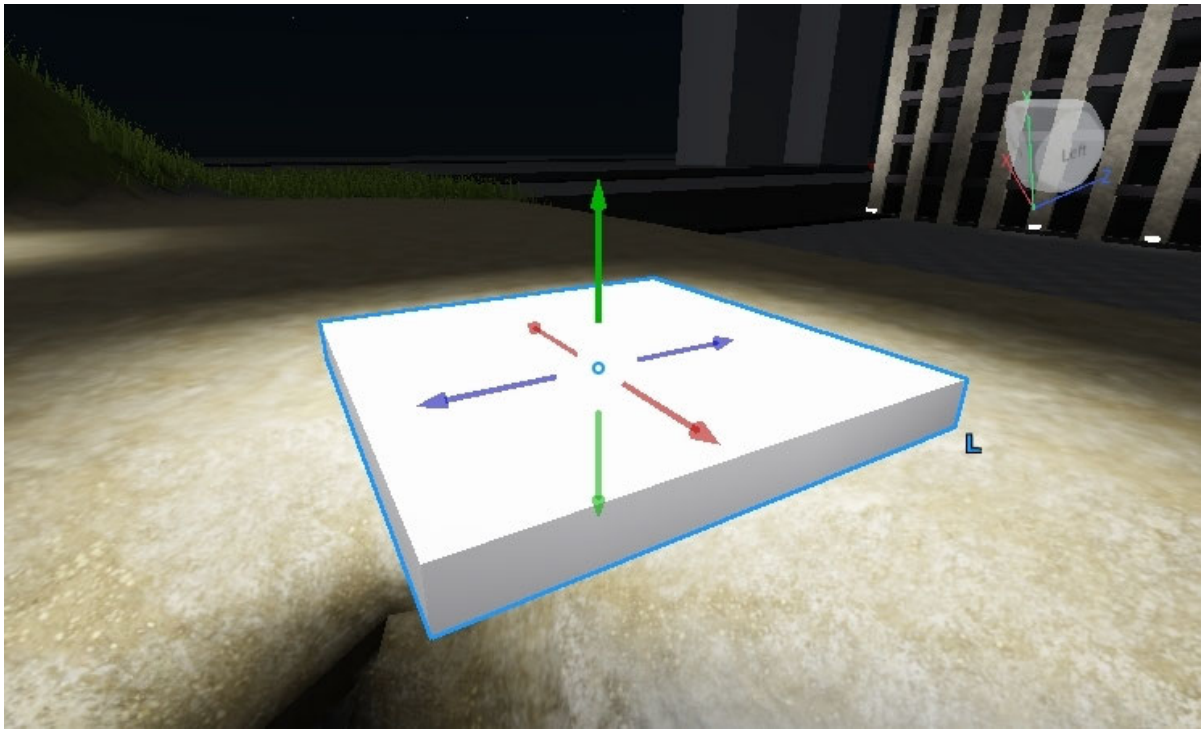


Finally, in addition to Point Lights and Spot we will can create a Surface

Please create a new part (block), rename it and change its size to
Add a Surface Light to this object, as we have done previously for the other lights: Right-click on that object then select Insert Object | Surface
Adjust the settings of the Surface set its Angle attribute to its Brightness to its Face attribute to and its range to

Appearance	
Angle	90
Brightness	18.08
Color	[255, 255, 255]
Enabled	☑
Face	Top
Range	16

You should see that the box that you have created is now emitting light.



Play your scene to see the result.



As you can see, it is rather easy to add lights to your game, and these can have a very nice impact and effect on the scene, so feel free to use them.

OceanofPDF.com

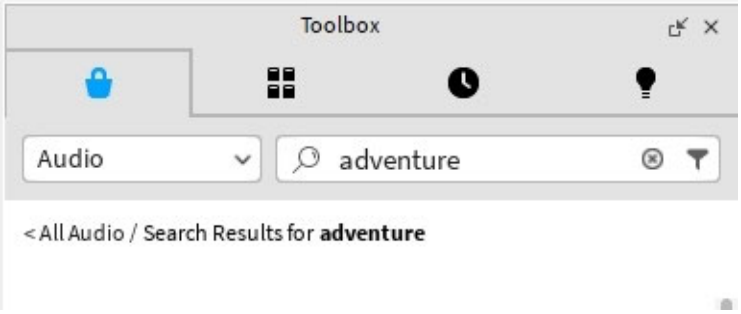
Adding Sounds and Music

Sounds and music can significantly enhance the player’s experience by setting the mood and providing audio feedback. In this section, we’ll learn how to add and control sounds in Roblox Studio.

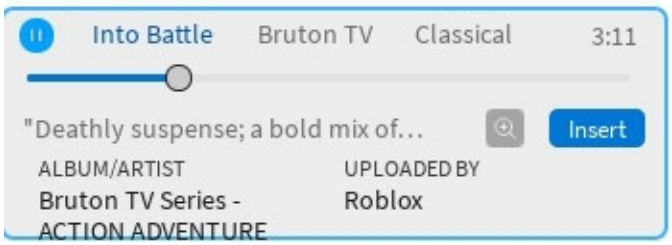
In the next steps, we will add background music and sound effects to our game, making it more immersive.

First, we will add some background music to our game.

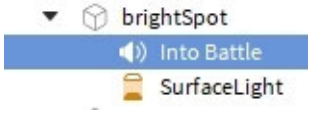
Select the object brightSpot that we have created earlier.
Open the Toolbox and search for a background music track using the Audio option.



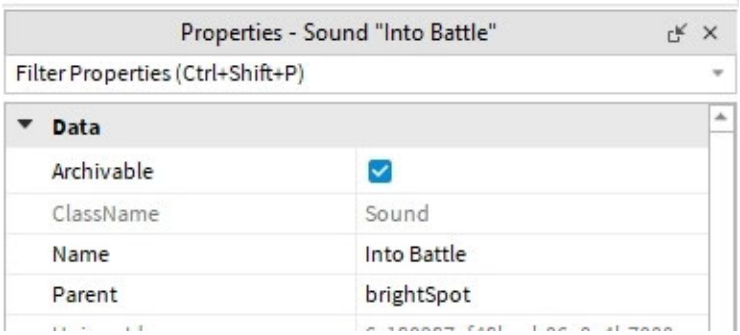
Select a track of your choice, for example



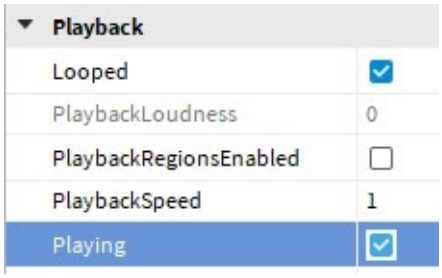
Click on Insert (or just click once on the sound).
You should now see in the Explorer that a new asset called Into Battle has been added as a child of the object



You can click on it in the and then look at the Properties window.



So that the sound plays from the beginning in a loop, please set the attributes Looped and Playing to



You can now play the game and move the character near the object you will notice the audio gets louder as you approach the object and fades as you move away.



You have effectively created a sound that is spatialized, meaning that its perceived volume will depend on your location and of any obstacle between you and that object.

Now that you’ve added music, your game will feel more dynamic and engaging. Next, we’ll learn how to optimize your builds for better performance and player experience.

OceanofPDF.com

Summary

In this chapter, we covered the basics of building in Roblox Studio, including creating and manipulating parts, understanding part properties, grouping objects, and creating simple structures. You've made significant progress in your game development journey, and your skills are growing with each step: Congratulations! In the next chapter, we will explore scripting, where you'll learn how to make your game interactive and engaging.

OceanofPDF.com

Quiz

Please guess whether the following statements are True or False (the solution is in the section Answers to at the end of the book).

The Move tool allows you to resize parts.
Grouping parts helps manage multiple parts as a single unit.
Anchored parts can move freely in the game.
The Scale tool is used to change the size of a part.
You can apply textures to parts using the
The Properties panel is where you modify part properties.
Unchecking the CanCollide property prevents parts from colliding with others.
Materials are applied in the Explorer panel.
The Rotate tool is used to change the color of a part.
Parts cannot be grouped together in Roblox Studio.

OceanofPDF.com

Challenge

Now that you've completed this chapter, try the following challenges to test your knowledge:

Build a Using the skills you've learned, create a detailed house in Roblox Studio. Make sure to use different materials and textures to give it a realistic look.
Design a Create a small park layout with benches, trees, and pathways. Use the Toolbox to add decorations and make it visually appealing.
Experiment with Take a part and experiment with all the properties available in the Properties panel. Try to understand what each property does and how it affects the part.

OceanofPDF.com

4
LUA Programming Concepts

In this section we will discover Lua programming principles and concepts, so that you can start programming. If you have already coded using Lua (or a similar language), you can skip this chapter.

After completing this chapter, you will be able to:

Understand basic Lua programming concepts.
Become familiar with variables, functions, and scope in Lua.
Understand key best practices for coding, especially in Lua.
Use conditional statements and loops effectively.

OceanofPDF.com

Introduction

When coding in a Lua-based application, you communicate with the system, giving it instructions to perform specific actions. To communicate effectively, you use a language—Lua—which consists of keywords, functions, and a syntax that ensures your instructions are understood properly. In computer science, these instructions need to be precise, unambiguous, and syntactically correct.

Lua is a lightweight, high-level scripting language. It is dynamically typed, meaning you don’t have to declare the type of variables explicitly. Lua is also easy to embed in applications, making it popular in game development.

In the next sections, we will learn how to use Lua’s syntax. If you’ve programmed in other languages, such as Python or JavaScript, some of this information may look familiar.

When scripting in Lua, you will often work with the following:

Functions
Variables
Tables

Loops
Conditional Statements
Operators
Comments
Scoping

This list may look a bit intimidating, but don't worry—we will explore these topics with hands-on examples.

OceanofPDF.com

Statements

When you write a piece of Lua code, you need to ask the system to execute your instructions (e.g., print information) using statements. A statement is literally an order or something you ask the system to do. For example, in the next line of code, the statement will tell Roblox to print a message in the Output window:

```
print("Hello World")
```

When writing statements, there are a few rules that you need to know:

Order of statements: Each statement is executed in the order it appears in the script. For example, in the next example, the code will print "hello," then "world"; this is because the associated statements are in that particular sequence.

```
print("hello")
```

```
print("world")
```

Statements are not separated by semicolons: Unlike many other programming languages, Lua does not require semicolons at the end of each statement, although you can use them if you prefer. Statements can also be added on the same line, separated by a semicolon, but this is optional.

```
print("hello"); print("world")
```

Multiple spaces are ignored for statements: However, it is good practice to add spaces around operators such as +, -, /, or % for clarity. For example, in the next example, we say that a is equal to b. There is a space both before and after the = operator.

```
a = b
```

Grouping statements together: Statements to be executed together (e.g., based on the same condition) can be grouped using what is usually referred to as code blocks. In Lua, code blocks are symbolized by the keywords do and end. So, in other words, if you needed to group several statements, we would include them all within the same set of as follows:

```
do

    print("hello stranger!")

    print("today, we will learn about scripting")

end
```

As we have seen earlier, a statement usually employs or starts with a keyword (i.e., a word that the computer understands). All these keywords have a specific purpose, and the most common ones (at this stage) are used for:

Printing a message in the Output window: The keyword is
Declaring a variable: In Lua, variables are typically declared with or without the local keyword, depending on the scope, and the type of the

variable is inferred from the value assigned (e.g., number, string, boolean).

Declaring a function: The keyword used is function. Lua does not require specifying the type of the data returned by the function, unlike in C#. You simply define a function and use the return keyword if you want to return a value.

```
function myFunction()

    return "Hello"

end
```

Marking a block of instructions to be executed based on a condition: The keywords are
Exiting a function: The keyword is

OceanofPDF.com

Comments

In Lua, you can use comments to explain the code and to make it more readable. This becomes important as the size of your code increases and it is also important if you work in a team, so that the other team members can understand your code and make amendments in the right places, if and when it is needed.

The code that is commented it is not executed. There are two ways to comment your code in Lua: you can use single or multi-line comments.

Single-line comments: In single-line comments, two dashes are added at the start of a line or after a statement, so that this line (or part thereof) is commented, as illustrated in the next code snippet.

```
-- This is a single-line comment

print("Hello")

-- This prints Hello
```

Multi-line comments: In multi-line comments, any text between --[[and]] will be commented and not executed. This is also referred as comment as illustrated in the next code snippet.

```
--[[

This is a multi-line comment.

It spans multiple lines.

]]

print("Hello World")
```

In addition to providing explanations about your code, you can also use comments to prevent part of your code from being executed. This is very useful when you would like to debug your code and find where the errors or bugs might be, using a very simple method. By

commenting sections of your code, and by using a process of elimination, you can usually find the issue quickly. For example, you can comment all the code and run the script, then comment half the code, and run the script. If it works, it means that the error is within the code that has been commented, and if it does not work, it means that the error is in the code that has not been commented. In the first case (if the code works), we could then just comment half of the portion of the code that has already been commented. So, by successively commenting more specific areas of our code, we can get to discover what part of the code includes the bug. This process is often called dichotomy as we successively divide a code section into two. It is usually effective to debug your code because the number of iterations (dividing part of the code in two) is more predictable and also potentially less time-consuming. For example, for 100 lines of codes, we can successively narrow down the issue to 50, 25, 12, 6, and 3 lines and therefore use 5 to 6 iterations in this case, rather than of going through the whole 100 lines of code

---

OceanofPDF.com

Variables

A variable is a container. It includes a value that may change overtime. When using variables, we usually need to: (1) declare the variable by specifying its type (e.g., local or global), (2) assign a value to this variable, and (3) possibly combine this variable with other variables using operators. This is illustrated in the next code snippet.

---

local myAge = 20 -- Variable holding an integer

myAge = myAge + 1 – Adding one to myAge

local name = "John" -- Variable holding a string

Note that variables in Lua are dynamically typed, meaning their type is determined at runtime. You don’t need to declare the type explicitly.

In Lua, you can declare a variable using the local keyword, which limits the variable's scope to the block where it is declared. Variables declared without local are global by default, which means they can be accessed anywhere in the script.

Note that in the previous code, we assigned the value myAge + 1 to The = operator is an assignment operator; in other words, it assigns a value to a variable and is not to be understood in a strict algebraic sense (i.e., that the values or variables on both sides of the = sign are equal).

In Lua, you can declare multiple variables in the same statement without specifying their type, as Lua is dynamically typed. For example, in the next code snippet, we declare three variables, and v3 in one statement:

```
local v1, v2, v3

local v4, v5, v6 = 4, 5, 6
```

In the previous code, the first line is used to declare the variables and In the second line, not only do we declare three other variables

simultaneously, but we also initialize them by setting a value.

When using variables in Lua, there are a few things that we need to determine, including their name, their type (which is inferred), and their scope:

Name of a variable: A variable is usually given a unique name so that it can be identified uniquely. The name of a variable is referred to as an identifier; it can contain letters, digits, underscores, but usually begins with a letter. Identifiers cannot be Lua keywords. For example, the keyword cannot be a variable name.
Type of variable: Variables can hold several types of data in Lua, including numbers (which can be integers or floats), text (i.e., strings), Boolean values (i.e., true or false), tables (which can be used like arrays or objects), as illustrated in the next code snippet:

---

local myName = "Patrick" -- The text is declared using double quotes

local currentYear = 2015 -- The year is declared as a number (integer)

local width = 100.45 -- Width is declared as a number with decimals

---

Variable declaration: A variable needs to be declared using the local keyword if you want it to be limited in scope to the block it is declared in. Lua does not require you to declare the type of the variable explicitly, as the type is determined automatically based on the value assigned. If you do not assign a value immediately, the variable will hold nil until you assign a value:

---

local myName

myName = "My Name"

In the previous example, we declare a variable called myName and then assign the value to it.

Scope of a variable: A variable can be accessed (i.e., referred to) in specific contexts that depend on where in the script the variable was initially declared. If declared with it is limited to the block, function, or file where it is declared. If declared without local, it is global and can be accessed anywhere in the program.

Common variable types in Lua include:

String: Text, enclosed in double or single quotes.

Number: Can be an integer or a floating-point number.
Boolean: true or false.
Table: A data structure in Lua that can be used as an array, dictionary, or object.

OceanofPDF.com

Operators

Once we have declared and assigned values to a variable, we can use operators to modify or combine variables. There are different types of operators in Lua, including arithmetic operators, assignment operators, comparison operators, and logical operators.

Arithmetic Arithmetic operators are used to perform arithmetic operations such as additions, subtractions, multiplications, or divisions. Common arithmetic operators in Lua include +, -, *, /, and % (modulo).

```
local number1 = 1  -- the variable number1 is declared

local number2 = 1  -- the variable number2 is declared

local sum = number1 + number2  -- adding two numbers and storing the result in sum

local sub = number1 - number2  -- subtracting two numbers and storing the result in sub
```

Assignment Assignment operators are used to assign a value to a variable. In Lua, the basic assignment operator is =, but you can combine assignment with arithmetic using +=, -=, *=, /=, and %=. However, in Lua, these combined assignment operators are not natively supported as in some other languages, so you would write them out explicitly.

```
local number1 = 1

local number2 = 1



number1 = number1 + 1  -- equivalent to number1 += 1 in other languages

number1 = number1 - 1  -- equivalent to number1 -= 1

number1 = number1 * 1  -- equivalent to number1 *= 1

number1 = number1 / 1  -- equivalent to number1 /= 1
```

```
number1 = number1 % 1  -- equivalent to number1 %= 1
```

String In Lua, the .. operator is used for string concatenation. When you use .. with strings, it will concatenate them (i.e., join them together to create a new string). For example, "Hello" .. "1" will result in "Hello1".

```
local greeting = "Hello" .. " World"  -- concatenates "Hello" and "World" to form "Hello World"
```

Comparison Comparison operators are often used in conditions to compare two values. Lua's comparison operators include ==, ~= (not equal), >, <, >=, and <=.

```
if number1 == number2 then  -- if number1 equals number2

    print("They are equal")

end
```

```
if number1 ~= number2 then  -- if number1 and number2 have
different values

    print("They are not equal")

end

if number1 > number2 then  -- if number1 is greater than number2

    print("Number1 is greater")

end


if number1 >= number2 then  -- if number1 is greater than or equal to
number2

    print("Number1 is greater or equal")

end

if number1 < number2 then  -- if number1 is less than number2

    print("Number1 is less")
```

```
end
```

```
if number1 <= number2 then  -- if number1 is less than or equal to
number2

    print("Number1 is less or equal")

end
```

OceanofPDF.com

Conditional Statements

Statements can be performed based on a condition, and in this case, they are called conditional statements. The syntax in Lua is as follows:

```
if condition then
    statement
end
```

This means if the condition is verified (or true), then (and only then) the statement is executed. When we assess a condition, we test whether a declaration is true. For example, by typing if a == b then, we mean "if it is true that a equals b." Similarly, if we type if a >= b then, we mean "if it is true that a is greater than or equal to b."

As we will see later on, we can also combine conditions. For example, we can decide to perform a statement if two (or more) conditions are true. For example, by typing if a == b and c == 2 then, we mean "if a

equals b and c equals 2." In this case, using the operator and means that both conditions will need to be true. We could compare this to making a decision on whether we will go sailing tomorrow. For example, "if the weather is sunny and the wind speed is less than 5km/h, then I will go sailing." We could translate this statement as follows:

```
if weatherIsSunny == true and windSpeed < 5 then

    iGoSailing = true

end
```

When creating conditions, as for most natural languages, we can use the operator Taking the previous example, we could translate the following sentence "if the weather is too hot or if the wind is faster than 5km/h, then I will not go sailing" as follows:

```
if weatherIsTooHot == true or windSpeed > 5 then

    iGoSailing = false
```

```
end
```

Another example could be as follows:

```
if myName == "Patrick" then
    print("Hello Patrick")
else
    print("Hello Stranger")
end
```

When we deal with combining true or false statements, we are effectively applying what is called Boolean logic. Boolean logic deals with Boolean variables that have two values: true and false. By evaluating conditions, we are effectively processing Boolean values and applying Boolean logic. While you don’t need to know about

Boolean logic in depth, some operators for Boolean logic are important, including the not operator. It means NOT or the opposite. This means that if a variable is true, its opposite will be false, and vice versa. For example, if we consider the variable weatherIsGood = true, the value of not weatherIsGood will be false (its opposite). So the condition if weatherIsGood == false then could also be written as if not weatherIsGood which would literally translate as "if the weather is NOT good."

OceanofPDF.com

Combining if/elseif

In Lua, there isn't a direct equivalent to C#'s switch statements, but we can achieve similar functionality using if...elseif...else structures. The idea behind this structure is that, depending on the value of a specific variable, we will branch to a particular portion of the code and perform one or several actions.

Let’s look at a simple example that simulates a switch statement in Lua:

```
local choice = 1

if choice == 1 then

    print("You chose 1")

elseif choice == 2 then

    print("You chose 2")
```

```
elseif choice == 3 then
    print("You chose 3")
else
    print("Default option")
end
print("We have exited the conditional structure")
```

In the previous code:

We declare the variable choice and initialize it to

We then create a series of statements whereby, depending on the value of the variable the program will branch to the relevant section (i.e., the portion of code starting with if choice == 1, elseif choice == 2, etc.). Note that in our code, we check for the values 1, 2, or 3. However, if the variable choice does not equal 1, 2, or 3, the program will execute the section under else. This is because this section is executed if none of the other possible choices (i.e., 1, 2, or 3) have been fulfilled (or selected).

In Lua, each condition is checked sequentially until one is found to be true. Once a condition is true, its corresponding block of code is executed, and the subsequent conditions are ignored.

Let’s consider the previous example and see how this would work. In our case, the variable choice is set to so the program will enter the if...elseif...else structure, and then look for the section that deals with a value of 1 for the variable This will be the section that starts with if choice == then the command print("You chose 1") will be executed, followed by the command print("We have exited the conditional

This kind of structure is very useful to organize your code when dealing with mutually exclusive choices (i.e., when only one of the choices can be processed) based on a variable's value, especially in the case of menus. Additionally, using if...elseif...else structures make for cleaner and easily understandable code in Lua.

OceanofPDF.com

Loops

There are times when you have to perform repetitive tasks as a programmer; many times, these can be streamlined using loops. Loops are structures that will perform the same actions repetitively based on a condition. The process is usually as follows:

Start the loop.
Perform actions.
Check for a condition.
Exit the loop if the condition is fulfilled or keep looping.

Sometimes the condition is checked at the start of the loop, while other times it is checked at the end of the loop.

Let’s look at the following example that uses a while loop in Lua:

```
local counter = 0

while counter <= 10 do
```

```
    print("Counter = " .. counter)

    counter = counter + 1

end
```

In the previous code:

We set the value of the variable
We then create a loop that is delimited by the do and end keywords, and that starts with the keyword
We set the condition to remain in this loop (i.e., counter <=
Within the loop, we increase the value of the variable counter by 1 and print its value.

So effectively:

The first time we go through the loop: The variable counter is increased to 1; we reach the end of the loop; we go back to the start of the loop and check if counter <= 10; this is true in this case (counter = 1).
The second time we go through the loop: Counter is increased to 2; we reach the end of the loop; we go back to the start of the loop and check if counter <= 10; this is true in this case (counter = 2).

…
The 11th time we go through the loop: Counter is increased to 11; we reach the end of the loop; we go back to the start and check if counter <= 10; this is now false in this case (counter = 11). As a result, we then exit the loop.

As you can see, using a loop, we have managed to increment the value of the variable counter iteratively, from 0 to 11, but using less code than would be needed otherwise.

Now, we could create a slightly modified version of this loop; let’s look at the following example:

```
local counter = 0

repeat

    print("Counter = " .. counter)

    counter = counter + 1

until counter > 10
```

In this example, you may spot two differences compared to the previous code:

The while keyword is now replaced by and it is placed at the end of the loop. So, the condition will be evaluated (or assessed) at the end of the loop.
A repeat keyword is now featured at the start of the loop.
In this example, we perform the statements and then check for the condition.

Another variation of the loop could be as follows:

```
for counter = 0, 10 do

    print("Counter = " .. counter)

end
```

In the previous code:

We declare a loop in a slightly different way: we say that we will use a variable called counter that will go from 0 to 10.
This variable counter will be incremented by 1 every time we go through the loop.
We remain in the loop as long as the variable counter is less than or equal to 10.
The test for the condition, in this case, is performed at the start of the loop.

Loops are very useful for performing repetitive actions for a finite number of objects, or to perform what is usually referred to as recursive actions. For example, you could use loops to create (i.e., instantiate) 100 objects at different locations (this will save you some code!), or to go through an array of 100 (or more) elements.

OceanofPDF.com

Functions

Functions in Lua can be compared to a friend or a colleague to whom you gently ask to perform a task based on specific instructions and return the information to you afterward. For example, you could ask your friend the following: "Can you please tell me when I will be celebrating my 20th birthday given that I was born in 2000?" You provide your friend (who is good at math!) with the information (date of birth), and they calculate the year of your 20th birthday and give this information back to you. In other words, your friend receives an input (i.e., the date of birth) and returns an output (i.e., the year of your 20th birthday). Functions in programming work exactly this way: they are given information (sometimes not), perform an action, and then (if needed) return information.

In programming terms, a function is a block of instructions that performs a set of actions. A function is executed when invoked (or simply called) or when an event occurs (e.g., the player clicks a button or collides with an object; we will see more about events in the next section). Functions are declared, and they can also be called when needed.

Functions are very useful because once the code for a function has been created, it can be called multiple times without the need to re-

write the same code over and over again. Also, because functions can take parameters, a function can process these parameters and produce or return information accordingly. In other words, they can perform different actions and produce different results based on the input. As a result, functions can do one or all of the following:

Take parameters and process them.

Perform an action.
Return a result.

A function in Lua has a syntax and can be declared as follows:

```
function nameOfTheFunction()

    -- Perform actions here...

end
```

In the previous code, the function does not take any input, nor does it return an output. It simply performs actions.

OR

```
function nameOfTheFunction(parameter1)

    -- Perform actions here...

    return result

end
```

In the previous code snippet, the function takes one parameter and then performs actions and returns a result.

Let’s look at the following function, for instance:

```
function calculateSum(a, b)

    return a + b

end
```

In the previous code:

The function is defined using the function keyword.
The function will return a number (the sum of two numbers).
The name of the function is
The function takes two arguments (i.e., parameters a and

The function returns the sum of the two parameters, which are referred to as a and b within this function.

A function can be called using the () operator, as follows:

---

nameOfTheFunction1()

nameOfTheFunction2(value)

local test = nameOfTheFunction3(value)

In the previous code:

A function is called with no parameter

A function is called with a parameter

In the third example, a variable called test will be set with the value returned by the function

OceanofPDF.com

Tables

In Lua, tables are versatile and can be used to store collections of values, making it easier to manage data without needing to declare multiple variables for similar purposes.

As we will see in the next section, tables (which can function like arrays) can help reduce the number of variables needed when storing data of the same type and can make accessing this data much simpler.

OceanofPDF.com

Single-Dimensional Tables (Arrays)

Let’s look at the simplest form of a table: a single-dimensional table, which is similar to a one-dimensional array in other programming languages. For example, let's say we have a group of 10 people, each with a name. If we wanted to store this information using separate variables, we would need to declare (and set) ten different variables:

---

local name1 = "Paul"

local name2 = "Mary"

-- ... and so on

While this code works, it would be much more efficient to store these names in a single table. A table is comparable to a list of elements that we access using an index. In Lua, the index typically starts at 1.

Here’s how we could do this using a table:

```
local names = {"Paul", "Mary", "John", "Mark", "Eva", "Pat",
"Sinead", "Elma", "Flaithri", "Eleanor"}
```

In the previous code, we have declared a table called names that includes 10 string values. Each element in the table can be accessed using its index, as shown in the next snippet:

```
print(names[1])  -- Outputs "Paul"

print(names[2])  -- Outputs "Mary"

print(names[10]) -- Outputs "Eleanor"
```

Remember, in Lua, the index of the first element in a table is and the index of the last element is the total number of elements in the table. For a table of size the index of the first element is and the index of the last element is

OceanofPDF.com

Multidimensional Tables

Now that we've covered single-dimensional tables, let's explore multidimensional tables, which can also be very useful for storing more complex data. A multidimensional table can be thought of as a table containing other tables, much like a grid or a matrix.

For instance, imagine a building with several floors, and on each floor, several apartments. We might want to store the number of tenants in each apartment. One approach would be to create separate variables for each apartment, like so:

```
local ap0_1 = 0  -- First apartment on the ground floor

local ap0_2 = 0  -- Second apartment on the ground floor

-- And so on...
```

But a more efficient way to manage this data is by using a multidimensional table:

```
local apArray = {}

for i = 1, 10 do

    apArray[i] = {}

    for j = 1, 10 do

    apArray[i][j] = 0

    end

end
```

In the previous code:

We declare apArray as a table.
We then initialize this table to be a 10x10 grid (10 floors and 10 apartments per floor).
Each element in this table represents an apartment, initialized to 0 tenants.

You can access or modify the elements in this multidimensional table as follows:

---

apArray[1][1] = 2 -- Sets the number of tenants in the first apartment on the first floor to 2

print(apArray[1][1]) -- Outputs the number of tenants in that apartment

---

OceanofPDF.com

Using Loops with Tables

One of the powerful features of tables in Lua is that you can use loops to efficiently access or modify all elements in the table. For example, you could iterate over all apartments in the building to reset the tenant count:

```
for i = 1, 10 do

    for j = 1, 10 do

    apArray[i][j] = 0

    end

end
```

This loop iterates through every floor and apartment, setting the number of tenants to 0. Using tables and loops together allows you to write more efficient and concise code.

OceanofPDF.com

Scope of Variables

In Lua, as in other programming languages, the scope of a variable determines where that variable can be accessed and modified within your code. Understanding variable scope is important for writing clean and error-free scripts, especially in Roblox development.

Global Scope: A variable with global scope can be accessed from any part of your script after it is declared. In Lua, a variable is global by default if it is declared without the local keyword. This means it can be accessed from anywhere in the script, which can be useful but also potentially risky if not managed carefully.

For example, let’s look at this code:

```
myGlobalVar = 10  -- Global variable

function printGlobalVar()

    print(myGlobalVar)
```

```
end

printGlobalVar()  -- This will print 10, because myGlobalVar is accessible globally
```

In the example above, myGlobalVar is a global variable, and it can be accessed inside the function printGlobalVar as well as anywhere else in the script.

Local Scope: A variable with local scope, on the other hand, can only be accessed within the block of code where it is declared. In Lua, you create a local variable by using the local keyword. This is useful for controlling where a variable can be used and preventing it from interfering with other parts of your script.

For example, let’s look at the following code:

```
local myLocalVar = 20  -- Local variable

function printLocalVar()

    local anotherLocalVar = 30
```

```
    print(myLocalVar)  -- This will print 20

    print(anotherLocalVar)  -- This will print 30

end

printLocalVar()

print(anotherLocalVar)  -- This will cause an error because
anotherLocalVar is not in scope
```

In this example, myLocalVar is accessible inside the function printLocalVar because it was declared in a broader scope. However, anotherLocalVar is only accessible within the printLocalVar function. Trying to access anotherLocalVar outside of its block of code will result in an error.

Function Scope

Variables can also have a scope limited to a function or a specific block within a function. Any variable declared inside a function using the local keyword will only be accessible within that function. This helps encapsulate functionality (i.e., to keep the code self-contained

within the function) and avoid unintended interactions with variables outside the function.

For example, let’s look at the following code:

```
function calculateSum(a, b)

    local sum = a + b  -- sum is local to this function

    return sum



end

local result = calculateSum(5, 10)

print(result)  -- This will print 15

print(sum)  -- This will cause an error because sum is not accessible outside the function
```

In the previous code, the variable sum is local to the calculateSum function and cannot be accessed from outside the function. This ensures that the variable sum does not accidentally interfere with other parts of your script.

As you can see, using the proper scope for a variable is important when you write your scripts in Lua, and you will find below a set of best practices that will hep you with variable Scope for Lua/Roblox:

Use Local Variables When Possible: Local variables are preferred in Lua because they help manage memory usage and prevent unintentional interference between different parts of your script.
Limit the Use of Global Variables: Use global variables sparingly, especially in large projects, as they can be accessed and modified from anywhere in your script, making it harder to track and debug issues.
Encapsulate Functionality: Keep variables within the scope they are needed. If a variable is only used inside a function, declare it locally within that function.

Avoid Overlapping Variable Names: Ensure that variable names in different scopes do not overlap to avoid confusion and potential bugs.

Understanding and effectively using variable scope in Lua is key to writing efficient, maintainable, and bug-free code, particularly in complex Roblox games.

OceanofPDF.com

Events

In Lua, especially within the context of Roblox, events are essential for responding to user interactions and game occurrences. An event in Roblox Lua is a trigger that executes a specific function when a certain condition is met. For example, when a player touches a part, clicks a button, or when a property changes, an event is fired, and you can connect this event to a function that handles the event.

Understanding Events

Events in Lua are similar to notifications. When an event occurs, such as a player touching a part or a button being clicked, a function is called to handle that event. This function, often referred to as an event handler, contains the instructions that should be executed when the event occurs.

For example, consider an event that is triggered when a part in Roblox is touched. You can connect this event to a function that prints a message:

```
local part = script.Parent  -- Assume this script is a child of a part
```

```
local function onPartTouched(otherPart)
    print(otherPart.Name .. " touched the part!")
end

part.Touched:Connect(onPartTouched)
```

In this example:

We get a reference to the part using
We define a function onPartTouched that takes otherPart as a parameter, representing the object that touched the part.

We connect the Touched event of the part to the onPartTouched function using

Whenever another part touches the specified part, the function is executed, and a message is printed to the output.

Common Events in Roblox

Here are some commonly used events in Roblox Lua scripting:

Touched This event is fired when a part is touched by another part, often used for triggers such as when a player walks over a certain area.

```
local function onTouched(hit)

    print("A part has been touched by: " .. hit.Name)

end

local part = script.Parent

part.Touched:Connect(onTouched)
```

MouseClick This event is triggered when a player clicks on a button (as part of the user interface).

```
local button = script.Parent

local function onClick()
```

```
    print("Button clicked!")

end

button.MouseButton1Click:Connect(onClick)
```

Changed This event is fired whenever a property of an object changes. It is useful when you need to react to changes in an object's properties.

```
local part = script.Parent

local function onPropertyChanged(property)

    print(property .. " has changed!")

end

part.Changed:Connect(onPropertyChanged)
```

PlayerAdded This event is fired when a new player joins the game, useful for initializing player data or welcoming players.

```
local Players = game:GetService("Players")

local function onPlayerAdded(player)

    print(player.Name .. " has joined the game.")

end

Players.PlayerAdded:Connect(onPlayerAdded)
```

Creating Custom Events

In Roblox Lua, you can also create custom events using the BindableEvent object. This allows you to create your own events that other scripts can respond to. This is illustrated in the next snippet.

```
local bindable = Instance.new("BindableEvent")
```

```
local function customEventHandler()
    print("Custom event triggered!")
end

bindable.Event:Connect(customEventHandler)

-- Trigger the event manually
bindable:Fire()
```

In this example:

We create a BindableEvent object.
We connect a custom event handler function to this event.
We manually trigger the event using which calls the connected function.

OceanofPDF.com

Practical Example of Events: Opening a Door

Let’s combine these concepts with a practical example where touching a part opens a door:

```
local door = workspace.Door

local trigger = workspace.Trigger

local function onTriggerTouched(otherPart)

    if otherPart.Parent:FindFirstChild("Humanoid") then  -- Check if a player touched it

    door.Transparency = 0.5

    door.CanCollide = false

    print("Door is now open")

    end
```

```
end

trigger.Touched:Connect(onTriggerTouched)
```

In this example:

We monitor a trigger part using the Touched
If a player touches the trigger (identified by checking if the parent has a Humanoid), we make the door transparent and disable its collision, effectively "opening" the door.

As you could see in this section, events in Roblox and Lua are important for creating interactive elements in your game. By connecting events to functions, you can respond to player actions, such as clicks, touches, or even changes in object properties. Understanding and utilizing events allows you to build dynamic and responsive experiences in your games.

OceanofPDF.com

Summary

Congratulations! By now, you should have a basic understanding of Lua syntax, including variables, functions, loops, and conditional statements. These are the building blocks of Lua programming and mastering them will allow you to write efficient and effective scripts.

In the next chapter, we will build on this knowledge and start applying these concepts to create and add more interactivity in your Roblox games.

OceanofPDF.com

Quiz

Please guess whether the following statements are True or False (the solution is in the section Answers to at the end of the book).

In Lua, the = operator is used for assigning values to variables.
Lua tables always start their indexing at 0 by default.
You must declare the type of a variable when using Lua, just like in C#.
In Lua, the local keyword limits a variable's scope to the block where it is declared.
Lua does not support multidimensional tables, so you cannot create grids or matrices.
The Touched event in Roblox is fired when a player interacts with a GUI element.
In Roblox Lua, the Connect method is used to associate an event with a function.
Lua requires semicolons at the end of each statement, like C#.
In Lua, nil represents the absence of a value, similar to null in other languages.
In Roblox Lua, you can use the BindableEvent object to create custom events that other scripts can respond to.

OceanofPDF.com

Challenge

Challenge 1: Documentation Scavenger Hunt

Visit the official Roblox Developer Hub or the Lua reference manual and find detailed information on the following topics:

How to use the Changed event in Roblox.
The purpose and usage of the local keyword in Lua.

Summarize your findings in a few sentences for each topic. This challenge encourages you to become familiar with navigating and using official documentation, which is a critical skill for any programmer.

Challenge 2: Concept Mapping

Create a concept map that visually organizes the key ideas from the chapter. Include concepts such as variables, tables, events, and their relationships in Lua and Roblox. Try to show how these concepts connect and how they are used in programming. This exercise will

help reinforce your understanding of how different elements of Lua scripting interact and depend on each other.

OceanofPDF.com

5

Introduction to Scripting in Roblox Studio

In this chapter, we will introduce you to scripting in Roblox Studio, a crucial skill for adding interactivity and functionality to your games. We'll start with the basics of Lua, the scripting language used in Roblox and guide you through creating your first script. By the end of this chapter, you'll be able to write simple scripts to control game objects and create interactive experiences.

After completing this chapter, you will be able to:

Understand the basics of Lua scripting.
Create and run your first script in Roblox Studio.
Use variables to store and manipulate data.
Implement basic operators and expressions.
Write simple functions and use them in your game.
Handle conditional statements for decision-making.
Use loops to automate repetitive tasks.
Debug scripts effectively using the Output window.

OceanofPDF.com

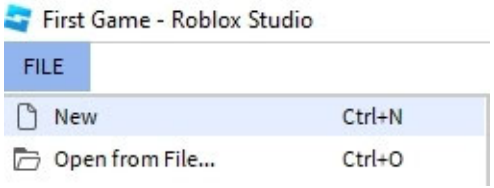
Starting to code in Lua

In this section, we will introduce you to Lua, the scripting language used in Roblox Studio. Lua is a lightweight, easy-to-learn programming language that is perfect for beginners. It allows you to add interactivity to your games by controlling how objects behave and interact with each other.

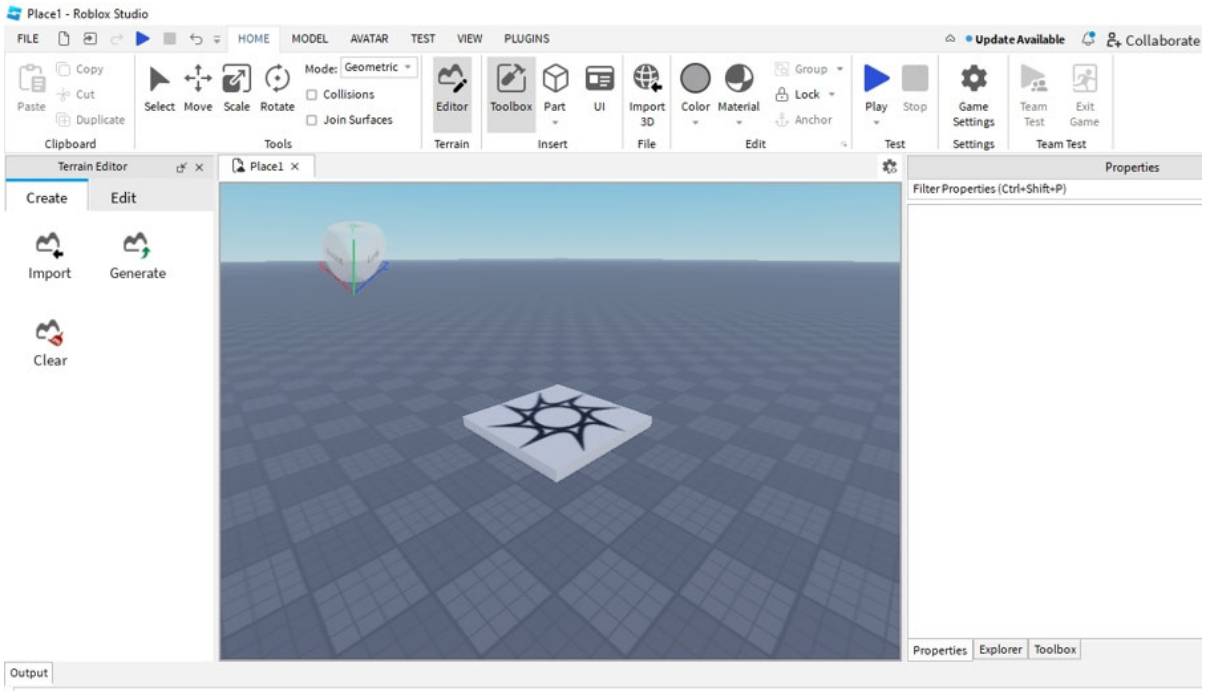
In the next steps, we will write a simple Lua script to change the color of a part when a player touches it. This will help you understand the basics of Lua scripting and how it can be used in Roblox Studio.

Please do the following:

Open Roblox Studio and select or create a new project (e.g., BasePlate): Select File | New from the top menu.



This will create a new project.

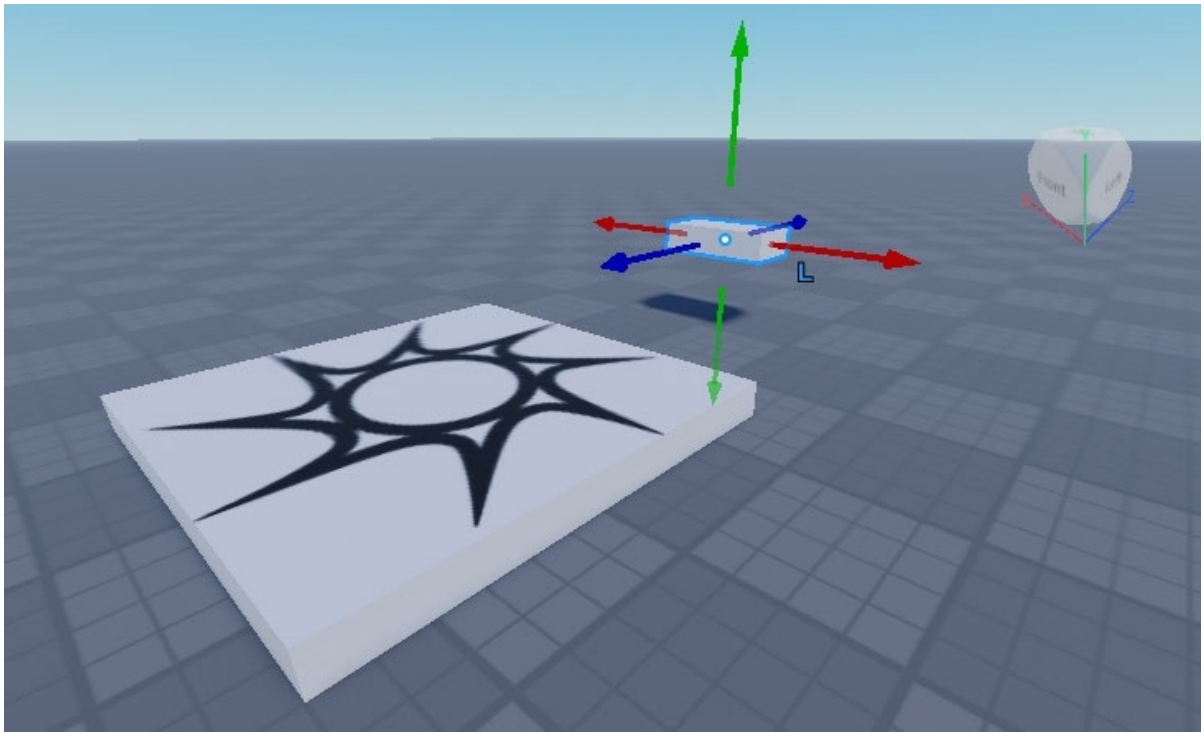


Now that the new project is open, we can start to add several parts and linked scripts.

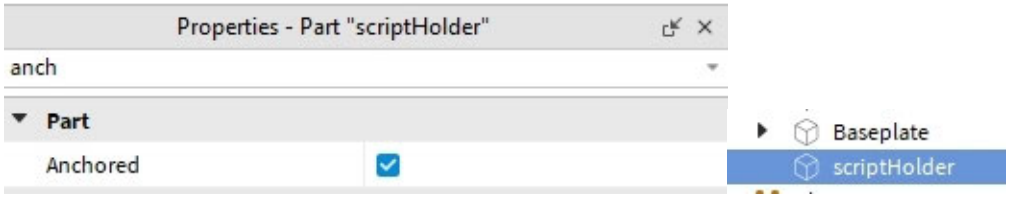
Create a Part in the workspace by clicking on the Home tab and selecting

Place the part where you want it in the workspace and rename it

Just make sure it is above the ground (i.e., not touching anything).

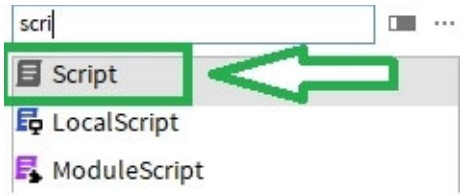


Using the Properties window, make sure that the option is set to True for that object.



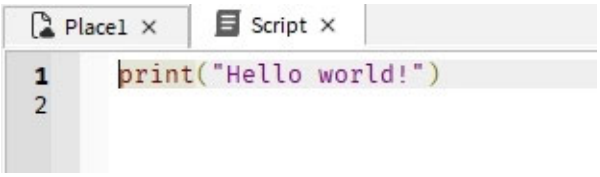
Using the Explorer panel, right-click on the part that you have just created (i.e.,

From the contextual menu select Insert Object | This will create a new script attached to the object



Now that we have inserted a script for that object, we will edit it and add some code to it.

In the Script you will see the default code print (“Hello world”) as per the next figure.



Please delete that default code and type the following script instead:

```
function onTouched(part)

= BrickColor.new("Bright red")

end

script.Parent.Touched:Connect(onTouched)
```

In the previous code:

We create a function called this function takes one parameter that will be called part within that function, and it refers to the part to which the script is attached.
Within the function, we change the attribute called BrickColor of the part (which is the parent of the script) to Bright
We then ensure, outside the function, that the function onTouched is called whenever the parent of the script (i.e., the part) is touched.

So effectively, this script will change the part’s color to red when a player touches it.

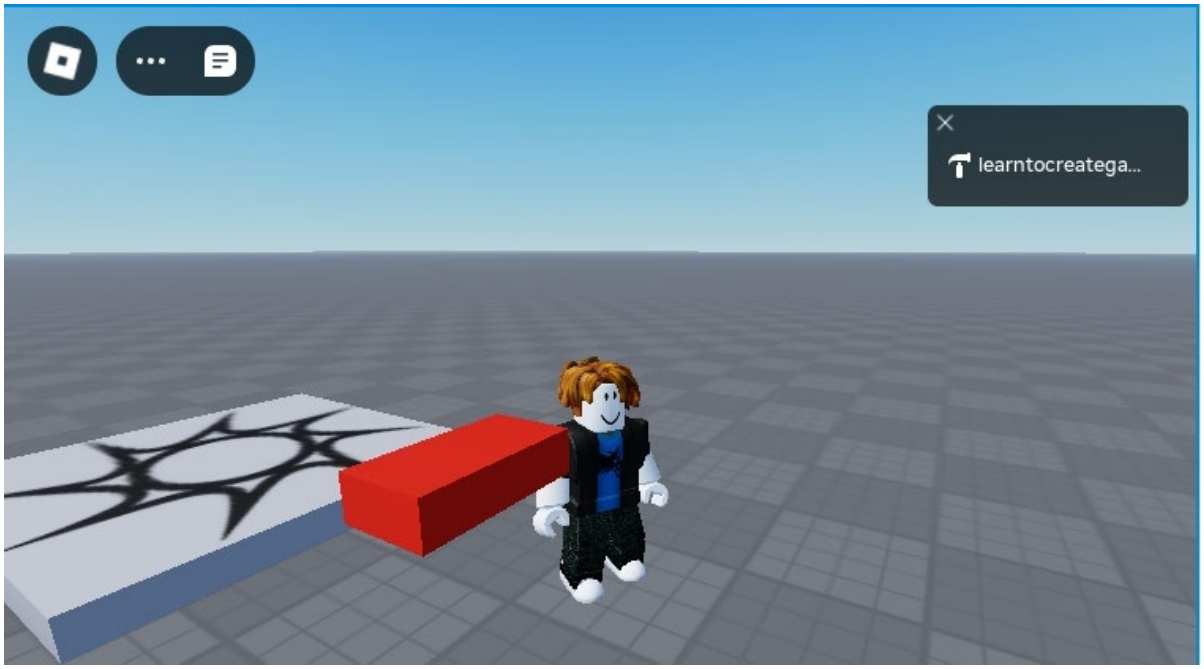
We can now test the script:

Click on the Play button at the top of Roblox Studio to run the game.

You should see that, at the start, the part is grey.



Move your player character so that it touches the part. After touching the part, you should see that its color changes to red.



You can also look at the Output window, and check that a message similar to the one in the next figure has been displayed.



Now that we've written and tested our first script, you can see how Lua allows us to control the behavior of objects in the game. In the next section, we'll dive deeper into variables and how they can be used to store and manipulate data in your scripts.

OceanofPDF.com

Using Variables

Variables are essential in scripting as they allow you to store and manipulate data that your scripts can use. In Roblox, variables can store numbers, text, and even objects like parts and players.

In the next steps, we will create a script that uses variables to store the color of a part and change it when the player touches it. This will help you understand how variables work and how to use them effectively in your scripts.

To do so, we will modify the script that we have already written:

Please open the script that you have created in the previous section. Comment or delete the previous code and type the following script:

```
local originalColor = BrickColor.new("Bright green")

local touchColor = BrickColor.new("Bright red")

script.Parent.BrickColor = originalColor
```

```
function onTouched(part)

    script.Parent.BrickColor = touchColor

end

script.Parent.Touched:Connect(onTouched)
```

In the previous code, we did the following:

We declared a variable originalColor to store the part's initial color.
We then declared another variable touchColor to store the color we want to change the part to.
We used the variable to set the color of the part to green at the start of the game.

Finally we used the variable touchedColor in the onTouched function to change the part’s color when touched.

So effectively this script stores the original color of the part in a variable and changes the color to green when touched.

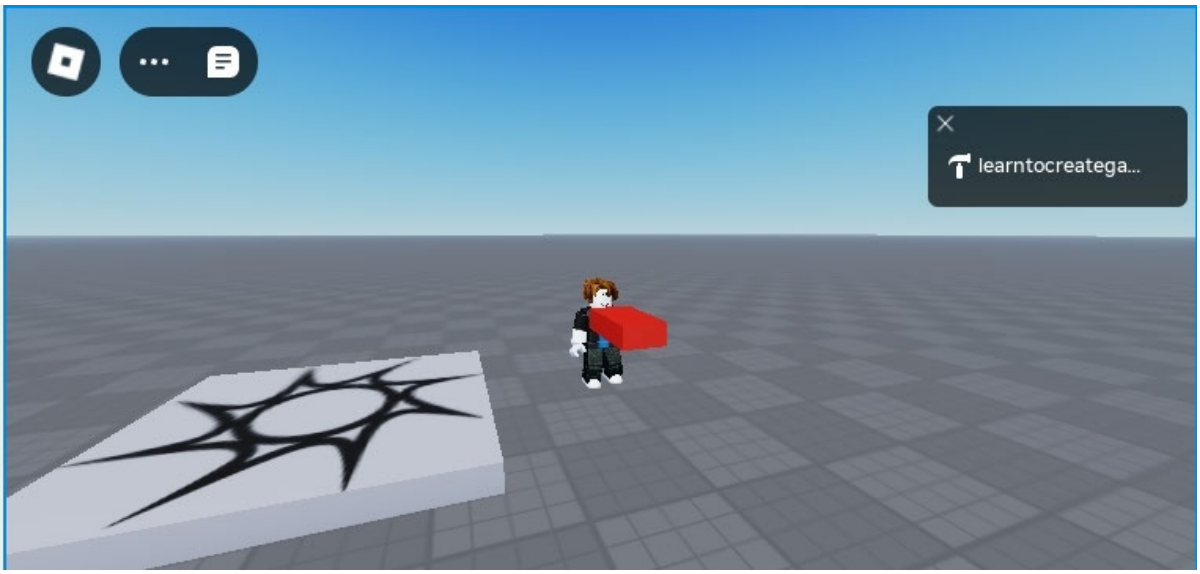
We can now test the script:

Click on Play to run the game.

You should see that the part is green at the start of the game.



Touch the part and observe that it changes to The color change is controlled by the variables that we defined earlier in the script.



You've now learned how to use variables in your scripts to store and manipulate data. In the next section, we'll explore basic operators and expressions that you can use to perform calculations and make decisions in your scripts.

OceanofPDF.com

Using Basic Operators and Expressions

Operators and expressions allow you to perform calculations and make decisions in your scripts. Roblox Studio supports a variety of operators, including arithmetic, comparison, and logical operators.

In the next steps, we will create a script that uses arithmetic operators to increase the size of a part each time it is touched. This will help you understand how to use operators and expressions in your scripts.

Please comment the previous code using --[[and

--[[local originalColor = BrickColor.new("Bright green")

local touchColor = BrickColor.new("Bright red")

script.Parent.BrickColor = originalColor

function onTouched(part)

= touchColor

end

script.Parent.Touched:Connect(onTouched)]]--

Type the following script instead:

```
local a = 2

local b = 3

local c = a + b

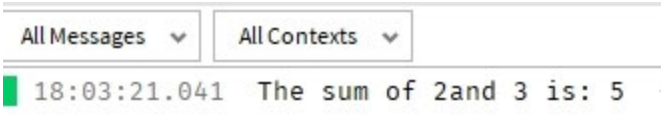
print("The sum of "..a.." and "..b.." is: "..c)
```

In the previous code:

We declare and initialise the variables b and
c is the sum of a and

We then display a message that displays the value of the variable c.

You can now play the scene and you should see a message that is similar to the one below; in the Output window.



Now that we have used additions, let’s try to use the multiplication operator:

Please change the code as follows (new code in bold):

---

a = 2

b = 3

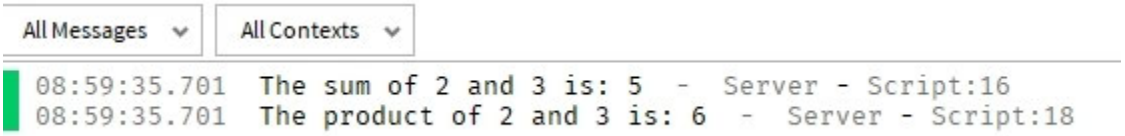
c = a+b

print("The sum of "..a.." and "..b.." is: "..c)

c = a*b;

print("The product of "..a.." and "..b.." is: "..c)

You can now play the scene and you should see a message that is similar to the one below; in the Output window.



OceanofPDF.com

Using Conditional statements

Now that we have seen how to perform simple arithmetical calculations, we will see how we can use operators to modify the attributes of the objects that we have created (e.g., position, seize, etc).

Please comment the previous code.
Add the following code to your script.

```
local a = 2

local b = 3

if (a > b) then

    print(a .. " is greater than " .. b)

else

    print(b .. " is greater than " .. a)

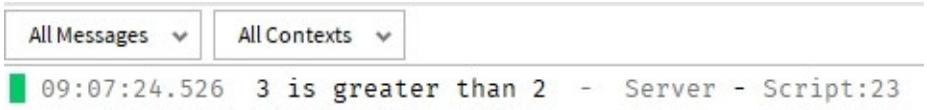
end
```

In the previous code, we do the following:

Declare variables a and b and set their initial values.
Create two conditional statements.
Provided that a is greater than b, the first statements will display that the value of a is greater than the value of b.
Provided that a is not greater than b, the second statement will display that the value of b is greater than the value of b.

The message displayed will depend on the value of a and b; so the statements used to display these messages will be executed based on conditions (a > b or b > a), hence the name: conditional statements.

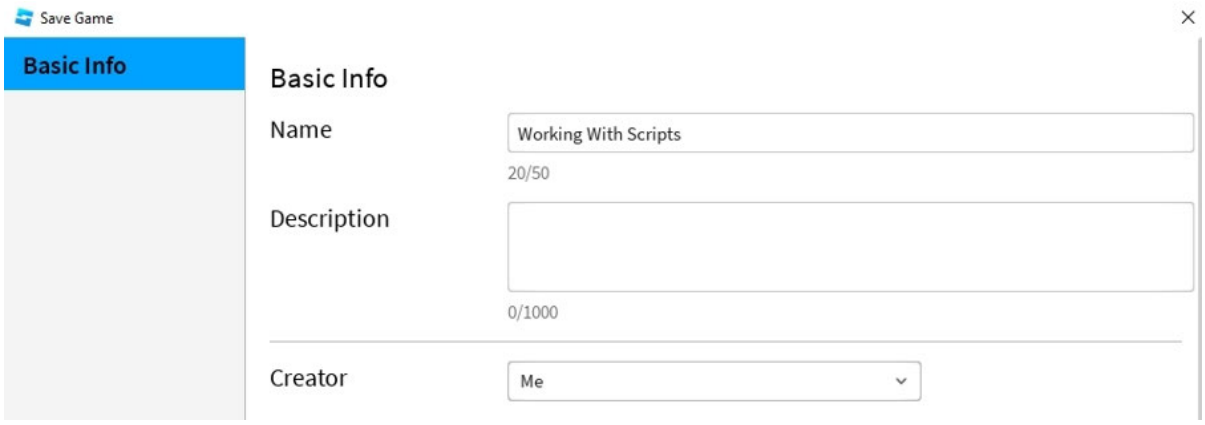
You can play the scene and you should see, in the Output window, that, because b is greater than the message is is greater than



Before we go ahead with the next steps, please save your project/code:

Select: File | Save to as you have done in previous sections.

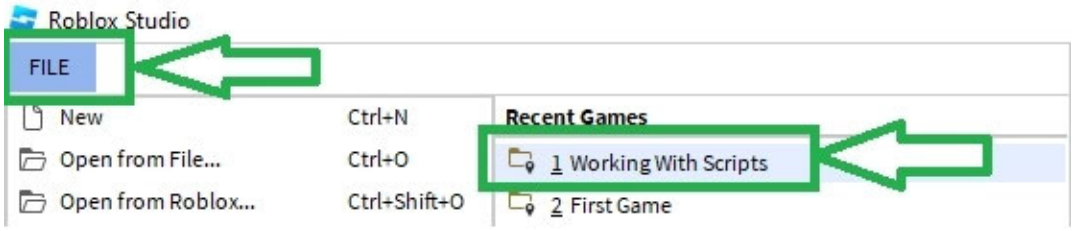
In the new window, give a name to your project, for example Working With



Leave the rest of the information as default.
Press Save.



If the project has closed, you can reopen it by selecting: File | Working with



OceanofPDF.com

Combining events and conditional statements

Now that you know how to use and combine conditional statements, we can combine these with events to design a behaviour whereby, upon collision between the player character and a block, and if the block has not yet been touched by the player character, this block's size will be modified. So here we use an event and a condition (i.e., no prior contact).

So, let's see the code that will make it possible to implement this behavior:

Please comment the previous code (in the script attached to the object and add this code:

```
local hasBeenTouched = false

function onTouched(part)

    local width = script.Parent.Size.x;

    local depth = script.Parent.Size.z;
```

```
    local height = script.Parent.Size.y;

    if (hasBeenTouched == false) then

    width = width+10;

    depth = depth + 10

    script.Parent.Size = script.Parent.Size + Vector3.new(width, height,
depth)

    script.Parent.Position = script.Parent.Position + Vector3.new(3,3,3)

    hasBeenTouched = true;

    end

end

script.Parent.Touched:Connect(onTouched)
```

In the previous code, we did the following:

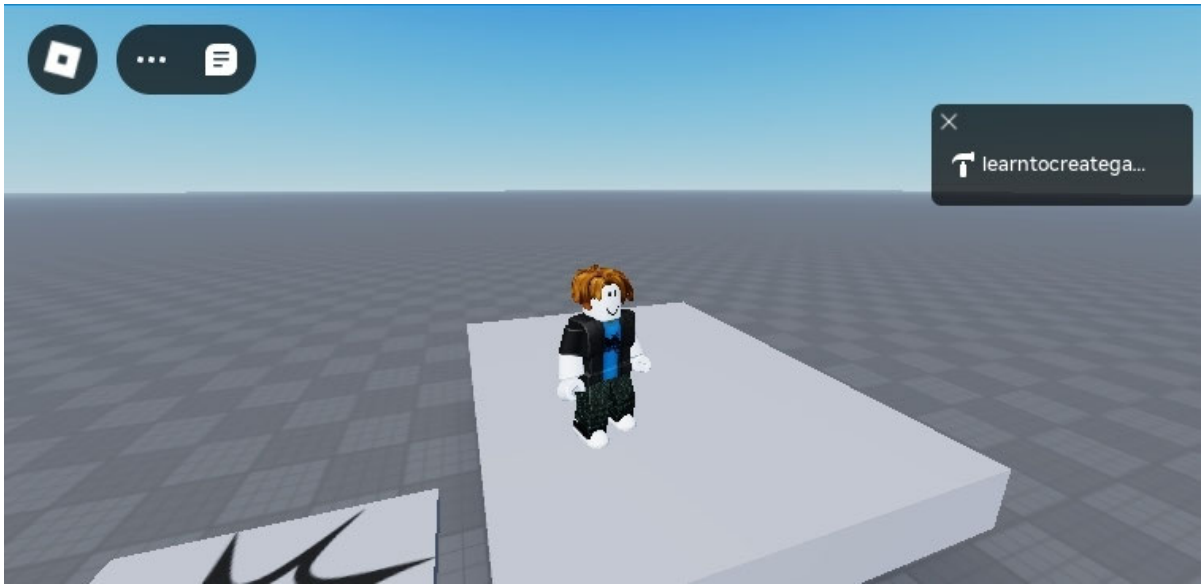
Declared a variable to track whether the part has been touched before.
Defined a function that executes when the part is touched.
Retrieved the current and height of the part and stored them in variables.
Used an if statement to check if the part had not been touched before, ensuring the code inside runs only once.
Increased the width and depth of the part by 10 units each.
Resized the part using the updated width, depth, and existing height values.
Moved the part's position by 3 units along each axis (x, y, and z).
Set the hasBeenTouched variable to true to prevent the code from running again.
Connected the Touched event of the part to the onTouched function, allowing the resizing and repositioning to happen when the part is touched.

You can now test the game:

Press you should see that the part is grey with its default size



As you jump on (or touch) the part, you should see that its size is modified and that it is higher also. This is because our code has not only moved the box up but also changed its size.



You've now learned how to use basic operators and expressions to manipulate objects in your scripts. In the next section, we'll introduce

functions and events, which are key to making your scripts more dynamic and interactive.

OceanofPDF.com

Using Arrays and Loops (For, While)

As you have seen in the previous chapter, loops are a fundamental concept in programming that allow you to repeat a block of code multiple times. In Lua scripting, you can use for loops, while loops, and repeat loops to perform repetitive tasks efficiently.

In the next steps, we will create a script that continuously changes the color of a part using a loop, cycling through different colors. This will help you understand how to use loops in your scripts.

Please do the following:

Comment the previous code.
Type the following script:

```
local colors = {BrickColor.new("Bright red"), BrickColor.new("Bright green"), BrickColor.new("Bright blue")}

while true do

    for i = 1, #colors do
```

```
    script.Parent.BrickColor = colors[i]

    wait(1)

    end

end
```

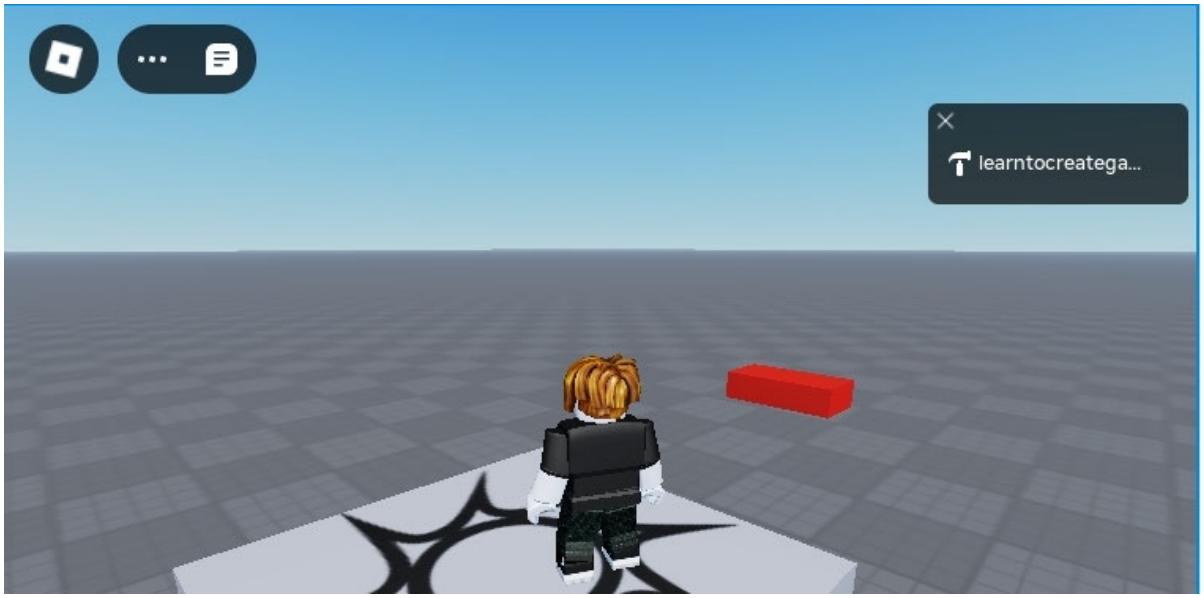
In the previous code:

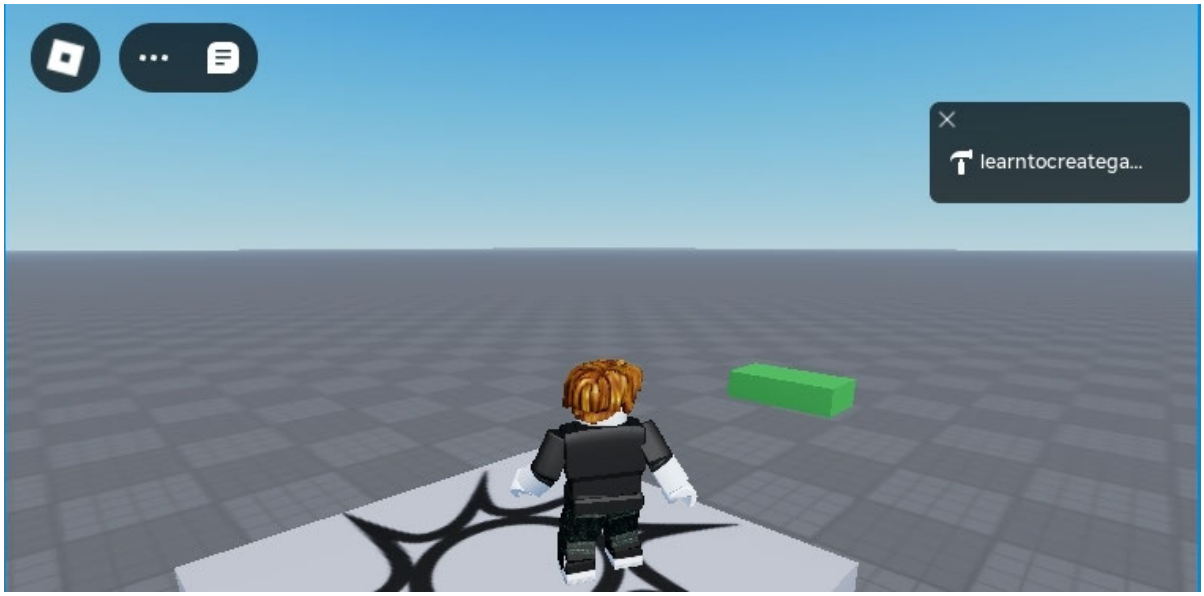
We define three colors and store them in the array called
We then create a while loop that continues as long as the condition remains
Within that loop we create a for loop that goes through all the colors included in the array called

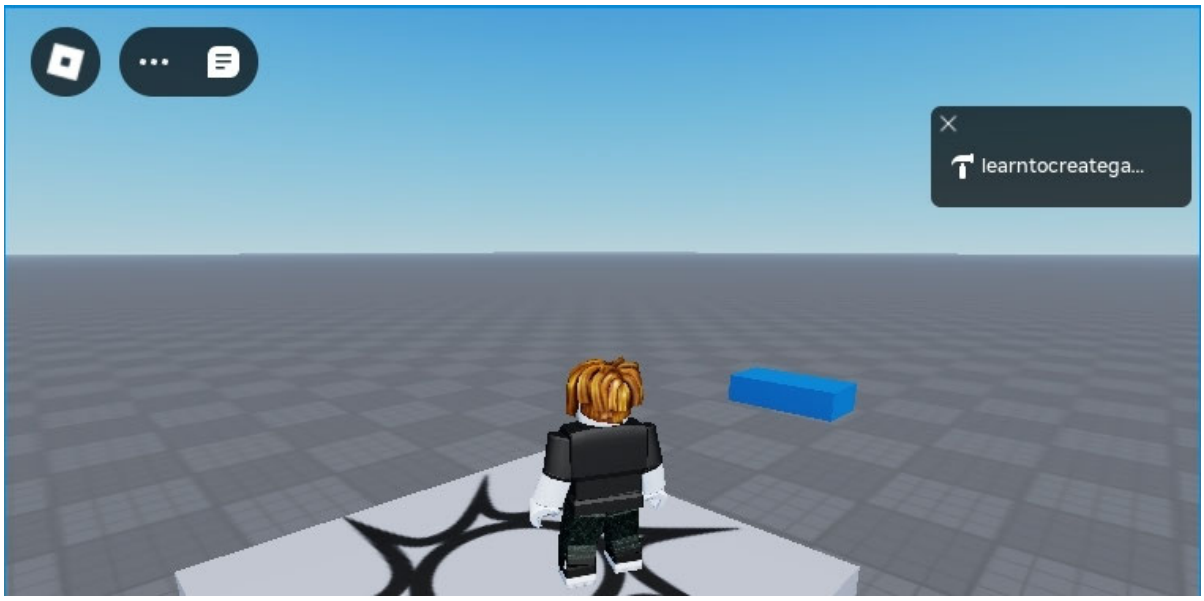
At each iteration, the color of the object linked to that script is changed to the corresponding color.

So effectively, this script cycles the part's color through red, green, and blue every second.

You can now save the script, and play the game, and you should see that the color of the blocks, cyclically changes to green, red and blue.


learntocreatega...


learntocreatega...



You've now learned how to use loops to automate repetitive tasks in your scripts. In the next section, we'll explore debugging techniques, helping you identify and fix errors in your scripts.

OceanofPDF.com

Debugging Scripts with the Output Window

Debugging is an essential skill in scripting, allowing you to identify and fix errors in your code. Roblox Studio provides several tools to help you debug your scripts, including the Output window and breakpoints.

In the next steps, we will deliberately create an error in a script and then use the Output window to identify and fix the issue. This will demonstrate how to effectively debug your scripts in Roblox Studio.

Please do the following:

In the Script Editor , comment the code that you have added so far.
Type the following script:

```
function onTouched(part)

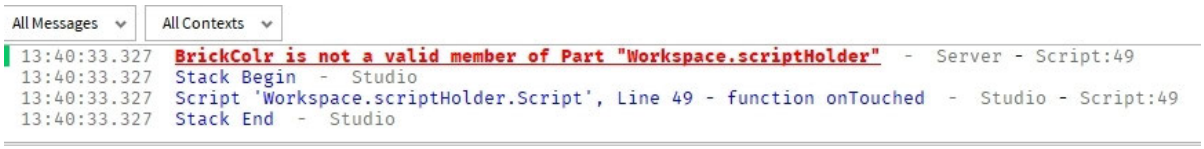
    script.Parent.BrickColr = BrickColor.new("Bright red") -- Intentional typo

end
```

script.Parent.Touched:Connect(onTouched)

As you may already have noticed, this script has an error: BrickColr should read

Make sure that the Output window is visible: Select View |
Click on Play to run the game:
Touch the part and observe the Output window. You should see an error message indicating that BrickColr is not a valid property.



If you look closer at the error message, you will notice that Roblox is indicating where this error is (i.e., the line where the error occurred).
Stop the game, go back to the Script and correct the typo (i.e., change BrickColr to
Run the script again to ensure that it now works correctly (i.e., no error message in the Output window).

You've now learned how to use the Output window to identify and fix errors in your scripts. With these debugging skills, you can ensure that your scripts run smoothly. In the next section, we'll delve into more advanced topics, like creating a simple game mechanic.

OceanofPDF.com

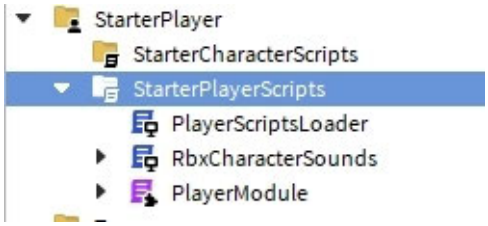
Basic User Input Handling

Handling user input allows you to create interactive and responsive games. In this section, we will explore how to handle keyboard and mouse input in Roblox scripting.

In the next steps, we will create a script that displays a message and ultimately moves a part in response to arrow key presses from your keyboard. This will demonstrate how to handle basic user input.

While we have in the past attached our scripts to objects, we will locate our script differently this time. The script that we are about to create will be used for the player, and any player joining this game, and because this script involves user input detection, it will have to be local (i.e., a local script), a type of script used for player events only.

So, with this in mind, please open the Explorer window and scroll down to the section



Right-click on the item labelled and select the option then choose this will add a local script to the object called
You can rename it as per the next figure.



Once the script is open in the Script Editor, you can remove the code that’s already present in that script, and add the following code instead:

```
local UserInputService = game:GetService("UserInputService")

function onKeyPress(input)

input.KeyCode == Enum.KeyCode.O then

("You pressed the key O")

input.KeyCode == Enum.KeyCode.P then

("You pressed the key P")



end

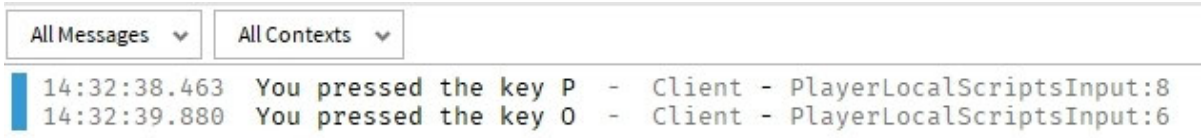
UserInputService.  InputBegan:Connect(onKeyPress)
```

In the previous code:

We declare a variable UserInputService that will be used to detect user interaction; this variable is linked to the servcie called UserInputService a service in Roblox that allows you to detect and respond to various user inputs, such as keyboard and mouse events.
We then define a function called that, when it is called, will check whether the keys O or P have been pressed.
If the key O is pressed, then the message “You pressed the key is displayed.

If the key P is pressed, then the message “You pressed the key is displayed.
We then link the event InputBegan to this function so that anytime an input begins (e.g., a key is pressed) we call the function

You can now save your code and press Play to run the game, then press the letters A or B on your keyboard, and check that a corresponding message has been displayed in the Output window.



You’ve now learned how to handle basic user input in your scripts. In the next section, we’ll explore Roblox services, which are key to managing

different aspects of your game.

OceanofPDF.com

Introduction to Services

As you have noticed in the previous section, we used a service (i.e., to be able to detect the user inputs (i.e., when a key had been pressed). And as you will discover through this book, there are many other services in Roblox, that you can use to unlock specific features in your game. This is because services in Roblox are essential components of the Roblox game development environment, as they provide a wide range of pre-built functionalities that streamline the process of creating and managing games. These services handle many behind-the-scenes tasks, such as player input, data storage, and game physics, allowing developers to focus more on the creative aspects of game development rather than building these systems from scratch. By utilizing these services, you can easily implement complex features and ensure that your game operates smoothly across various devices.

Some of the key services that you will probably use in your games include the following:

This service, as we have seen in the previous section, is used to detect and respond to user inputs, such as keyboard presses, mouse clicks, and touch events. It is often use for handling player controls, detecting

when specific keys or buttons are pressed, and managing different input types across devices.

This service makes it possible for developers to save and retrieve player data, such as progress, currency, and inventory, across different gaming sessions. It is often used for storing player achievements, saving in-game purchases, and managing leaderboards.
This service helps to manage all player-related functionalities, including tracking active players, managing player properties, and handling player events. You can use it for accessing player information, responding to player joins and leaves, and customizing player avatars.

These services are integral to building functional, interactive, and data-driven games on Roblox, allowing developers to implement robust features with ease

In the next steps, we will see how we can use some of these services for simple tasks in your game.

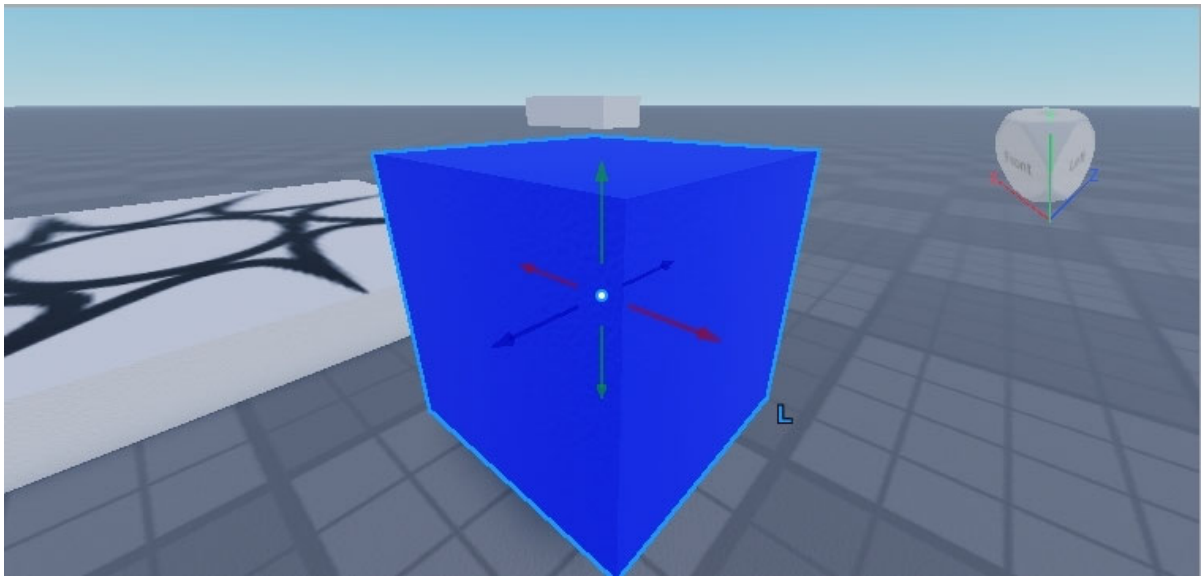
You've now learned how to use Roblox services to manage game elements and player interactions. In the next section, we'll use these skills to create a simple game mechanic involving collectible items

OceanofPDF.com

Simple Game Mechanic: Collectible Items

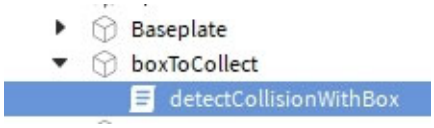
Creating game mechanics like collectible items adds interactivity and goals to your games. In this section, we will script a simple mechanic where players can collect items that disappear when touched by the player character.

Using the same project as in the previous section (i.e., the project called Working With please create a new Part (a Block) that is blue with a size of (2,2,2) and place it a few meters away from the player (i.e., Spawn). Rename this part



Next, we will add a script to that box so that it detects whenever the player touches it, and destroy this same box in that case.

Please add a new script to the object boxToCollect and rename that script



Open this script in the Script Editor and replace the existing code with the following:

---

function onTouched(part)

part.Parent:FindFirstChild("Humanoid") then

Collected")

end

script.Parent.Touched:Connect(onTouched)

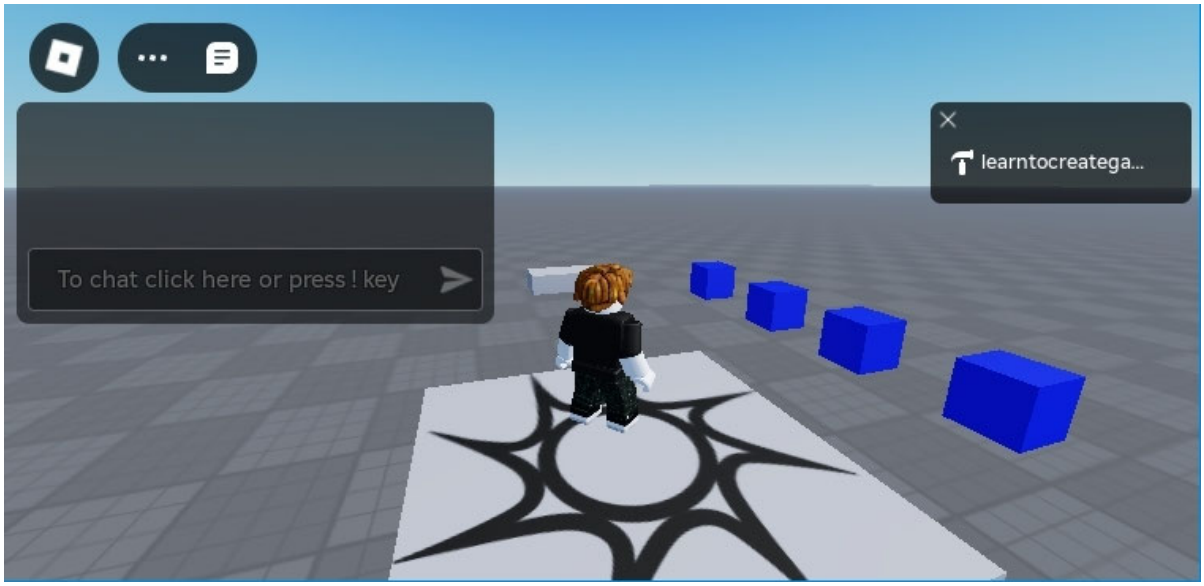
In the previous code we do the following:

We create a function called that will be called when an object touches (i.e., collides with) the box attached to that script.
When this function is called, we check that the object colliding with the box has a child called Humanoid (which would typically be the case for the player)
In that case, we display the message in the Output window.
We also destroy the box linked to the script.

You can now test your game and ensure that the blue box disappears when the player collides with it.



You can now duplicate this object boxToCollect three times (CTRL + D), move the duplicates apart, and test the scene again to ensure that you can collect all four boxes.



You've now learned how to create a simple collectible item in your game. This game feature is very common and we will reuse it on the next chapter where we will an interactive game.

OceanofPDF.com

Summary and Next Steps

In this chapter, you've learned the basics of scripting in Roblox Studio, including working with Lua, using variables, operators, functions, and events, handling user input, and managing services. You've also created simple game mechanics like interactive objects and collectible items.

So, you've made significant progress since the beginning of this book and you now have a solid foundation in scripting. In the next chapter, we'll delve deeper into more advanced scripting topics, including handling more complex game mechanics.

OceanofPDF.com

Quiz

Please guess whether the following statements are True or False (the solution is in the section Answers to at the end of the book).

Variables in Lua can store different types of data like numbers, text, and objects.
The if statement is used to create loops in Lua.
The Touched event is triggered when a part is clicked by a player.
Lua is the programming language used in Roblox Studio.
Functions in Lua allow you to group code into reusable blocks.
The Output window is used to change the color of parts.
The + operator in Lua is used for addition.
Conditional statements allow your script to make decisions based on certain conditions.
Loops in Lua are used to repeat tasks multiple times.
Debugging involves fixing errors in your script.

OceanofPDF.com

Challenge

Try these challenges to put your newly acquired scripting skills to the test:

Parkour Create a parkour obstacle where the player needs to jump over platforms and collect boxes.
Color Changing Create a platform that changes color every time it’s touched, cycling through a list of colors.
Automatic Part Create a part that continuously grows in size and then resets to its original size after reaching a certain point.

OceanofPDF.com

6
Building Your First Interactive Game in Roblox Studio

In this chapter, we will guide you through the process of creating your first interactive game in Roblox Studio. You'll learn how to design the game environment, script basic gameplay mechanics, and add interactive elements to engage your players. By the end of this chapter, you'll have a working game that you can expand on in future chapters.

After completing this chapter, you will be able to:

Design a simple game environment in Roblox Studio.
Script basic gameplay mechanics like player movement and object interactions.
Create and add interactive elements such as buttons, and collectibles.
Add sounds and lighting to enhance the game’s atmosphere.
Test your game and make improvements.

OceanofPDF.com

Designing the Game Environment

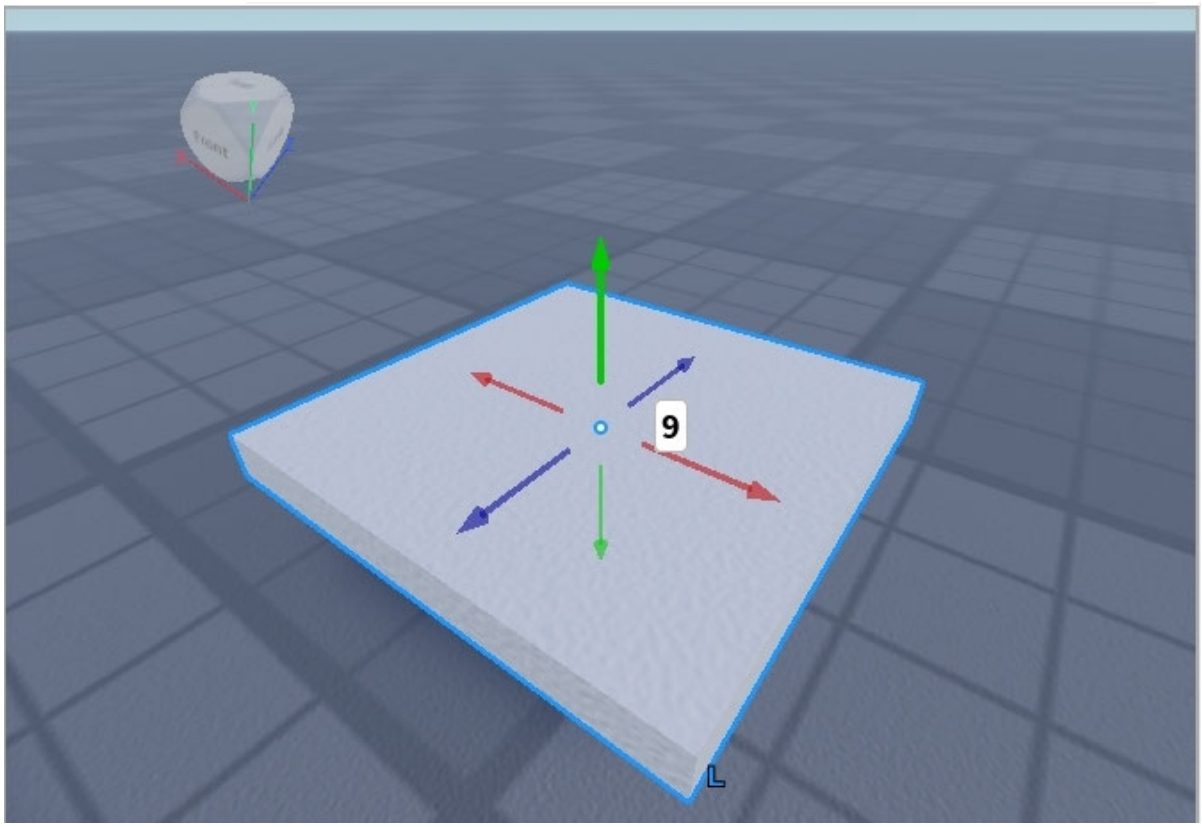
In this section, we will design a simple environment with platforms to serve as the foundation of our game.

Please do the following:

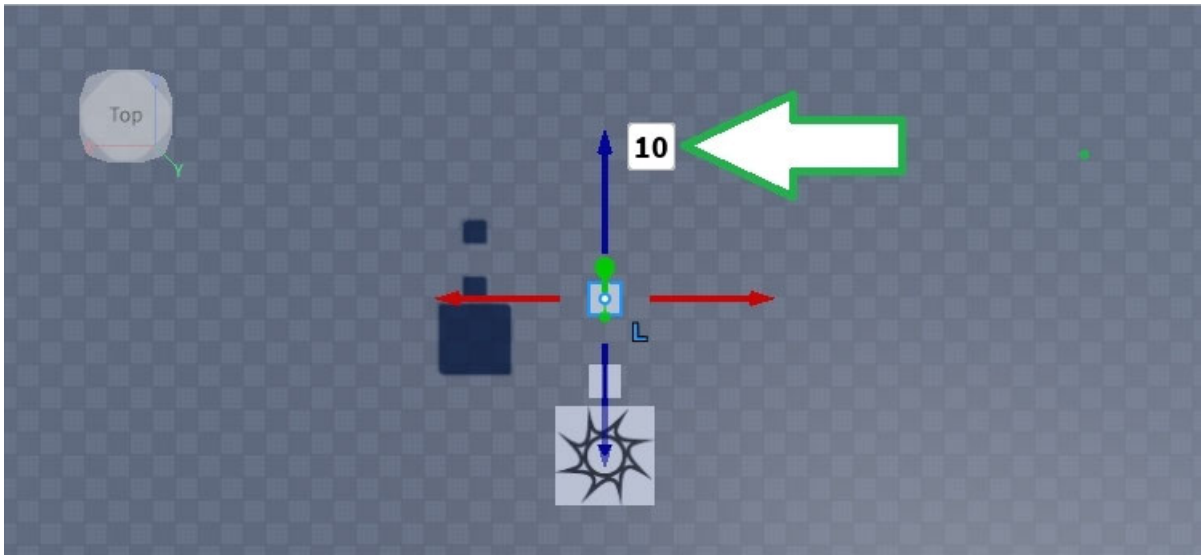
Save the current project | Save To
Close that project | Close
Create a new project |
Save that project | Save to Roblox As | Create New and name it PlatformGame or any other name of your choice.

Now that you have created a new project, we can start to add objects within:

Create Platforms: Click on the Home tab and select Part to create a basic block.
Change the size of the block to (2, 0.2, 2) using the Scale tool or the Properties window to form a platform.
Rename this object



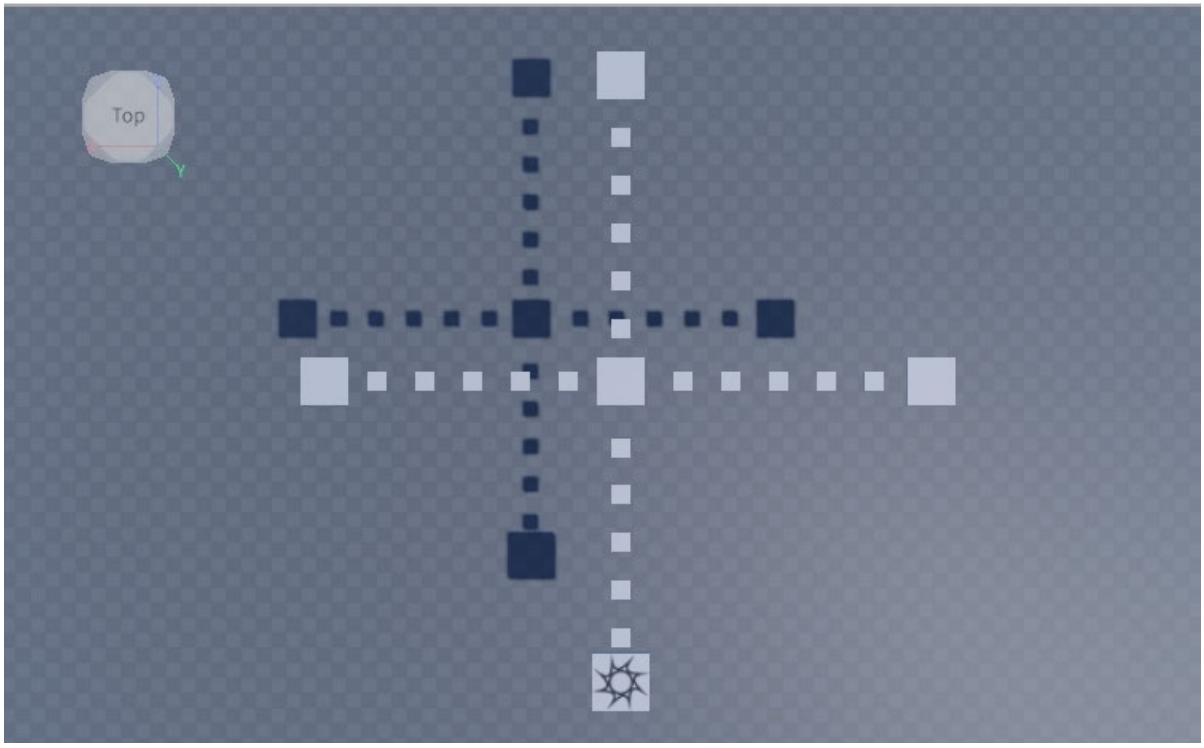
Duplicate the platform by selecting it and pressing Ctrl + D and ensure that there is approximately 10 meters between then. This can be checked, as you use the Move tool, a number that displays the distance between the two tiles will appear and increase as they are further apart, as per the next figure.



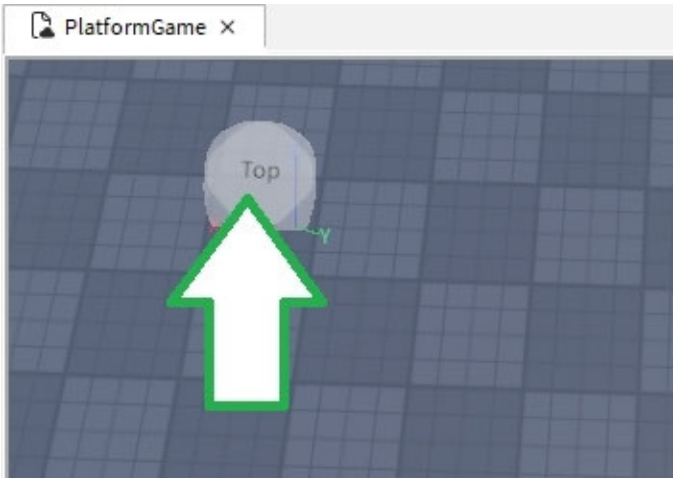
Create two more duplicates and test the scene to see if the distance between the tiles is achievable.



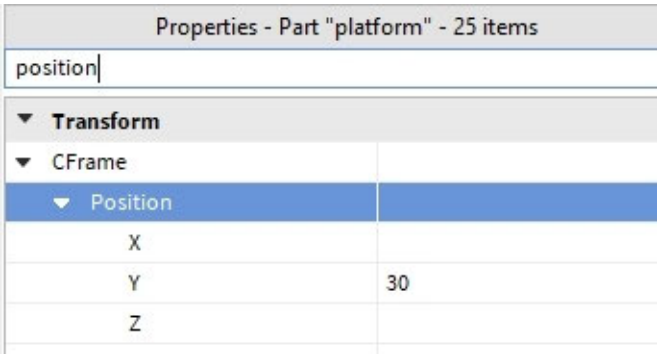
You may readjust the distance accordingly.
When this is done, you can create the full layout similar to the one illustrated in the next figure with a mix of small and big platforms.

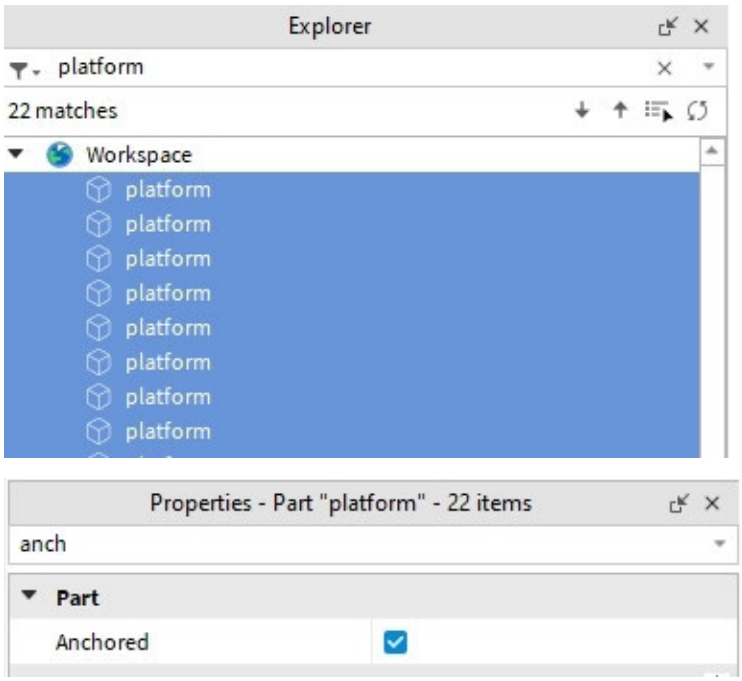


To make things easier, you can use a top view by clicking on the face labelled Top on the Gizmo in the 3D



Select all the platforms and ensure that their y attribute (for the position) is set to that they and their Anchored attribute is set to





Set the y attribute for the position of the object SpawnLocation to 30 so that the player starts at the same level as the platforms.

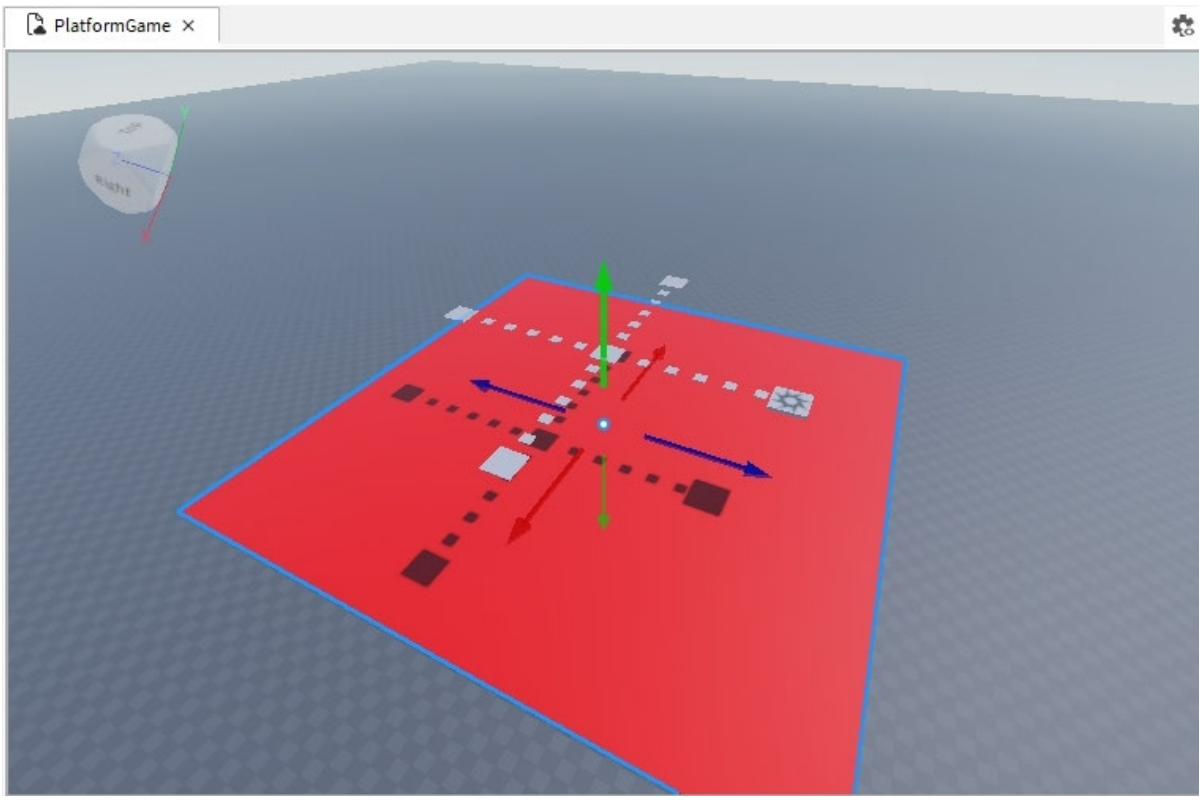


Once this is done, you can start testing the level and play the scene; your player should be able to jump from platform to platform to reach the bigger platforms.



Now that we have added the platforms, we can add two more elements: a box that will symbolize a deadzone to avoid, and items to collect.

Please create a new Part (a change its size to set its color to and place it so that its y coordinate is as illustrated in the next figure. This object will be used to detect whether the player has fallen (i.e., missed a platform).

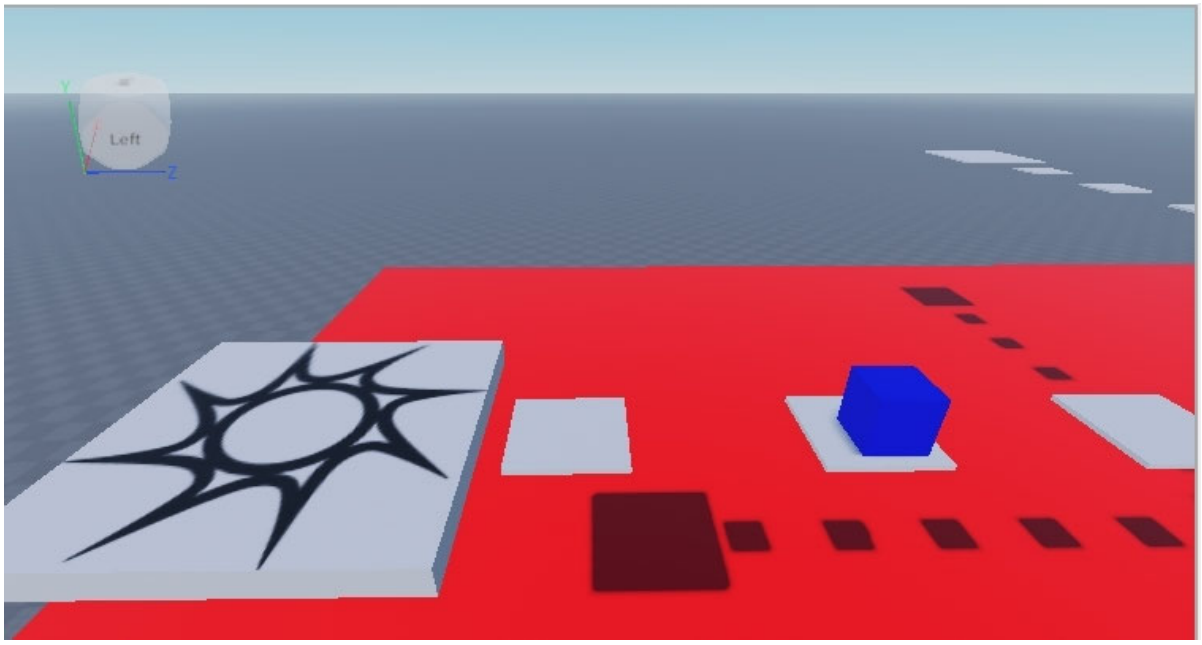


OceanofPDF.com

Adding Collectible Items

Now that the dead zone has been created, we can start to add items to be collected:

Add a new Part (a change its size to and its color to
Rename this item
Set its Anchored property to True and move this object just above one of the platforms, for example near the start of the game.



Add a new script to this object and rename the script
Open the script in the Script Editor and replace the existing code with this code:

---

function onTouched(part)

part.Parent:FindFirstChild("Humanoid") then

Collected")

end

script.Parent.Touched:Connect(onTouched)

Play the scene to check that you can effectively collect this item.

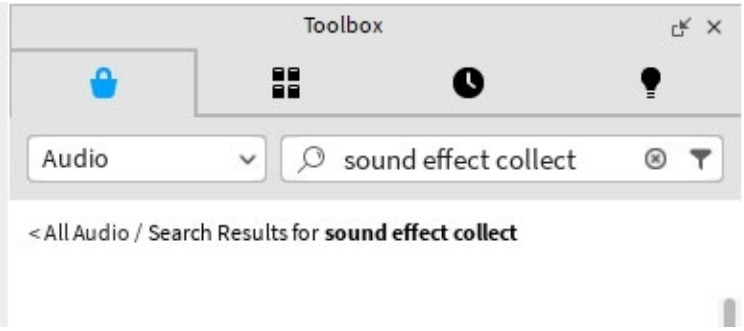
OceanofPDF.com

Using Sound Effects

While we can collect and destroy the box, it would be great to add a sound effect whenever this happens; a sound effect is a good way to provide feedback to the player and it can play a crucial role in making your game more engaging. In this section, we will add a sound effect that plays when the box has been collected by the player.

Please do the following:

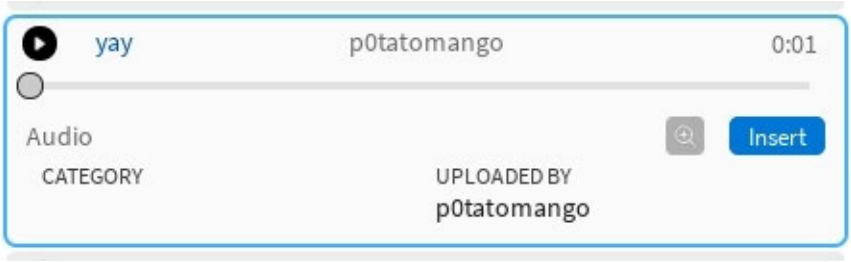
In the Explorer panel: Select the object called
Open the Toolbox panel and type effect in the search field as per the next figure.



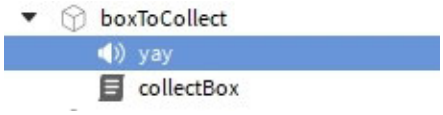
Select (i.e., click on) one of the results, for example the sound



Then click on the button

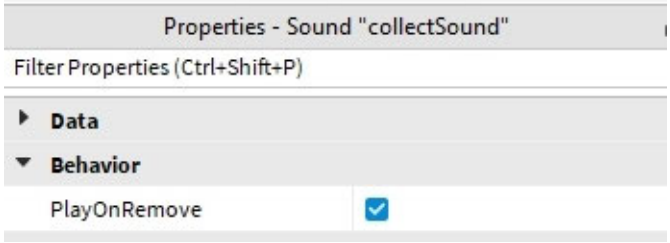


You should then see, in the Explorer window, that a new sound called yay has been added to the object.



Please rename this sound

Using the properties window, set the attribute Behavior | Play On Remove to True for this object (i.e.,

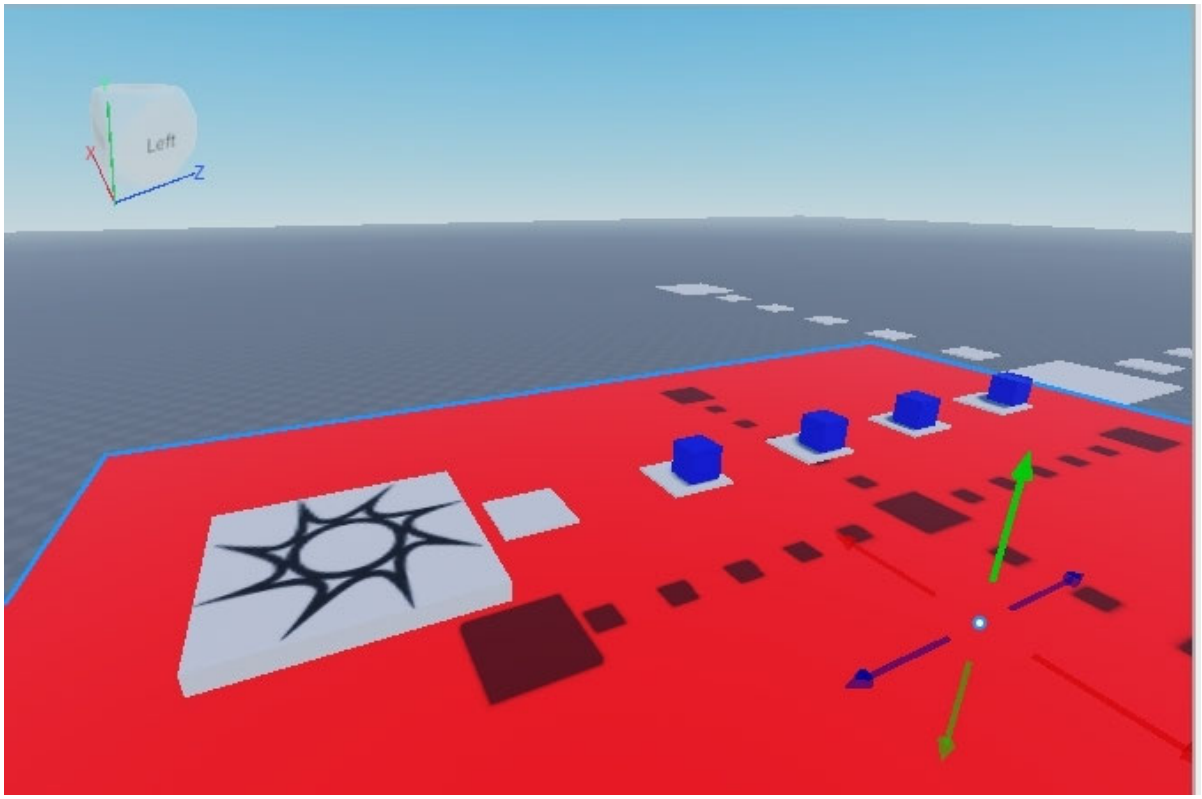


You can now play the game and check that the sound effect plays when the player character collects the box.

Because the sound is attached to the box to be collected, it will be removed from the game whenever the box has been collected, and, as a result, the sound will be played, since the option PlayOnRemove is set to

---

Duplicate the box several times.



You can test your game cand check that you can see and reach and collect the different blue boxes.



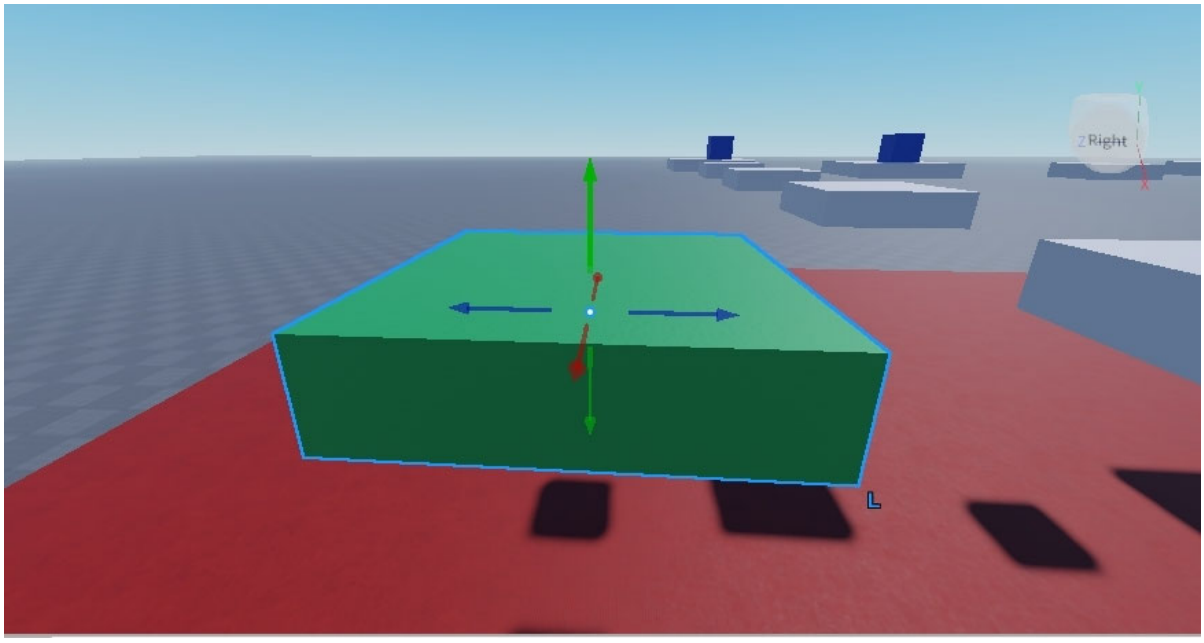
Now that we've designed the basic environment, it's time to add some interactivity to our game and we will do that in the next section.

OceanofPDF.com

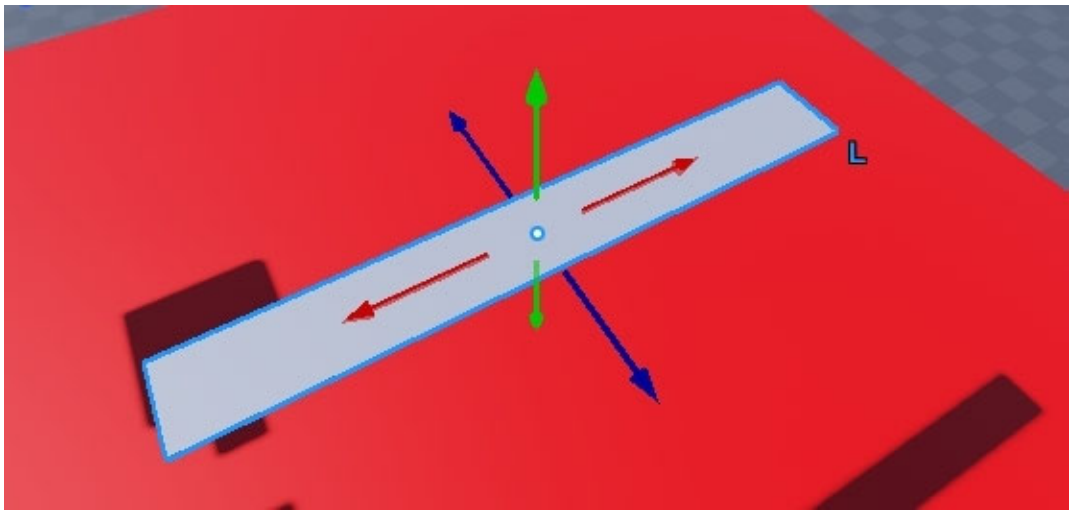
Creating an Interactive bridge

In this section, we will create a magic platform that changes sizes whenever the player has landed on it, allowing the player to progress further, even if the initial gap seemed impossible to breach.

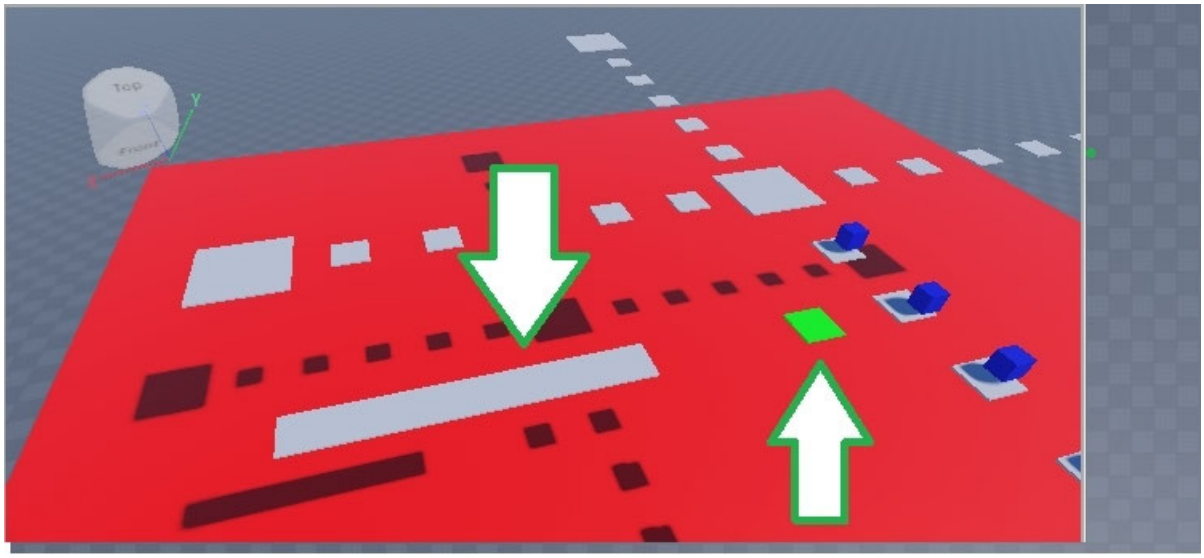
Please duplicate one of the small platforms, rename the duplicate magicPlatform and change its color to



Once this is done, create a long platform by resizing an existing small platform, for example, change its size to (4,1,31) or (31, 1, 4) depending on your layout, as illustrated in the next figure.



Rename this object bridge and place it about 20 units away from the object as per the next figure.



Since there is a significant gap between the magic platform and the bridge, we will create a script that increases the size of the magic platform

when the player steps on it, so that s/he can then move on to the bridge easily.

Please select the object
Right click on it and select Insert Object |

Once the script opens in the Script Editor, remove the code that’s already present, and add the following code:

---

local platform = script.Parent

local resized = false

local function onPlayerNear(otherPart)

character = otherPart.Parent

character:FindFirstChild("Humanoid") and resized == false then

= platform.Size + Vector3.new(2, 1, 20)

= true

```
end

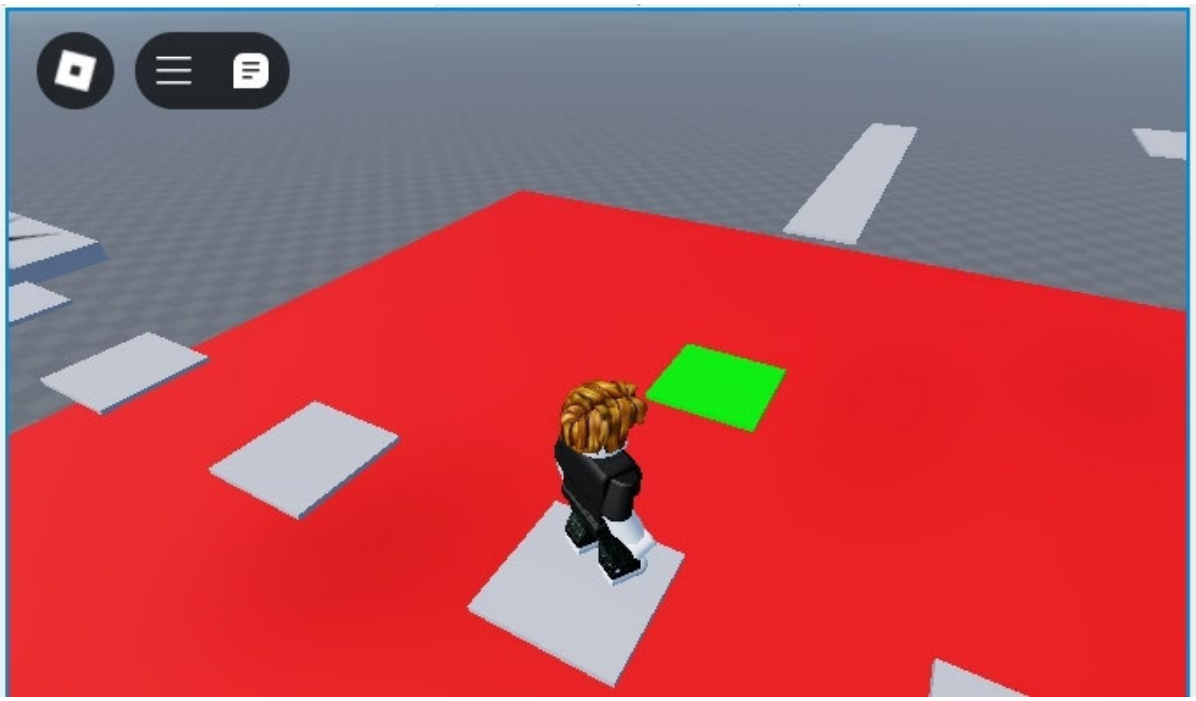
platform.Touched:Connect(onPlayerNear)
```

In the previous code:

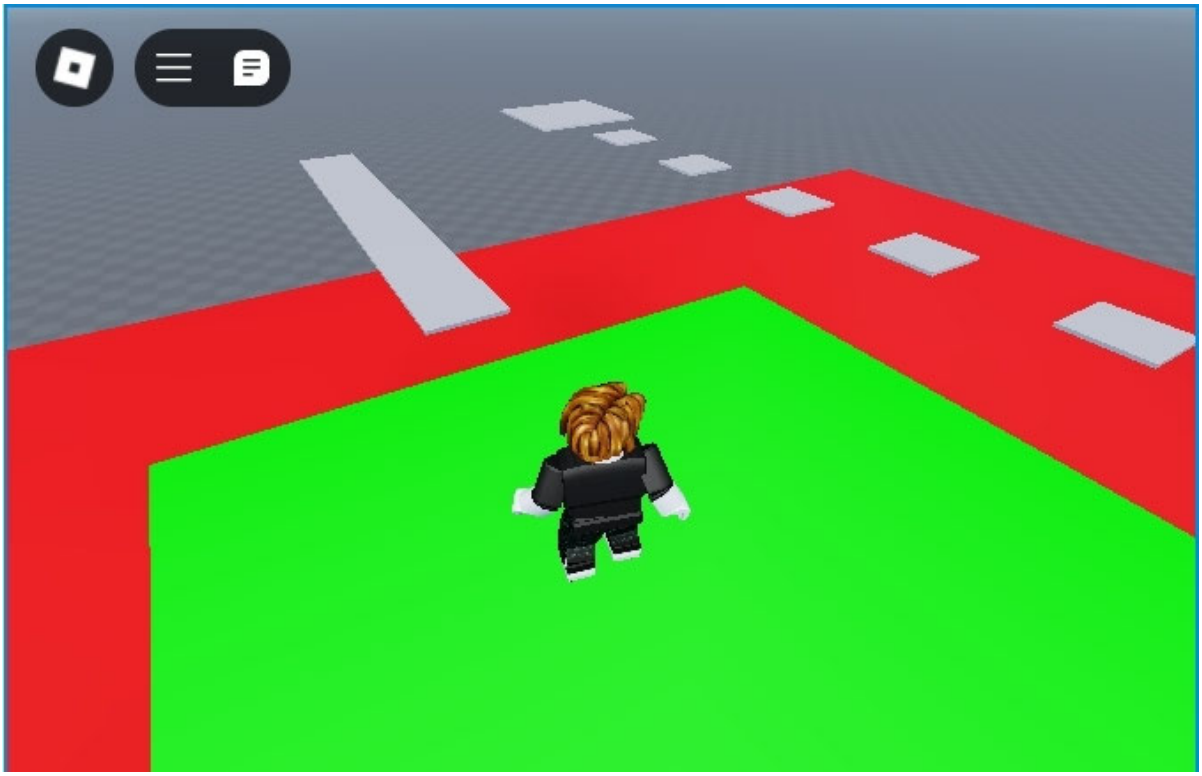
We declare a local variable called platform that refers to the magic platform to which the script is attached.
We also declare a variable called resized that will track whether we have already resized the platform.
We then create a function called
When this function is called, we check that the object touching it is the player by checking that that object includes a child called
If that’s the case, we modify the size of the platform along the x, y, and z axes.

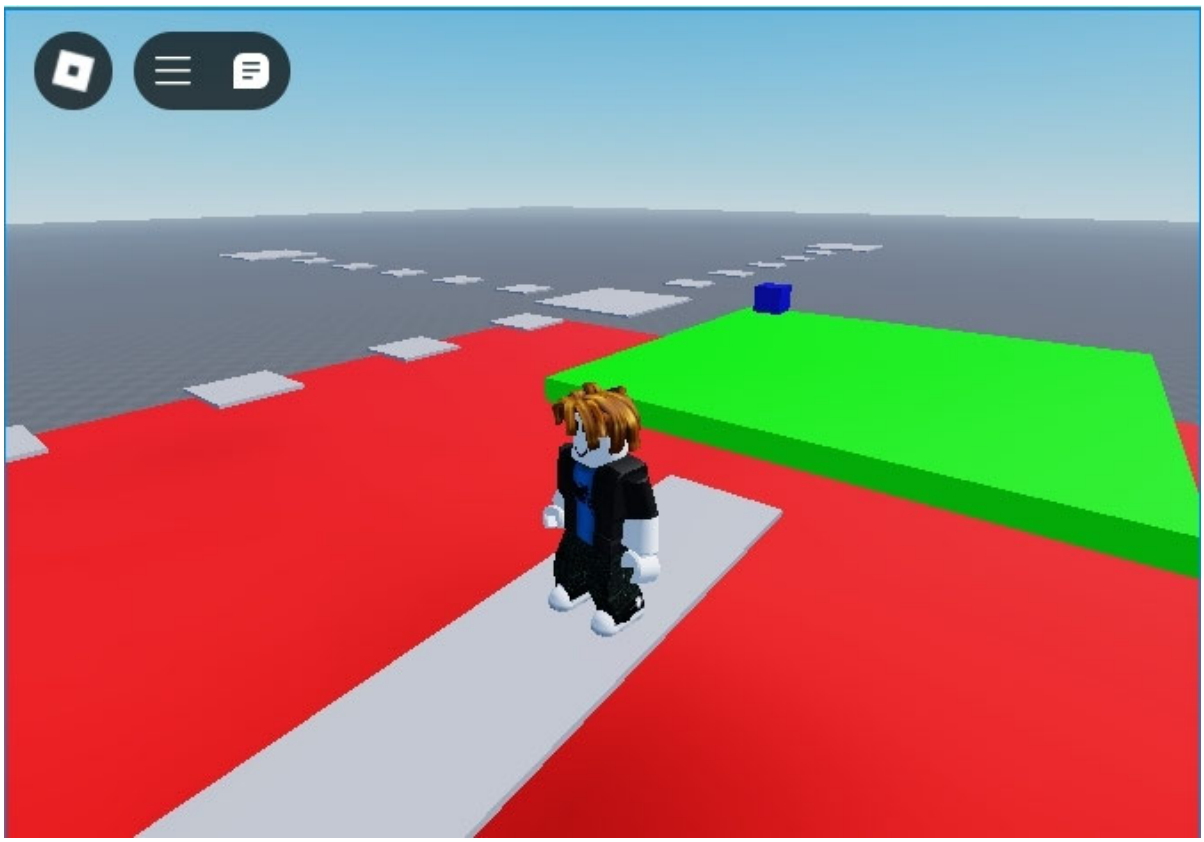
You can now test your game; for practicality, you could temporarily move the object SpawnLocation near the magic platform so that you don’t have to navigate the full level to test this feature.

As you start the game, you should see that there is a significant gap between the magic platform and the bridge



However, after stepping on the magic platform, you should see that its size has increased, and that its now possible to reach the bridge.



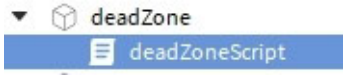


OceanofPDF.com

Adding a Script for the Dead Zone

In this section, we will create a script that ensures that the player is moved back to its initial position if s/he falls to the ground; this will involve detecting when the player has hit the ground (i.e., the dead zone) and then moving the player back above the object called Spawn Location (i.e., the starting point).

Please select the object that you have created earlier.
Add a script to this object: Right-click on the object, and select Insert Object |
Using the Explorer, rename the new script



Open the script in the script editor and remove the code that’s already present.
Add this code instead.

```
local deadZone = script.Parent

function resetPlayerPosition(otherObject)

character = otherObject.Parent
```

character:FindFirstChild("Humanoid") then

= workspace.SpawnLocation.Position + Vector3.new(0,2,0)

end

deadZone.Touched:Connect(resetPlayerPosition)

In the previous code:

We create a local variable called deadZone that will refer to the object deadZone attached to this script.
We then create a function called this function, when called, and provided that the collision is with the player, will move the player just above the object which is where the player was spawned at the start of the game.
Finally, we connect the function resetPlayerPosition to the event therefore, whenever the player touches the object the player will be moved back to its initial position.

You can now save your code and play the game. If you fall from one of the platform intentionally, you should see that the player character will be moved back to its initial position.





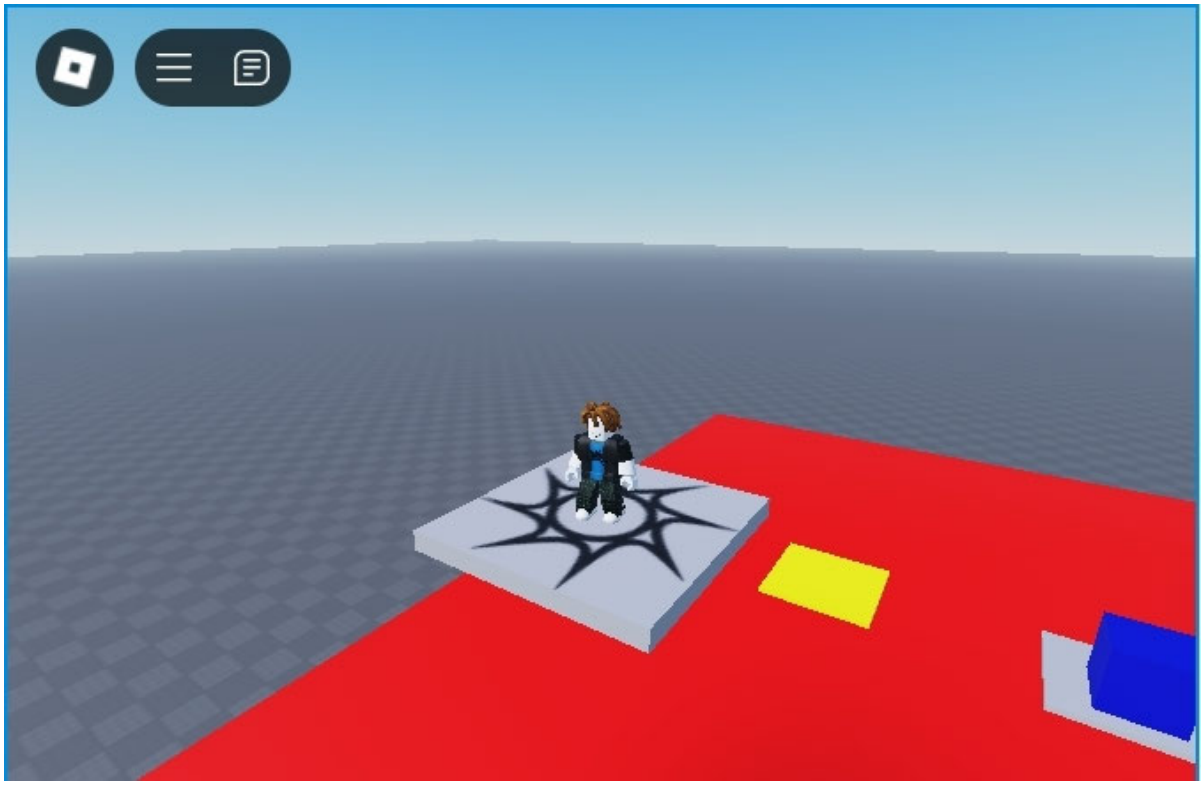


OceanofPDF.com

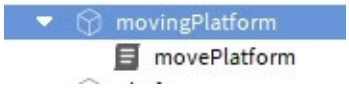
Adding Moving Platforms

Now that we have managed to reset the players' position, we will add moving platforms, a feature often found in platform games; moving platforms add some challenge to the game because they are difficult to reach due to the fact that they are constantly moving; So in this section, we will use a platform based on existing part, and attach a script to it that will move this platform from side to side.

Please select one of the platforms already present in your game, change its color to yellow, so that it can be noticed easily.



Rename that Block add a new script to it, and rename the script



Open the script in the Script Editor and replace the code already present by the following:

```
local angle = 0

local RunService = game:GetService"RunService"

local movingPlatform = script.Parent

local Heartbeat = RunService.Heartbeat

function movePlatform(DeltaTime)

+= DeltaTime

= movingPlatform.Position + Vector3.new(math.cos(angle) * .1,0,0)

end

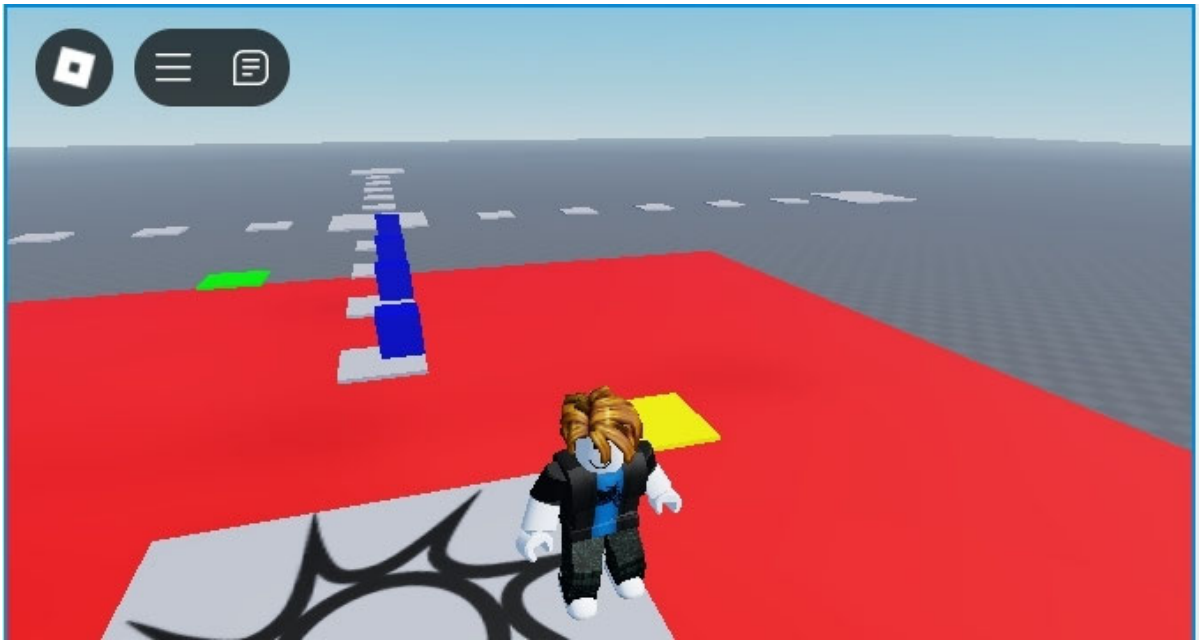
Heartbeat:Connect (movePlatform)
```

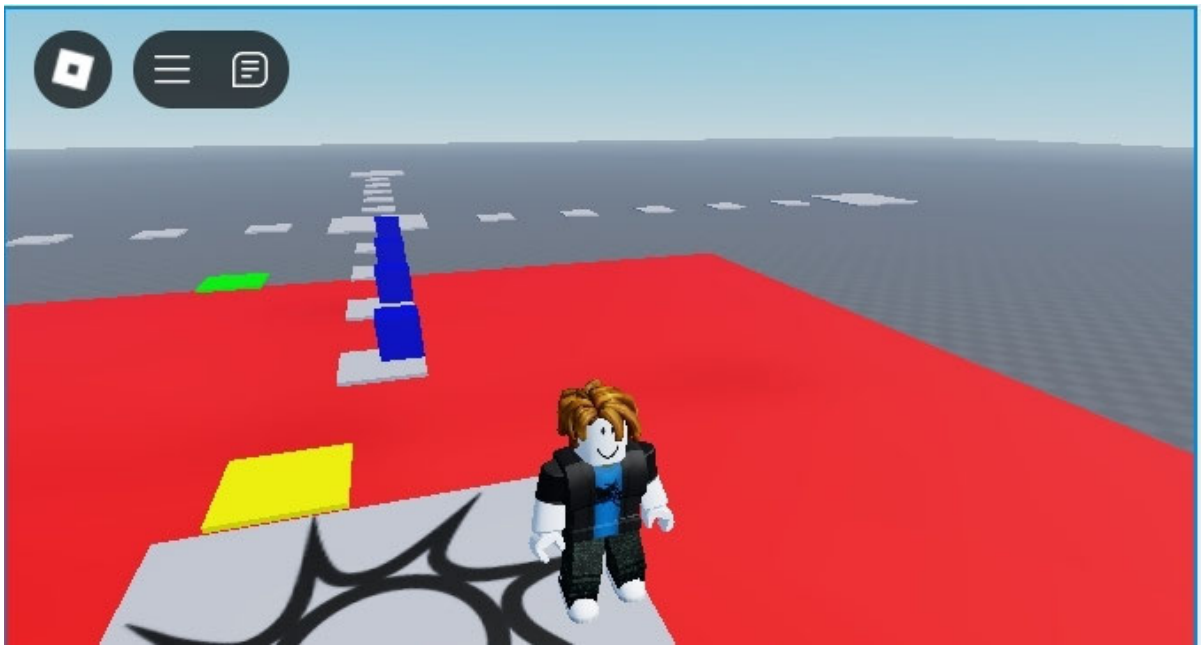
In the previous code:

We create a variable angle to keep track of the current angle used in the cosine function for calculating the platform's movement.
We retrieve the RunService from Roblox, which allows us to use the Heartbeat event that fires every time the physics engine updates the screen content.
We reference the which is the object the script is attached to, so we can change its position.
We then store the Heartbeat event in a variable, which gives us the time elapsed between frames
We define a function movePlatform(DeltaTime) that increments the angle variable based on the time elapsed since the last frame.

In this function, we update the platform's position using the cosine of the angle, moving it side to side on the x-axis.
Finally, we connect the movePlatform function to the Heartbeat event, so the platform moves continuously in sync with the game's frame updates.

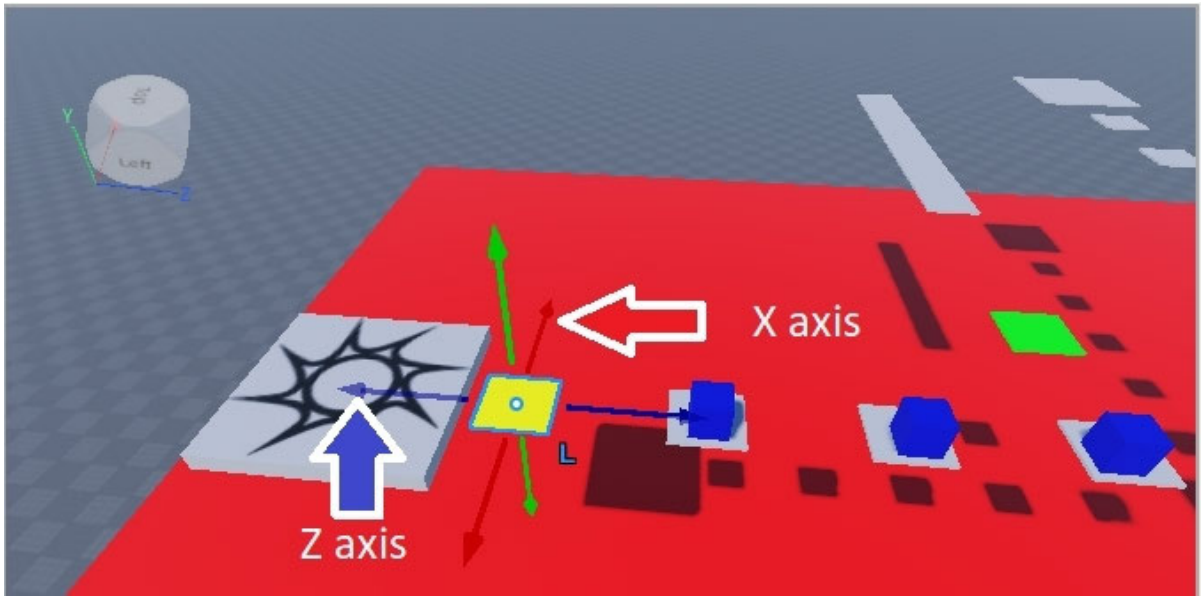
So effectively, every time the screen is refreshed (every frame) we increase the value of the variable we calculate the corresponding cosine (which varies between -1 and +1), and then move the platform accordingly.





You can now save your code and play the game; you should see that the yellow platform moves from side to side.

Note that, as per the next figure, the x axis of the yellow box is pointing to its side, so effectively, creating a movement along its x axis will cause it to move from side to side.



So, if instead you want this object to move along its z axis (the red axis on the figure), you could use the following code instead.

---

movingPlatform.Position = movingPlatform.Position + Vector3.new(0, math.cos(angle) * .1,0)

OceanofPDF.com

Creating a Generalized Script for All Boxes

At present, we have several boxes in the game, and each of them has a script attached; while this works, there is a limitation with this approach; that is: if we want to change the behaviour of all the boxes we need to modify each script; this may work well for 2 or 3 boxes, but as the number of boxes increases in your game, the process of updating the attached script can become significantly tedious; so a good solution would be to centralize the script for these boxes, so that it is updated once and then used on all the boxes.

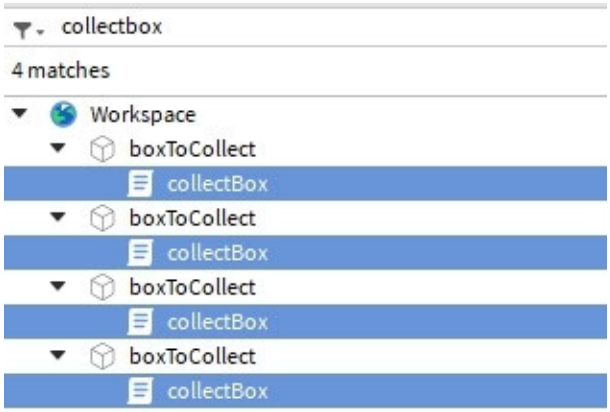
For this purpose, we could do the following:

Apply a label to each box.
Create an empty object that holds the script.
Find all the boxes to collect in the game
Add the script to these boxes based on their label at the start of the game.

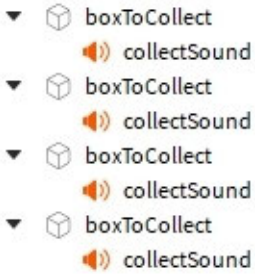
So, let’s go ahead and implement this solution.

First, we will remove the current scripts attached to the boxes:

Select all the scripts attached to each of the boxes called

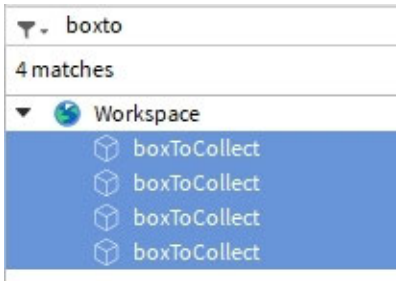


Press so that these scripts are deleted.

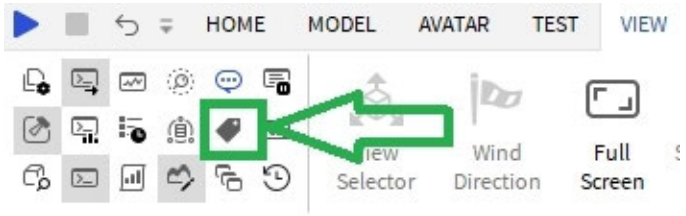


Now that we have removed the initial scripts from these boxes, we will create and attach a tag to them, so that we can identify them later on regardless of whether their name has changed.

Please select all the objects called

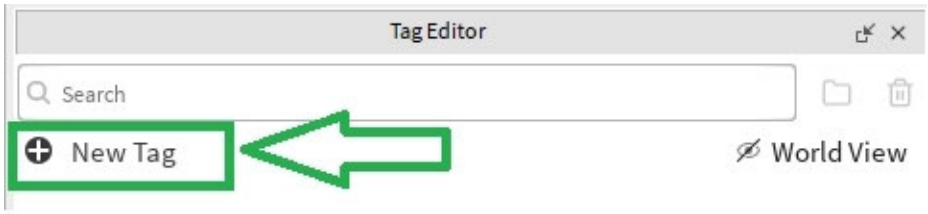


Open the Tag Editor in Roblox Studio by going to the Home tab and clicking Tag Editor (or display it using View | Tag





In the explorer, select the boxes called boxToCollect that you want to be collectible.
In the Tag Editor window, type the text click on New



Type to the right of the + button and press



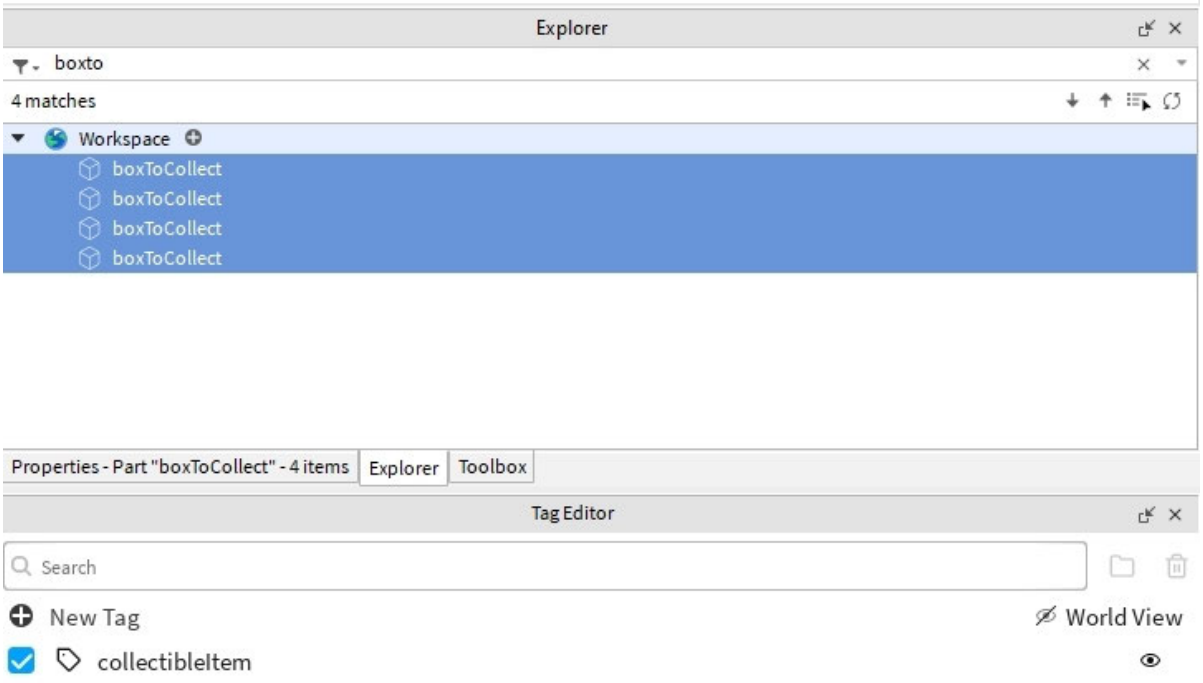
This will create and list the new tag, as per the next figure.



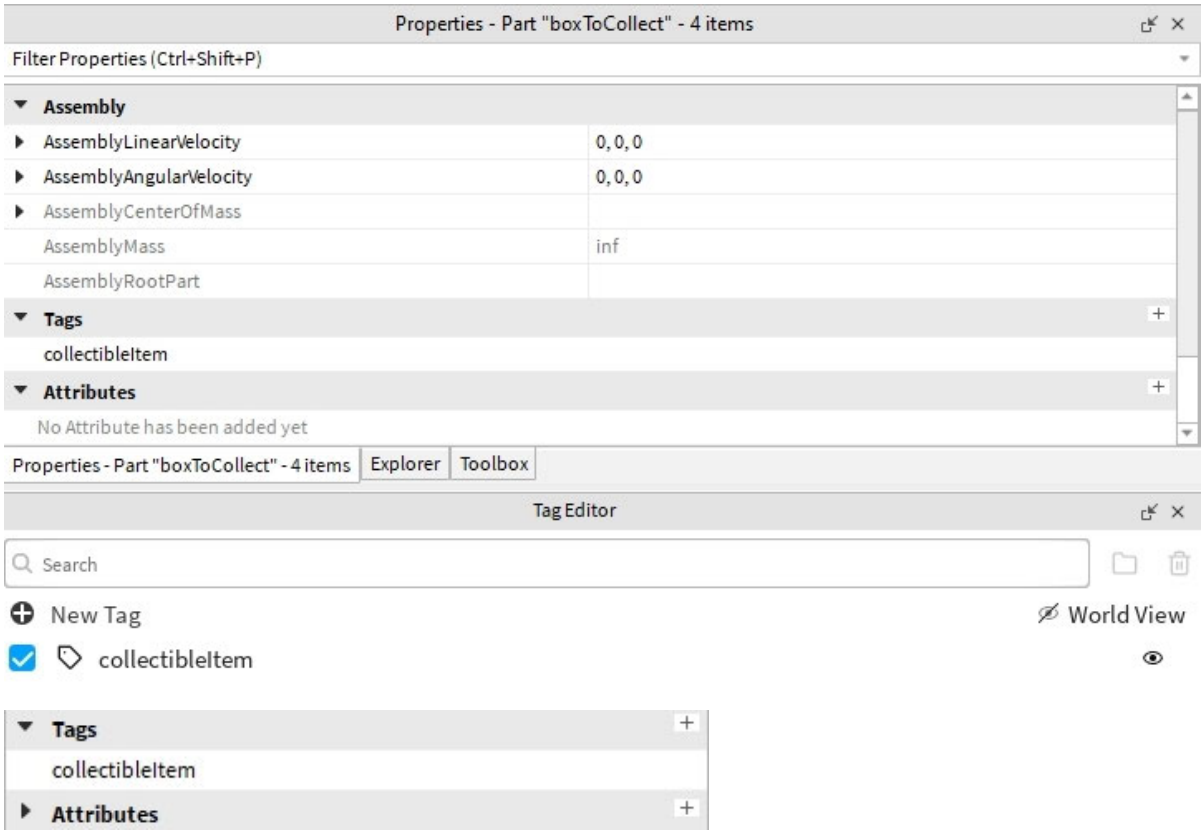
You can now tick the box to the left of the tag so that it is applied to the objects selected (i.e., all the items named



Once this is done, we can apply the tag to the box called



Now that the tag has been created and applied, you can check that the tag has effectively been applied to these objects by displaying the Properties window.



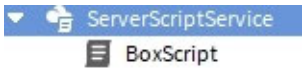
Now that we have created and assigned a tag, it is time to create a script that will automatically add functionality to these boxes tagged as

For this purpose, we will use a solution that clones a pre-existing script and attaches these cloned scripts to each box. In this solution, we will first create a script called which handles what happens when a player touches a box (like updating the score and making the box disappear). This script is stored in ServerScriptService so it runs securely on the server. Instead of creating new scripts for each box during the game (which isn't allowed),

we clone this pre-made script and attach it to every box. A main script finds all the boxes by looking for a tag (such as and attaches the cloned script to each one. This way, all boxes behave consistently, and the process is easy to manage and scale (only one script is modified but it is applied to multiple boxes).

First, we will create a script in ServerScriptService with the logic for the box:

Please right-click on the ServerScriptService object, and select Insert Object |
This will add a new script to
Rename this script



Open this script and replace the existing code with the following:

---

local function onTouched(part)

part.Parent:FindFirstChild("Humanoid") then

end

```
script.Parent.Touched:Connect(onTouched)
```

This script is the exact same as what we had for each box.

Next, we will create an empty object that will include a script responsible for the cloning of the script above.

Please create a new part.
Rename it
Add a new script to it and rename the script
Please replace the existing script with the following code:

```
local CollectionService = game:GetService("CollectionService")

local boxScriptTemplate =
game.ServerScriptService:WaitForChild("BoxScript")  -- Reference the
pre-existing script

local function addScriptToBox(box)

clonedScript = boxScriptTemplate:Clone()
```

= box

end

local function attachScriptToTaggedBoxes()

taggedBoxes = CollectionService:GetTagged("collectibleItem")

_, box in pairs(taggedBoxes) do

box:IsA("BasePart") then -- Ensure it's a part (or any valid object type)

end

attachScriptToTaggedBoxes()

In the previous script:

We get the This allows us to work with tags in Roblox, specifically to find objects tagged with a label like This is because it lets us target only the

objects (boxes) that are tagged, avoiding the need to manually select them by name.

We reference the pre-existing script The script is stored in and it contains the logic that should be attached to each box. The reason for referencing it is to ensure that the same functionality is applied to every box.
We create a function to add the script to each box: In this function, we clone the script BoxScript and attach it to the box. Cloning ensures each box gets its own copy of the script, making the functionality independent for each box. The script is then attached to the specific box by setting it as the parent.
We retrieve all objects tagged as This allows us to find all boxes that have been tagged as The reason is that tags are a more flexible way of managing groups of objects than relying on their names.
We loop through the tagged boxes: In the loop, we check if the object is a valid part (like a box) and then call the function to attach the script to it. The reason for this loop is to ensure that only appropriate objects (like parts) are modified, not other objects that might be tagged but shouldn’t have the script attached.

We call the function to attach the script: Finally, the function that handles attaching the script is executed, ensuring that all tagged boxes receive the functionality (such as increasing the score when touched). This automates the process, so you don't need to manually attach scripts to every box in the game.

You can now play the game and check that upon colliding with each of the boxes, the boxes is destroyed and a sound effect is played.

OceanofPDF.com

Creating a Simple Scoring System

At this stage, we have most of the mechanics covered: collectible items, a moving platform, an extendable platform, as well as a dead zone; we could now include a scoring system to keep track of the player's progress.

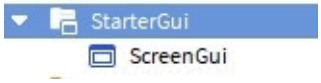
A scoring system is essential for tracking player progress and adding competitiveness to your game.

So, in this section, we will create a simple scoring system that increments every time the player collects an item.

Click on StarterGui in the



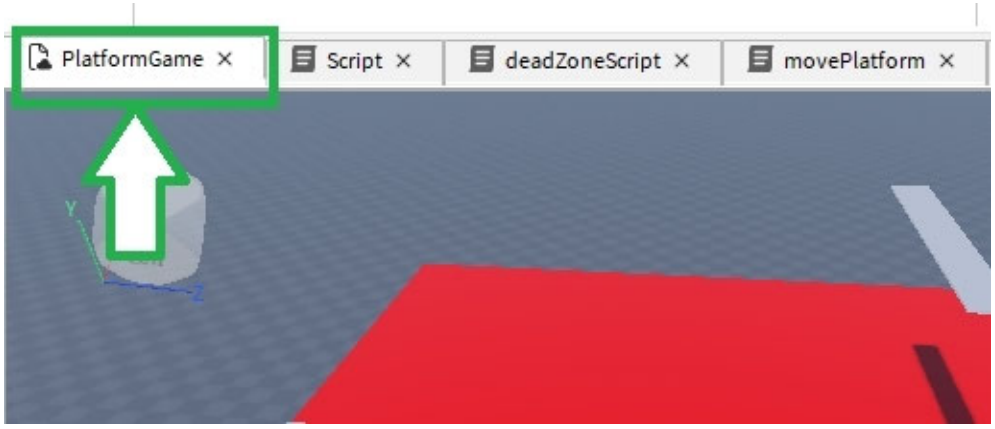
Right-click and select Insert Object |
This will create a new item called



Rename it



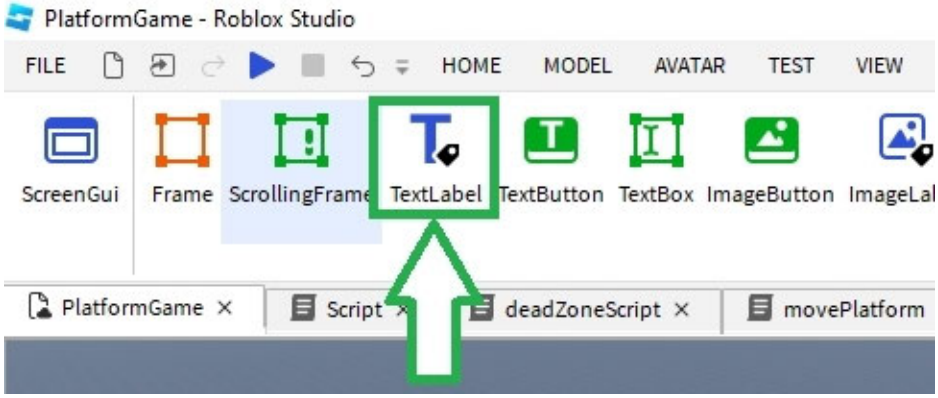
In the Explorer: click on the item
Click on the tab with the 3D Viewport called



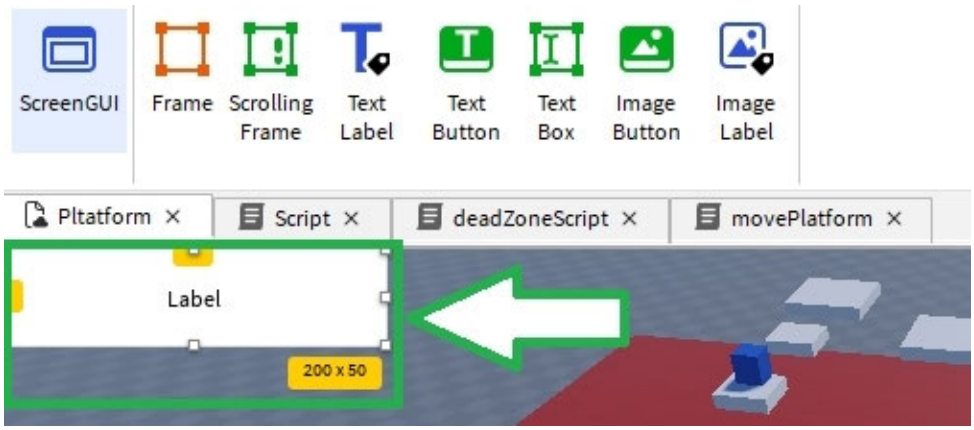
and you should see a set of tools appearing at the top of the workspace.



Click on the icon Text



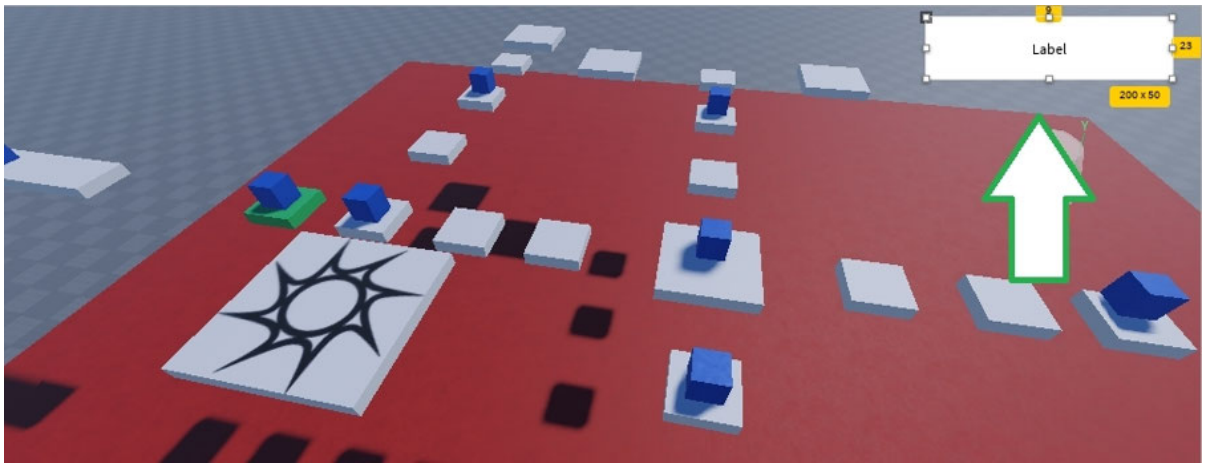
This will create a new label, as per the next figure:



This will create a new object called TextLabel in the



In the 3D move (i.e., drag and drop) this label to the top right corner of the window, as per the next figure.



Now that our label has been created, we just need to modify our script so that the score is increased and update onscreen whenever we collect a box.

First, we will create a leader board for the player, that is to say, data attached to the player that will track its progress; in our case we just want to follow its score (i.e., increased every time the player collects a box)

In the Explorer panel, add a script to the object called ServerScriptService and rename that script
Add the following code to it (after removing the code already within):

---

function initPlayerStats(player)

leaderstats = Instance.new("Folder")

= "leaderstats"

= player

score = Instance.new("IntValue")

= "Score"

= 0

= leaderstats

```
end

game.Players.PlayerAdded:Connect(initPlayerStats)
```

In the previous code:

Every time a new player is added to this game we are going to create a folder and a variable that holds its score.
First, we create new folder and rename it
We also attach this folder to the object player (which becomes its parent)
We then create an integer variable called Score that we initialize to 0

We put the code in ServerScriptService because it ensures that the script runs securely on the where it can control important game functions like player scores and prevent cheating. This way, only the server can manage key game data, making the game fair and safe for all players.

Finally, we just need to modify our original script called BoxScript to increase the score every time we collect a box:

Open the script
Modify the script as per the next code snippet (new code in bold).

```
local function onTouched(part)
    if part.Parent:FindFirstChild("Humanoid") then
    local player =  game.Players:GetPlayerFromCharacter(part.Parent)
    local score = player.leaderstats.Score.Value;
    score = score + 1;
    player.leaderstats.Score.Value = score;
    player.PlayerGui.ScoreGui.ScoreLabel.Text = "Score: "..score
    print(score)
    script.Parent:Destroy()  -- Destroy the box when touched
    end
end
```

In the previous code, we do the following:

The function onTouched is defined: This function is triggered whenever the object, that this script is attached to, is touched by another object (such as a player’s character).
Check if the touched object belongs to a player: The script checks if part.Parent (the object that touched it) contains a A Humanoid object is only found in player characters in Roblox, so this check ensures that the box was touched by a player, not by something else in the game (like another part or object).
Retrieve the player object: If the player’s character touched the part, the script uses game.Players:GetPlayerFromCharacter(part.Parent) to find the Player object associated with the character. This links the player's character model to the player controlling it.
Access the player's score: The script looks for the player's leaderstats folder (which holds the player's stats) and accesses the Score value. This is where the player's current score is stored.
Increment the player's score: The current Score is incremented by The new score is then assigned back to player.leaderstats.Score.Value to update the player's stats.
Update the player's UI: The script updates the TextLabel in the player's PlayerGui to reflect the new score. The label is updated to show " followed by the new score value.

Print the new score: For debugging purposes, the script prints the new score to the Output window so that we can verify that the score has been correctly updated.

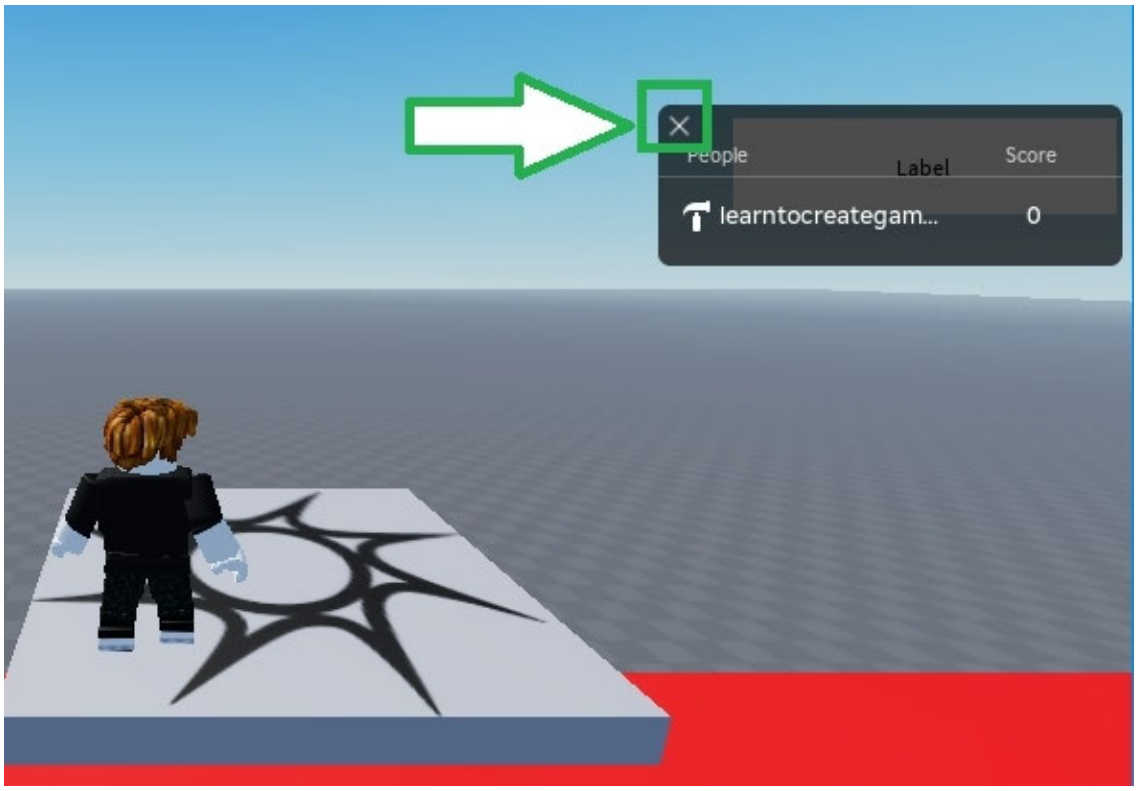
Destroy the box after being touched: After the player touches the box and the score is updated, the script removes (destroys) the box from the game to simulate that it has been collected. This ensures that the box cannot be touched again.

So effectively, this script updates the score and displays it on the screen whenever the player collects an item.

You can now play the game, collect boxes and see how the score is updated onscreen.



Note that to see the text label properly, you may need to close the stats window that is display atop, as per the next figure:



With a scoring system implemented, players now have a way to track their progress.

OceanofPDF.com

Summary

In this chapter, you've learned how to design a simple game environment, script basic player movement, and add interactive elements like collectibles. You also added sound effects and a scoring system to make the game more engaging.

You've made significant progress since the start of the book, and your game is starting to take shape. In the next chapter, we'll explore more advanced mechanics, including multiple levels and checkpoints to further enhance your game.

OceanofPDF.com

Quiz

Please guess whether the following statements are True or False (the solution is in the section Answers to at the end of the book).

You can create a part in Roblox Studio using the Home
The Touched event in Roblox is triggered when a player clicks on a part.
Vector3 is used in Roblox to represent 3D coordinates like position or size.
The PlayerMovement script allows players to move their character using the arrow keys.
The Destroy method in Roblox permanently removes an object from the game.
A script attached to a part can detect when the part is touched by another object.
Collectible items are created by resizing a block and adding a destroy script.
The ChangeState method is used to move parts around the game environment.
Interactive doors can be scripted to open when a player gets close.
The Insert Object menu in the Explorer panel is used to add scripts to objects.

OceanofPDF.com

Challenge

Try these challenges to put your newly acquired skills to the test:

Create a Moving Script a platform that moves back and forth when the player steps on it.
Add a Scoring Create a system that keeps track of how many collectibles the player has gathered.
Design a Simple Create a door that only opens when the player has collected a certain number of items.

OceanofPDF.com

7

Creating and Managing Multiple levels

In this chapter, we will explore how to create, structure, and manage multiple levels within your game. In the previous chapters, you have learned how to set up a system where the player collects boxes and updates their score. Now, we will build on that by introducing level progression, triggers, and win conditions that allow the player to advance through various levels. By the end of this chapter, you'll have created a multi-level game with smooth transitions between stages, and clear win conditions.

In this section, we will cover:

Structuring multiple levels.
Managing player progression.
Implementing triggers for level transitions.

So, after completing this chapter, you will be able to:

Structure different levels in Roblox Studio.
Implement progression tracking for players as they advance.
Create and script triggers for transitioning between levels.

Define and implement win conditions for levels.

OceanofPDF.com

How to Implement Multiple levels

In this section, we will learn how to build multiple levels in your game and organize them effectively. A well-structured game should provide a smooth flow between levels, keeping the player engaged as they progress through each stage.

There’s mainly two ways to design multiple levels in Roblox:

Create the multiple levels display only the current level. For example, hide levels 2 and 3 while the player is in level1

Or

Create and display all levels, place these levels far apart and teleport the player to the next level when a condition has been met, for example after reaching a check point.

For practicality, we will go with the second option as it avoids unintentionally modifying objects that are invisible, and also it makes for a clearer overview of the full game.

So, in the next section we will do the following:

Duplicate the current level.
Modify the duplicate (i.e., the second level) so that it can be recognized easily.
Create a starting point in the second level.
Track the current level.
Teleport the player to the starting point of the second level whenever s/he has reached a specific check point in the first level.

Ensure that falling on the dead zone in the second level restarts the player in the same level.

OceanofPDF.com

Structuring Levels

When designing multiple levels, it is important to maintain consistency while introducing new challenges. So, we will organize our levels using folders, ensuring that they are easy to manage and modify.

First, we will create a folder for each level and

In the Explorer panel, right-click on the object Workspace and select Insert Object |
Name the folder
Right-click on the folder levels and select Insert Object | Folder again.
Name the new folder

Now that we have created a folder for the first level, we can add all the content of our level (the objects that we have created) inside that folder:

Select all the objects that you have in the current level: In the Explorer panel, under look for all the objects that make up your current game level (these would typically include the terrain, parts, models, collectibles, and other elements of your game). Hold Ctrl (or Cmd on Mac) and click each of these objects to select them all.
The objects to be selected include for level1 are: all the platforms, all the objects named

The objects NOT to be included in level1 are: scriptHolder and Terrain (unless you want to include a terrain in the second level).

Tip: To select multiple objects quickly, hold the Shift key and click to select everything in between two objects.

Note that you do not need to select global objects such as or SoundService unless they are specific to the level.
Move the objects to the folder After selecting all the relevant objects (terrain, parts, models, etc.), Copy/Paste them (i.e., CTRL + C and CTRL + or move (drag and drop) them to the folder
This will ensure all the current level elements are now organized in the level1 folder.

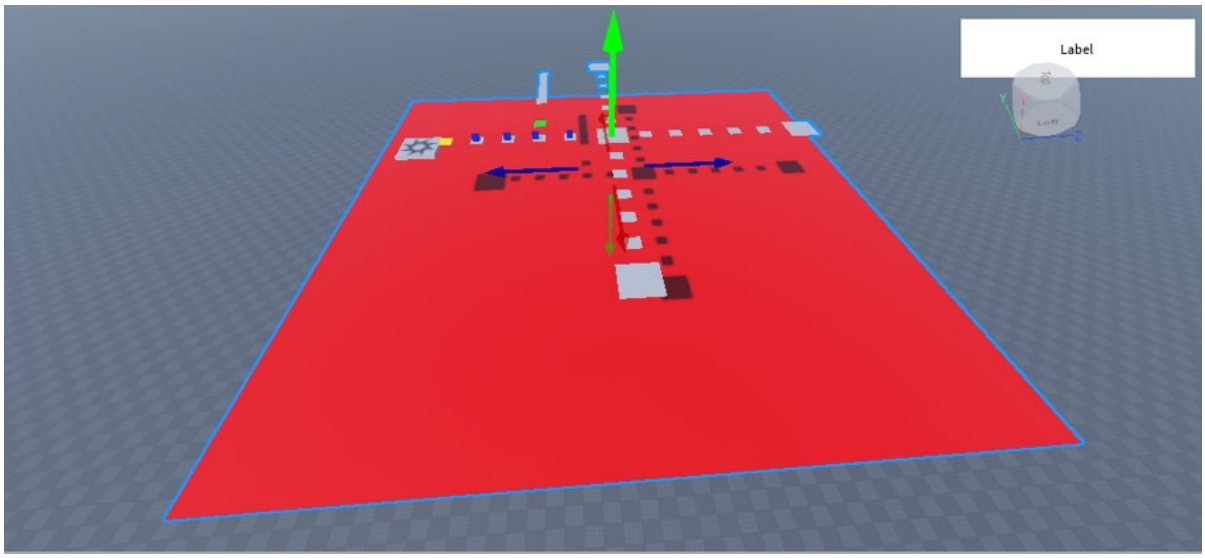
Now that the content of our level is saved in level1 we will duplicate that level and apply slight changes to that level to create a new (but similar) level:

Duplicate the folder level1 (i.e., CTRL + C and CTRL + and rename the duplicate
If this folder appears outside the folder levels, please drag and drop it inside that folder.

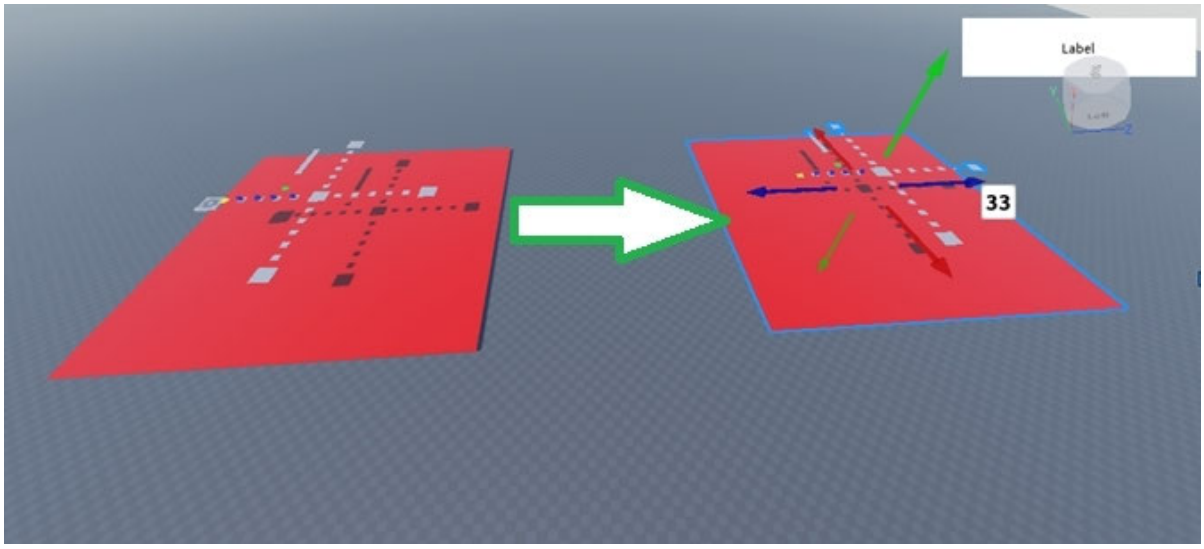


In the Explorer, select the folder called level2 and then select the Move tool in the 3D

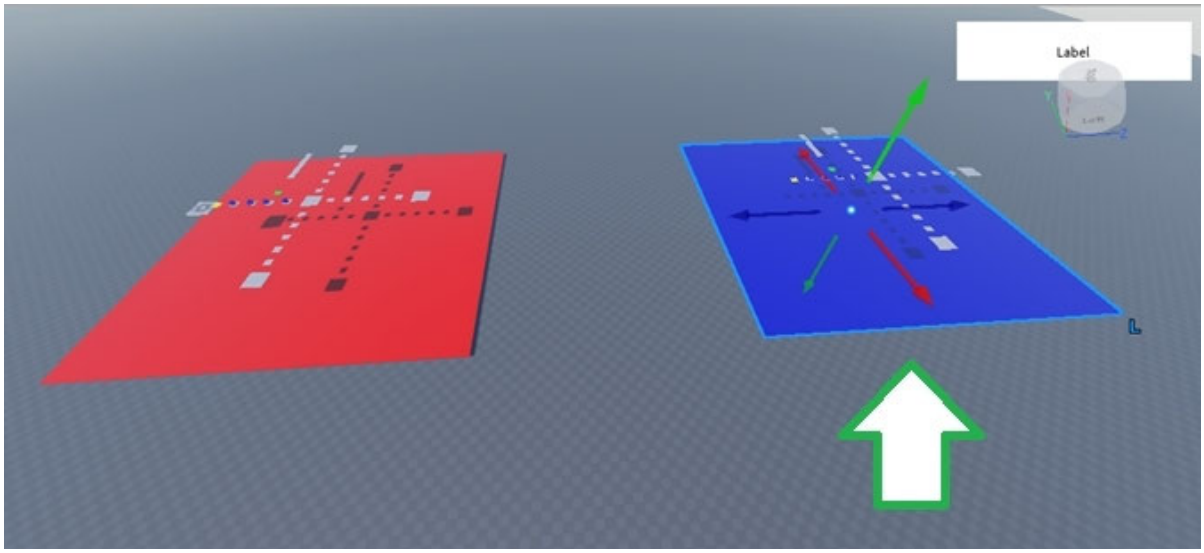
One you select the Move tool, the usual green, red and blue arrow will appear in the 3D since we have selected the folder it means that any transformation that we apply will be applied to all the object in the folder



In the 3D using the Move tool, move the content of the folder further down the x or z axis.



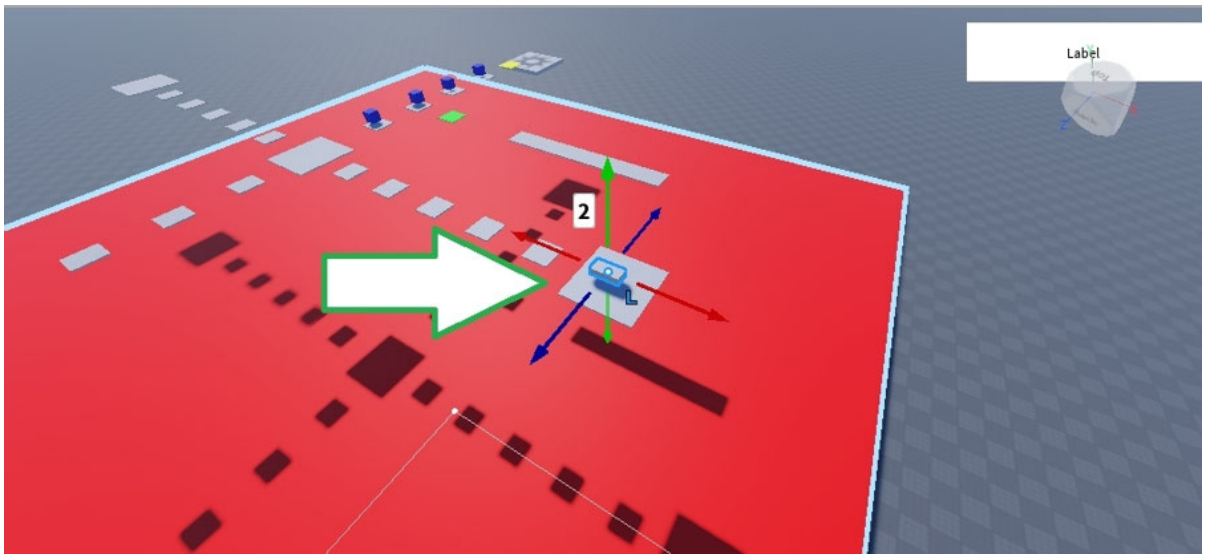
When this is done, you can start to modify the content of For example, the next figure illustrates how the second level was modified to make it look significantly different, for example, using a blue dead zone.



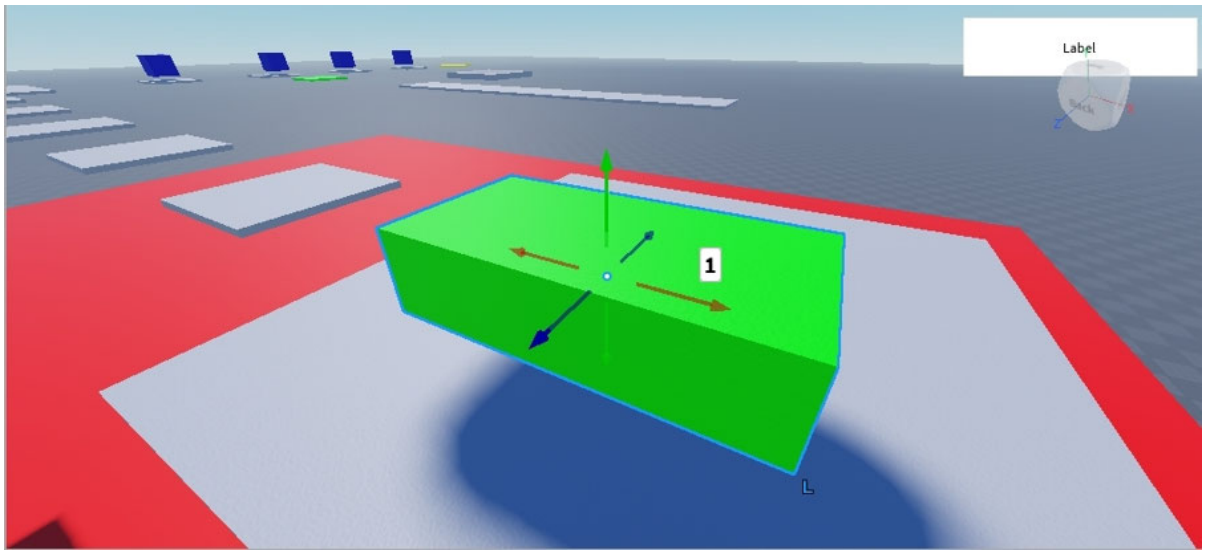
Creating a Checkpoint

Now that you have created the second level, it’s time to create a checkpoint in the first level to teleport the player to the second level:

Please create a new part and make sure that it is stored in in the Explorer window, right-click on the folder then select Insert | Object |
Rename this part checkpoint and move it on one of the existing platforms, as per the next figure.



Using the Properties window, change its color to green (or any color of your choice) and its Anchored attribute to

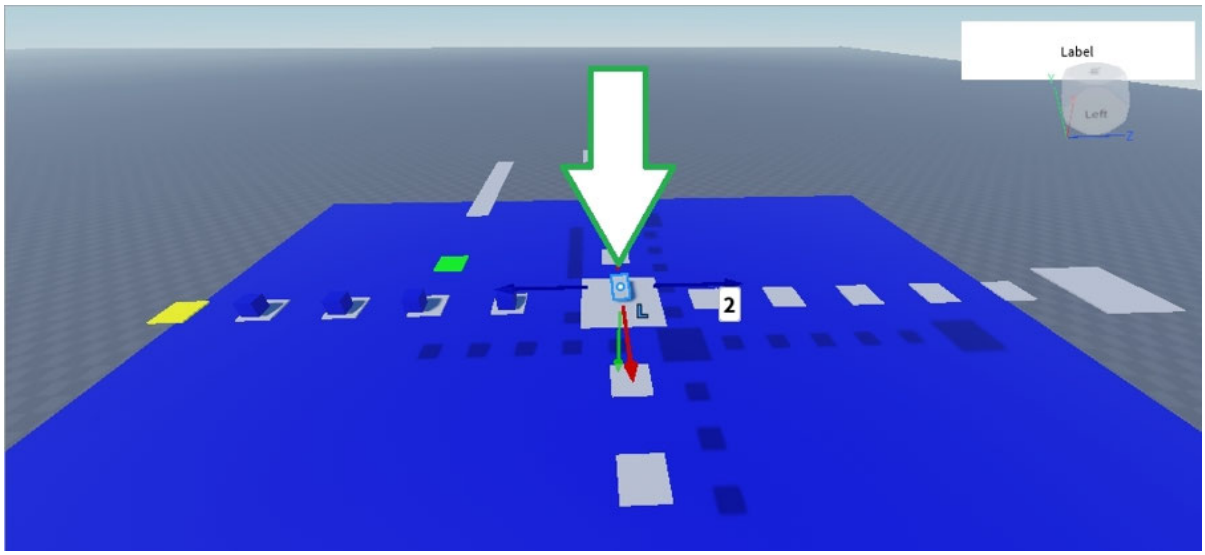


OceanofPDF.com

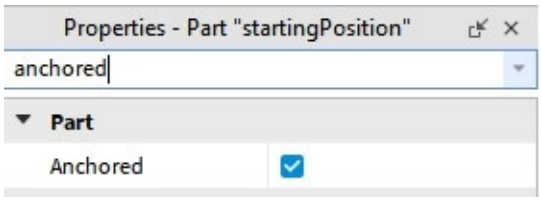
Creating a Starting Point

Now that we have created the zone that will be used as a checkpoint, we will create a starting point in the second level to indicate where the player should be teleported to in that level after reaching first level's checkpoint:

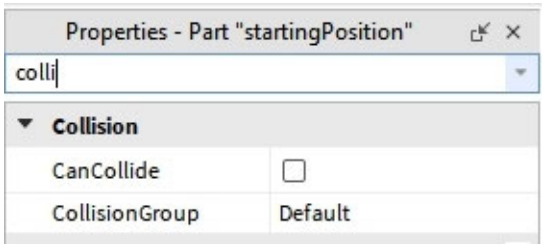
Using the Explorer window, right-click on the folder and select Insert Object |
Rename that part
Place it above an existing platform in as per the next figure.



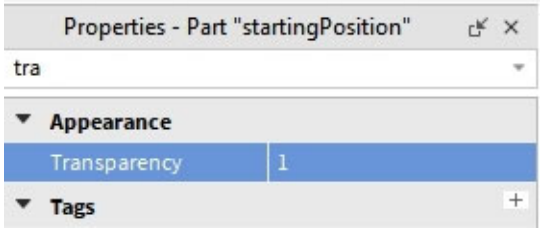
Using the Properties window, Set this object's Anchored | attribute to True



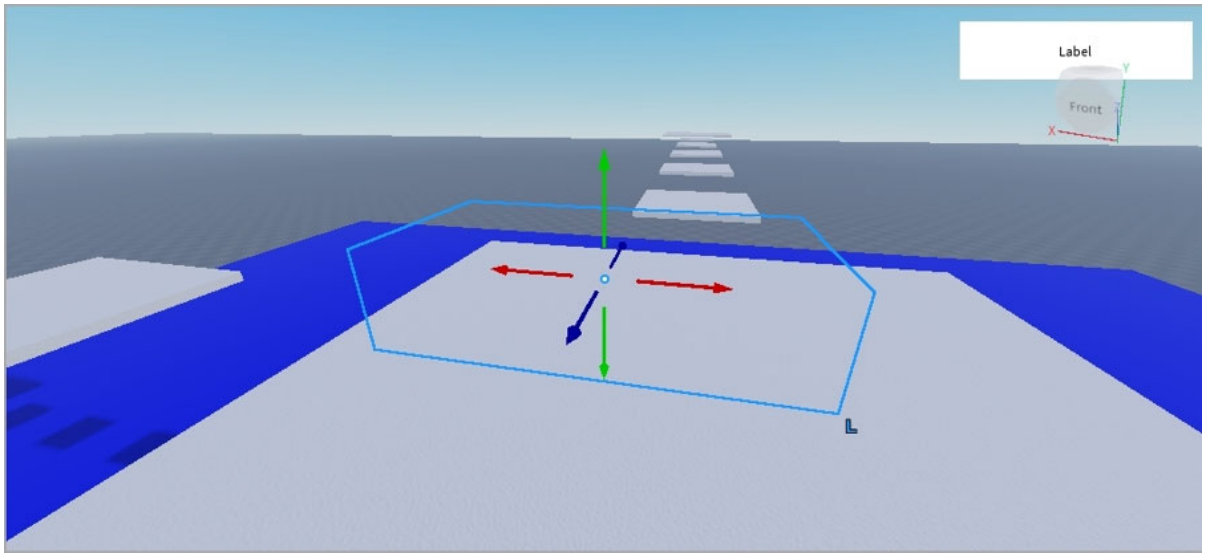
Set its CanCollide attribute | to



Set its Transparency attribute | to



Once this is done, the object should look like the next figure: transparent with a blue highlight around its edges, when it is selected.



OceanofPDF.com

Transitioning from the Checkpoint to the Second Level

So, at this stage, we have a checkpoint for level1 and also a starting point for and we need to link them; but before that, we will create and initialise a new leaderboard variable that will track the level reached by the player.

Open the script
Add the following code to it (new code in bold).

---

function initPlayerStats(player)

leaderstats = Instance.new("Folder")

= "leaderstats"

= player

score = Instance.new("IntValue")

= "Score"

= 0

= leaderstats

level = Instance.new("IntValue")

= "Level"

= 1 -- Initial level is 1

= leaderstats

end

game.Players.PlayerAdded:Connect(initPlayerStats)

In the previous code, we create a new integer value called Level that will be used to track the current level for the player; this variable is set to 1 by default.

Next, we will create the code to be executed when the player reaches the checkpoint:

Create a new script attached to the object checkpoint and rename that script
Open the script, and replace the existing code with this code:

```
function teleportPlayer(part)

player = game.Players:GetPlayerFromCharacter(part.Parent)

player then



= player.leaderstats.Level.Value + 1



end

script.Parent.Touched:Connect (teleportPlayer)
```

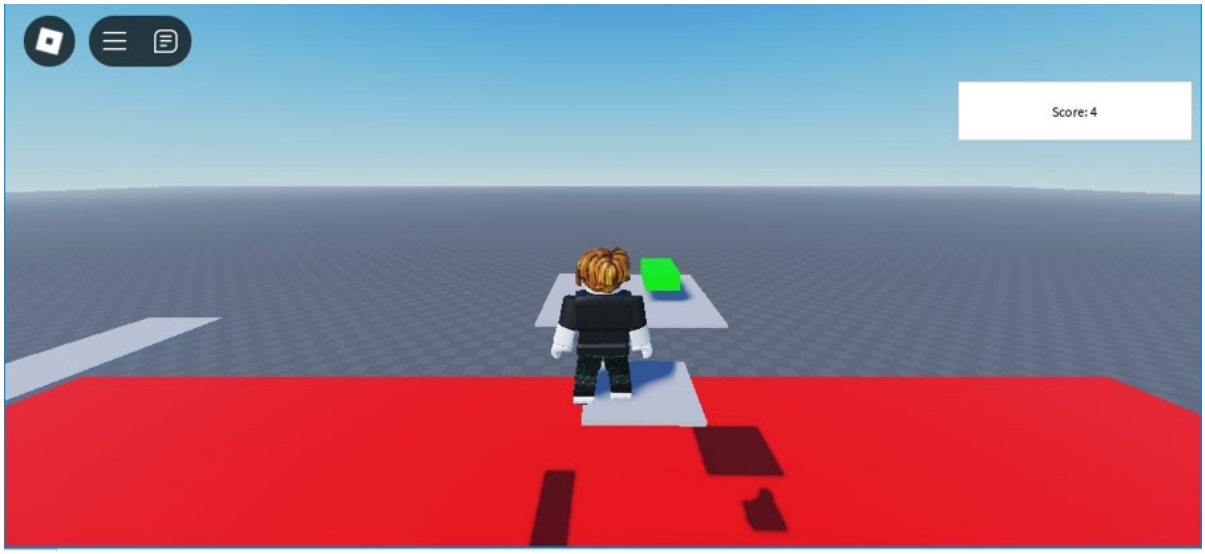
In the previous code:

We create a function called teleportPlayer that will be used to teleport the player to the second level when it reaches (i.e., collides with) the checkpoint.

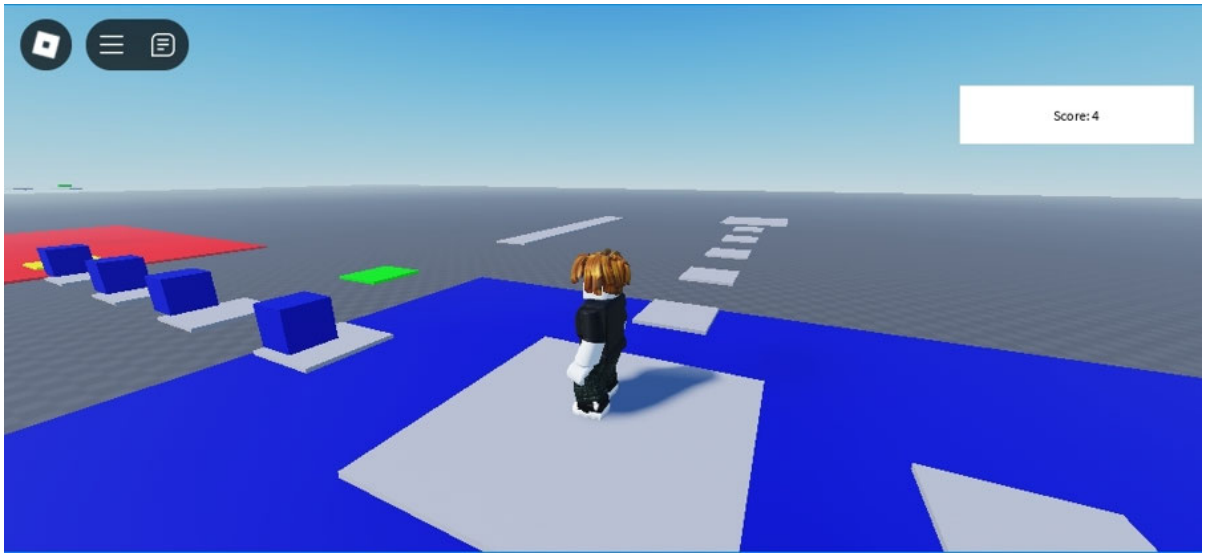
In that function, we check that the object colliding with the checkpoint is the player.
If that’s the case, then we move the player to the same position as the object startingPoint from in the second level.

Note that we use the function This method sets the position and orientation of the primary part of the player’s character. The primary part is usually the which acts as the central part of the player’s character model, responsible for movement and positioning.

Finally, we connect the Touch event to the function
You can now save your code and play the game.
Move the player character near the checkpoint.



You should see that as soon as the player character reaches (i.e., collides with) the checkpoint, it is moved to the second level.



OceanofPDF.com

Modifying the second dead zone

Great; so, this seems to work well and your player can transition to the second level smoothly after reaching the first checkpoint. While this works, we still need to include some improvement; for example: the dead zone in the second level still contains code that will reset the player to the first level upon collision, and you can check this by playing the game, and reaching the check point. Once you are in the second level, jump on the blue dead zone, you should see that your character is then moved to the first level; this is because the code associated to the dead zone in level2 was created before we created the second level; so all we need to do is to change that code so that upon falling on the blue dead zone (in level2) the player is moved to the starting point in the second level (and not the first level):

Please select the object deadZone from the second level.
It should have a script called deadZoneScript attached to it, as per the next figure.



Open this script and replace the code within with the following (new code in bold):

```
local deadZone = script.Parent

function resetPlayerPosition(otherObject)

character = otherObject.Parent

character:FindFirstChild("Humanoid") then



= workspace.SpawnLocation.Position + Vector3.new(0,2,0 )

= workspace.levels.level2.startingPosition.Position



end

deadZone.Touched:Connect(resetPlayerPosition)
```

In the previous code, we move the player to the starting point for

You can now test your game and ensure that if you the player character falls from a platform in level2, that it is moved to the starting point for

that level.

OceanofPDF.com

Creating and Displaying a Timer

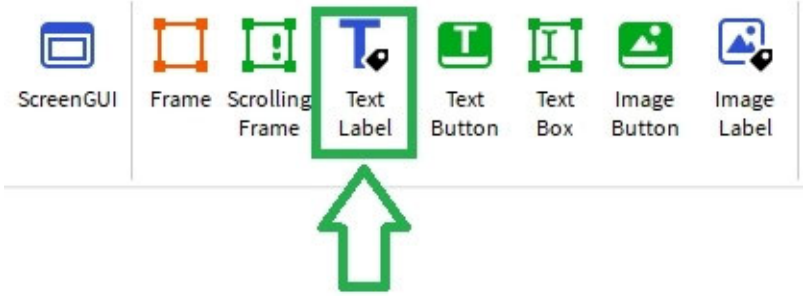
In this section, we will learn how to create a countdown timer that challenges players to complete each level within 60 seconds. Timers are often used in action games as they add a layer of challenge and urgency to the game, making each level more engaging. If the players fail to complete the level in time, they will be returned to the starting point of that level.

So, in this section, we will cover:

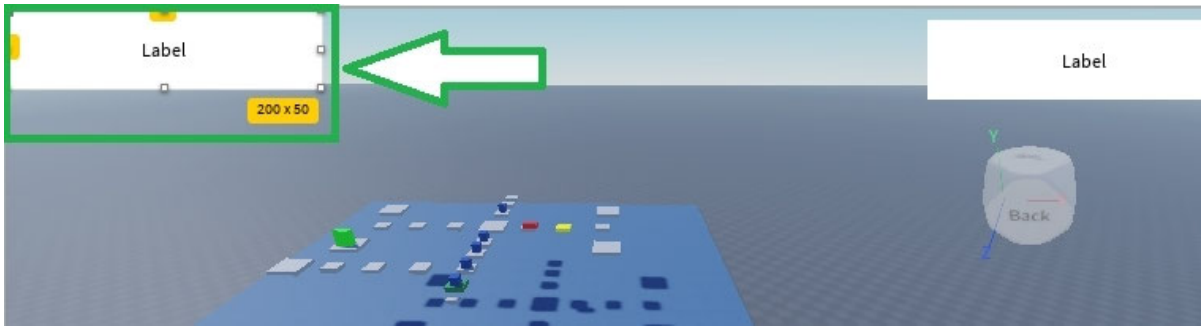
Creating a timer that starts when the player enters a level.
Returning the player to the level's starting point if time runs out.
Displaying the timer on the screen.

First, let's add and display the timer:

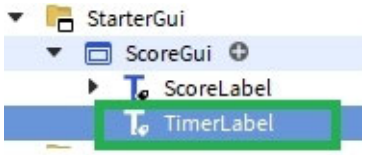
In the Explorer panel, click once on StarterGui |
In the top left menu panel, click on the icon Text



This will create a next Text Label element in the top left corner.

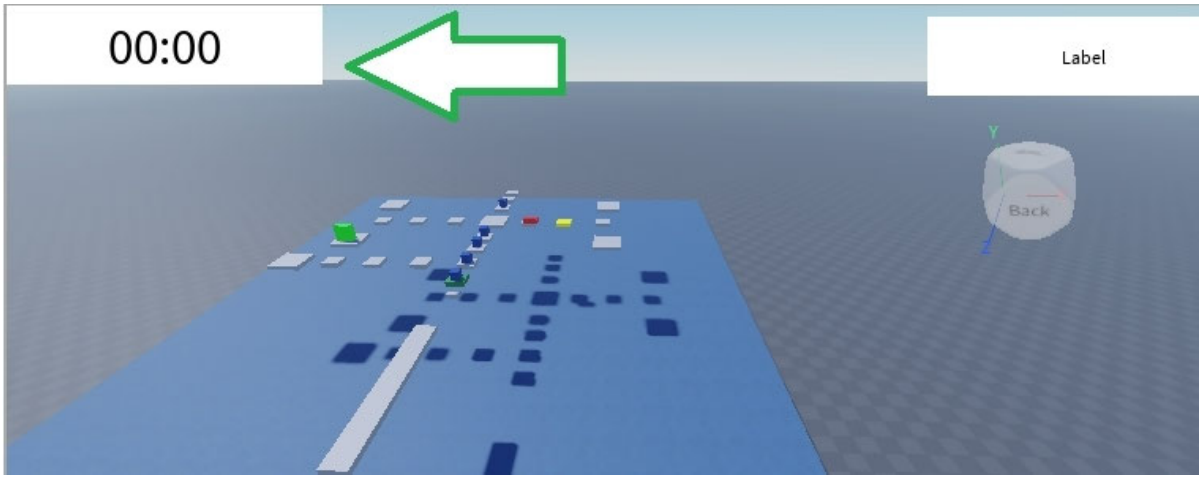


Using the change its name to



Using the Property window, customize the TextColor and Font properties to fit the theme of your game. For example, in the next figure, the font size | was changed to 40 and the text | Content to 00:00

If you can’t modify the Text Content directly in the Properties window, you can double-click on the TextLabel in the 3D Viewport and modify the text there instead.



Now that the timer has been created, we will create a script that will decrease the value of that timer, up until it reaches when this happens, we will teleport the player back to its original position, and that position will depend on the level achieved by the player (i.e., the SpawnPosition object for level1 or the startingPosition object for

Please right-click on the object
Insert a new script Object |
Rename this script
Open the script and replace the existing code with the following:

---

local timeLimit = 20

local currentTime = timeLimit

```
function updateTime (player)

Added, Timer Starting")

leaderstats = player:WaitForChild("leaderstats")

level = leaderstats:WaitForChild("Level")

Wait until PlayerGui is available

playerGui = player:WaitForChild("PlayerGui")

scoreGui = playerGui:WaitForChild("ScoreGui")

timerLabel = scoreGui:WaitForChild("TimerLabel")

currentTime > 0 do

-- Countdown every second

= currentTime - 1

" .. currentTime)
```

= "Time: " .. currentTime .. "s"

(currentTime <1) then -- Reset timer

= timeLimit

end

game.Players.PlayerAdded:Connect(updateTime)

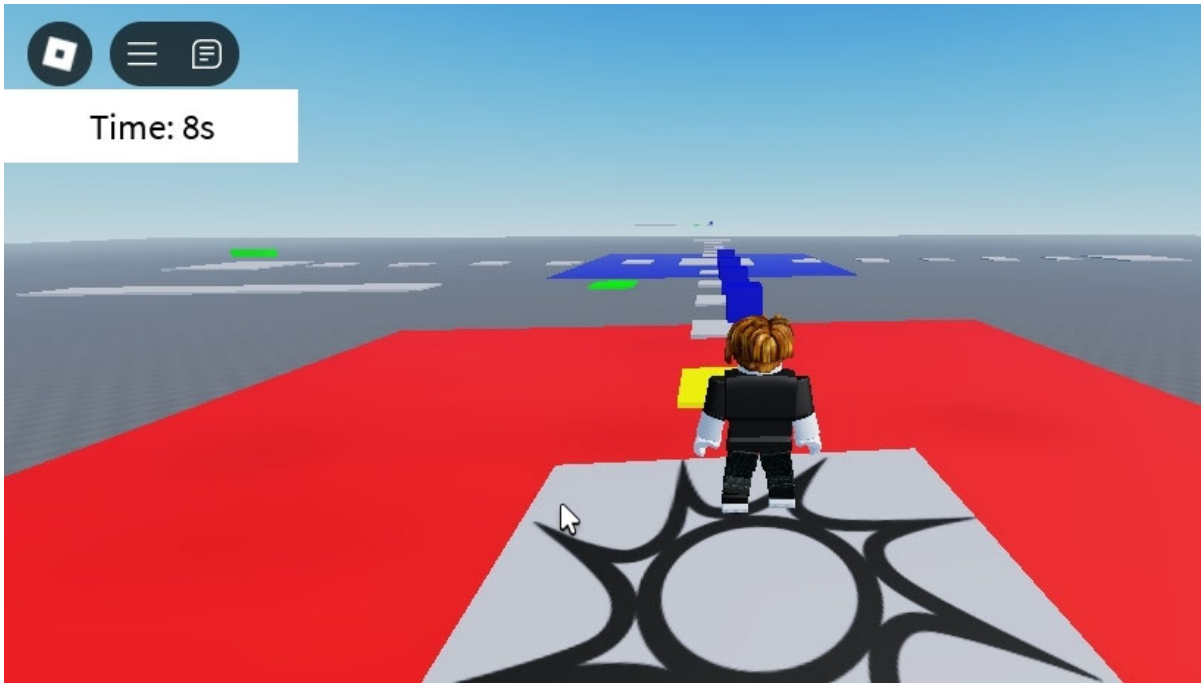
In the previous code, we do the following:

We set an initial timeLimit and define currentTime to store the remaining time.
We create function called updateTime that will be called when the player joins the game.
We refer to the object and timerLabel and wait until there are available.
We access the player's leaderstats and retrieve the
The while loop runs as long as currentTime is greater than 0, decrementing it by 1 every second.
The TimerLabel in the player's ScoreGui is updated with the remaining time.

The timer is reset to the timeLimit after it has reached 0.

Note that we have set the time to 20 seconds for testing purposes here, but for the game it will be set to 60 seconds.

You can now run this code, and check that the timer is displayed onscreen and that it decreases over time.



Now that we know the timer is working, we need to ensure that if it reaches 0, the player is moved back to its initial position:

Please add the following code after the statement that updates the score in the GUI (new code in bold):

---

player.PlayerGui.ScoreGui.TimerLabel.Text = "Time: " .. currentTime .. "s" -- Update timer display

if currentTime <= 0 then

Determine the respawn point based on player's level

is Up. Level = "..level.Value)

level.Value == 1 then

spawnPosition = workspace.SpawnLocation.Position + Vector3.new(0, 1, 0) -- 1 meter above SpawnLocation for level 1

startPosition = workspace.levels.level2.startingPosition.Position + Vector3.new(0, 1, 0) -- 1 meter above startingPoint for level 2

= timeLimit -- Reset timer

end

In the previous code, we do the following:

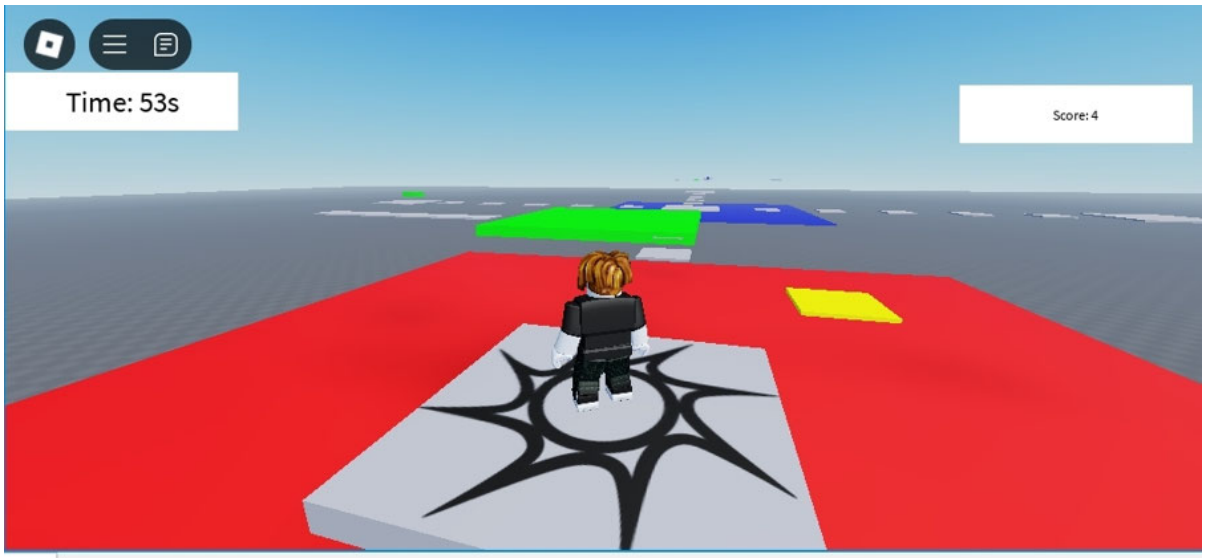
We check if currentTime is less than or equal to indicating the time is up.
We print the player's current level using level.Value for debugging.
If the level is 1, we teleport the player 1 meter above SpawnLocation in Level
If the level is greater than 1, we teleport the player 1 meter above startingPosition in Level
After the teleportation, we reset currentTime to the original restarting the countdown.

To have enough time to reach you may set the time limit to 60 seconds in the script.

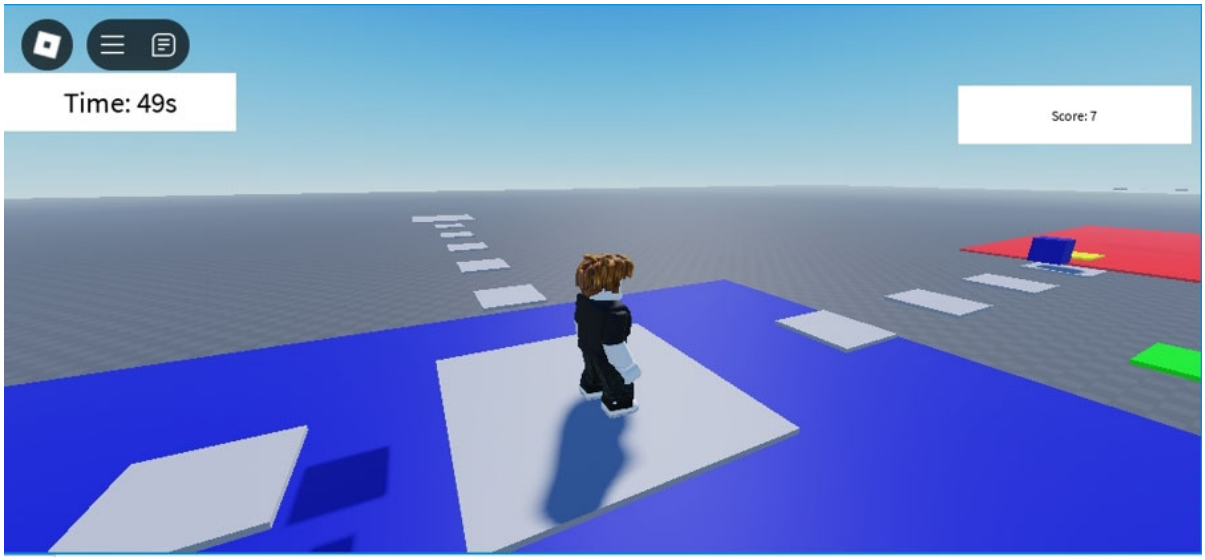
You can now play the game and check that the code works:

In move to a platform, away from the spawning point, and wait until the time is up; you should see that the player is automatically moved back to

the starting point.



In from level1, move to the check point; after being teleported to move to a platform, away from the starting point, and wait until the time is up; you should see that the player is automatically moved back to the starting point.



OceanofPDF.com

Creating a Starting Screen

At this stage, we have a working game where the player can transition from the first level to the second one, and both levels have a time limit, beyond which the player will be taken back to the start of the level.

The only things we need to complete the game is:

A splash screen that explains the goal of the game and a Start button that the player can press to start the game.
Additional conditions in both the first and the second level: collect 4 boxes in the first level to be able to move to the next level, and have a score of 10 or more in the second level to win the game.
Messages displayed in the second level that indicates whether the player has a score that is high enough, after reaching the checkpoint
An end screen that congratulates the player on achieving to complete the game.

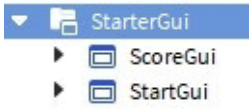
So, in this section, we will focus on the splash screen that will consist of a text to describe the objectives of the game and a button to start the game; as soon as the player presses the button, s/he will start the game and the splash screen will disappear.

First, let’s create the interface for the splash screen.

In the right-click on the object StarterGui and select Insert Obejct |

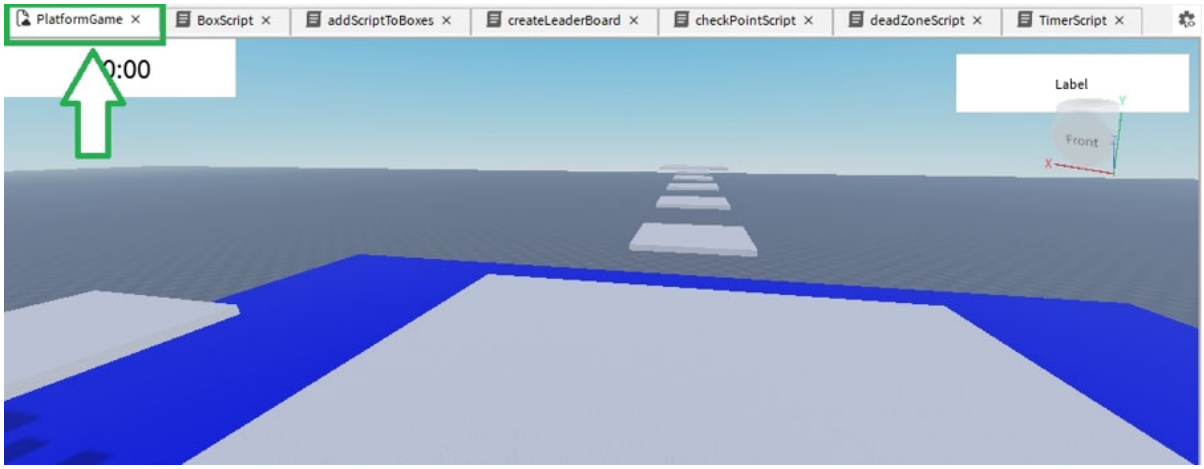
This will create a new object called as a child of the object

Rename this new object



Select the object

Click on the tab for the



Using the tools in the top left corner, select the Frame tool.



This will add a Frame object to the Explorer and a white rectangle to the view, as per the next figure.



Move this object to the center of the screen and modify its size as per the next figure, using its handles.



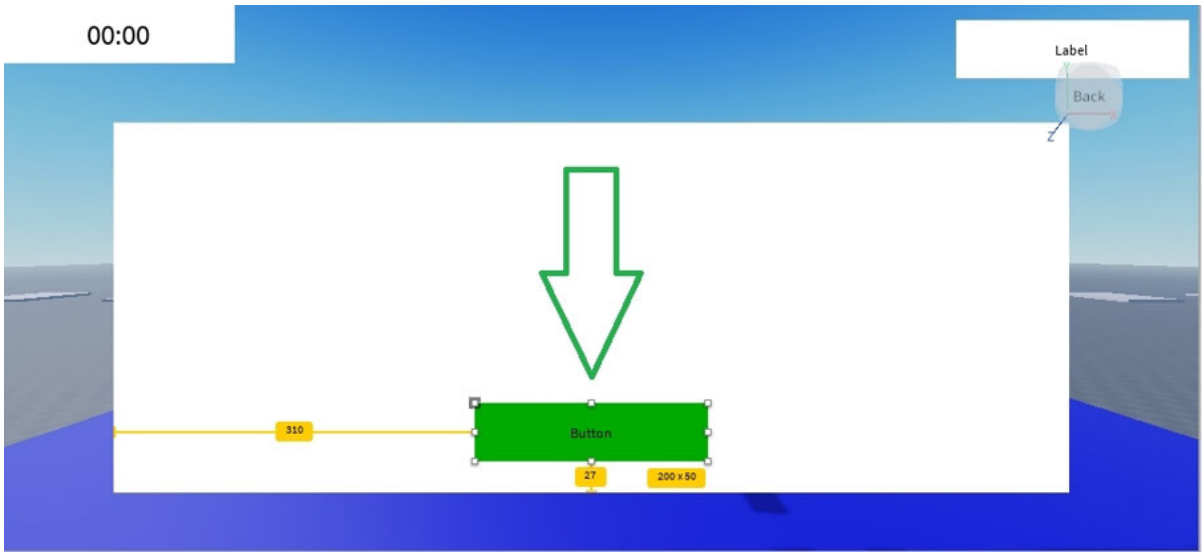
Next, well create the button and the instructions:

Right-click on the object and select: Insert | Object |
This will create a new button as a child of the object
Select that button.

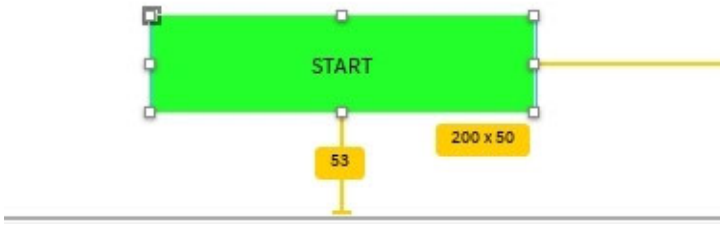
Using the Properties window, change its background color to green |

Data	
Archivable	☑
AbsolutePosition	0, 0
AbsoluteRotation	0
AbsoluteSize	200, 50
Active	☑
AnchorPoint	0, 0
AutoButtonColor	☑
AutomaticSize	None
BackgroundColor3	[35, 255, 42]

Move the button to the bottom of the frame.

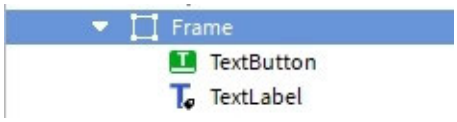


Double-click on the button, and change its label to

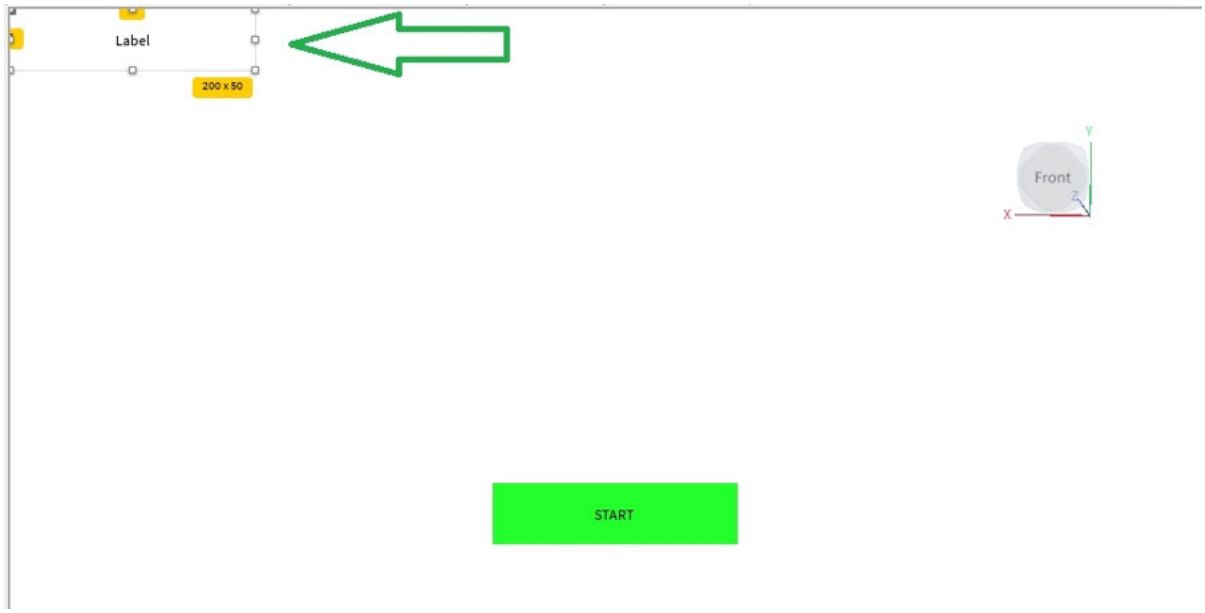


Once this is done, we just need to add some instructions for the player.

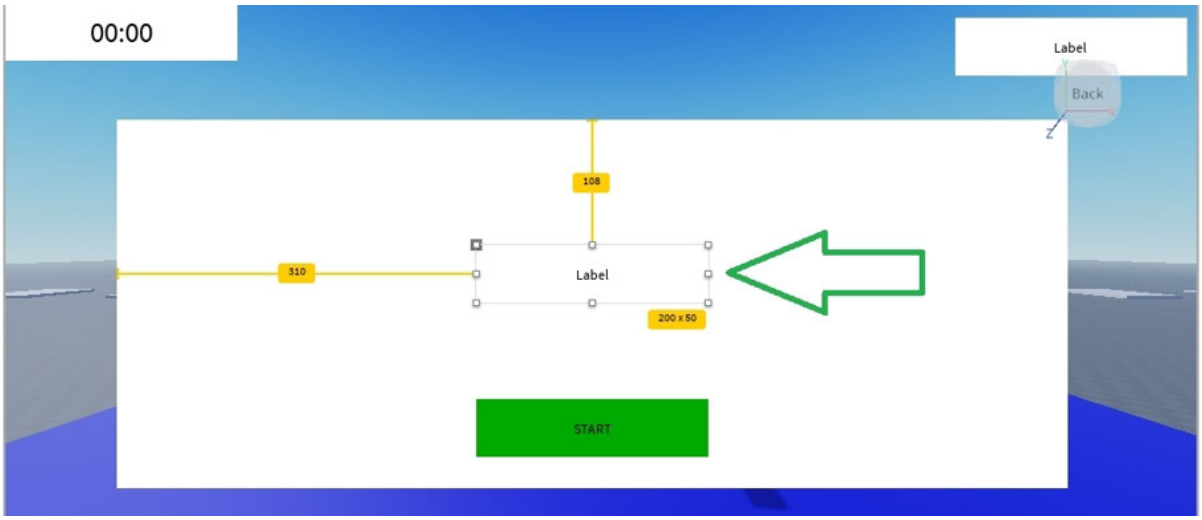
In the right-click on the object and select: Insert | Object | Text
This will create a new object called TextLabel as a child of the object



You should also see a white rectangle in the top-right corner of the screen.



You can move this text label to the center of the screen.



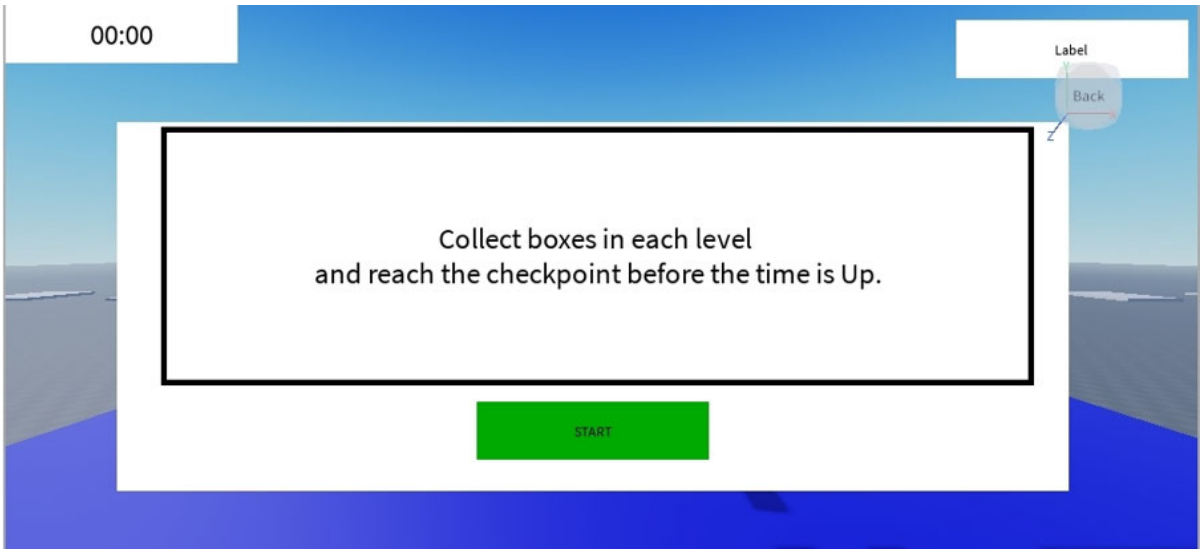
Using the Properties tab, change its size | to (500, change its font type | Font to a font of your its font | Font to and then change the label to boxes in each level and reach the checkpoint before the time is
You can also change its border size | to 5px.

Data	
BorderColor3	[0, 0, 0]
BorderMode	Outline
BorderSizePixel	5

If the text is too long, and you would prefer it to be on multiple lines, you can do so by typing your text (with carriage return) in a text editor of your choice (for example text pad).

```
1  Collect boxes in each level
2  and reach the checkpoint
3  before the time is up.
```

You can then paste that text in the attribute Text | Text for the Text



Now that we have created the interface, we just need to create code that will process the click on the button and also hide this interface (i.e., the frame object and its children object: the text for the instructions and the button) so that the player can play the game.

In the right-click on the button that you have created and select Insert Object | Local

---

Note that we use specifically a local script here to handle the clicks on the button; This because local scripts are used for client-specific operations like handling GUI interactions (including clicks), because they can directly access the PlayerGui object. Regular scripts, running on the server, cannot interact with player-specific GUI elements. Button clicks and similar actions are player-side events, so they must be handled client-

side with a local script. This ensures efficient interaction, limited to the specific player.

This will add a local script to that object.

Please open the script and replace the code already present in the script with the following:

```
local startButton = script.Parent

function hideStartGui()

= false

end

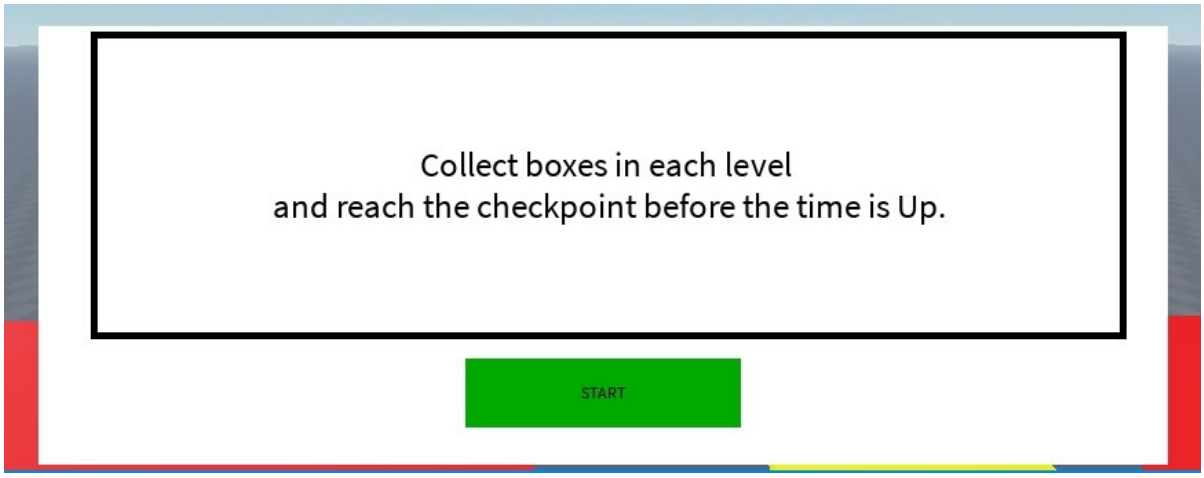
startButton.MouseButton1Click:Connect(hideStartGui)
```

In the previous code:

We refer to the object attached to the script (i.e., the button).
We create a function called

When this function is called, it hides the parent of the button (i.e., the Frame object).
Finally, we connect the event MouseButton1Click (i.e., left button click) to the function

You can now play the game; you should see the splash screen, and that whenever you click on the Start button, this screen disappears, so that you can start playing the game.

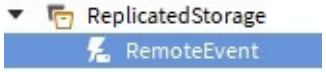


At this stage, you may notice that the time has already started, and we may need to change our code so that the timer only starts (or resets), when we press the Start button.

For this purpose, we will create a custom event that corresponds to the button being pressed, and when this event is fired, we will then reset the timer.

First, let’s create this event

In the right click on the object Replicated and select: Insert | Object | Remote
This will create a new remote event as a child of the object Replicate



Please rename it

Remote events need to be created in ReplicatedStorage because it is one of the few shared services accessible by both the client and the This allows local scripts (running on the client) and scripts (running on the server) to communicate efficiently. By placing the RemoteEvent in both sides can easily access it to send and receive events, facilitating client-server interaction for tasks such as sending player actions or requesting data from the server. In our case, we're firing the RemoteEvent from a LocalScript when a button is clicked, so that a server-side script can reset the timer. Remote events are essential for this because the button click happens on the while the timer logic is handled by the Since the client and server can’t communicate directly, the RemoteEvent allows the client (via the button click) to send a message to the server, instructing it to reset the timer. This ensures client-server synchronization for time management.

Now that we have created the event, we just need to do two things: to fire this event whenever the button has been pressed, and to then catch (or

handle) this event from the script and then reset the timer accordingly.

First let’s write the code that fires the event:

Open the local script attached to the START button.
Modify the code as follow (new code in bold):

---

local startButton = script.Parent

local resetTimerEvent =
game.ReplicatedStorage:WaitForChild("ResetTimer")

function hideStartGui()

= false

end

startButton.MouseButton1Click:Connect(hideStartGui)

In the previous code, we do the following:

We use WaitForChild to access the ResetTimer event from ensuring the event is available.
We fire the ResetTimer event to reset the timer when the function hideStartGui is called.
We connect the MouseButton1Click event to the function so that the function is called when the START button is clicked.

Next, we need to write the code in the script TimerScript to catch this event when it is fired and reset the timer accordingly:

Please open the script
Add the following code at the beginning of the script (new code in bold);

---

local timeLimit = 20

local currentTime = timeLimit

local resetTimerEvent =
game.ReplicatedStorage:WaitForChild("ResetTimer")

In the previous code, we create a variable resetTimerEvent that refers to the event that we have created in the folder Replicated

Add the following code (new code in bold):

---

function updateTime (player)

Added, Timer Starting")

leaderstats = player:WaitForChild("leaderstats")

level = leaderstats:WaitForChild("Level")

= 0 -- Reset the timer to zero when the start button is clicked

---

In the previous code,

We connect the OnServerEvent of the resetTimerEvent directly to an inline function without creating a separate named function. We used what is often called an online function.

An inline function is a function that is defined directly within another statement or event connection without being given a separate name. Its main benefit is improved readability and organization when the function is small and only used once, as it reduces clutter in your code. Inline functions also keep related logic in one place, making the code easier to understand.

The event triggers the inline function whenever the ResetTimer event is fired by the client (in this case, when the START button is clicked). Inside the function, we reset currentTime to 0, which ensures that the timer is reset to zero whenever the event is triggered.
This method eliminates the need for a separate function by defining it directly within the connection.

That’s it; you can now save your code and play the game; as you press the START button, you should see that this time the timer starts at 20s (or the time that you have set yourself in your code) and not 0.

OceanofPDF.com

Checking for Winning Conditions

So far, we have not required any particular condition for the user to move from the first to the second level; so in this section, we will add the condition of collecting 4 boxes in the current level to be able to move to the second level, and then a condition for the score to be higher than to win after completing the second level and reaching the corresponding check point.

For this purpose, we just need to modify the scrip called checkPointScript as follows (new code in bold):

---

function teleportPlayer(part)

player = game.Players:GetPlayerFromCharacter(part.Parent)

player then

Score is: "..player.leaderstats.Score.Value)

(player.leaderstats.Score.Value >= 4) then

= player.leaderstats.Level.Value + 1

("You Need to collect more boxes")

end

In the previous code:

If the player has collided with the checkpoint, we print the current score in the Output window.

We then check whether the score is greater than 4 and if this is the case, we move the player to the second level and we increase the level number, as we had done previously.
Finally, if the player has not collected enough boxes, we print a message in the Output window that indicates that the player has not collected enough boxes

You can now save your code and play the game; you can first try to reach the checkpoint in the first level, without having collected any boxes, and you should see that the player will remain in the first level and that the following message should be displayed in the Output window.

You Need to collect more boxes

You can then try to collect 5 boxes and then reaching the checkpoint, the player should then be teleported to the second level.

OceanofPDF.com

Displaying Messages to the user

In the previous section, we have managed to add a condition for the player to move to the second level (i.e., collecting four boxes or more) and a message displayed in the Output window, in case the player did not fulfill that condition upon colliding with the checkpoint; however the message will not be seen by the player, and a better solution would be to display it onscreen (as part of the game's user interface), as opposed to the Output window.

So, we will display these messages onscreen using the following steps:

Creating a text label that will contain the message for the user.
Hiding this text label at the start of the game.
Activating this text label when the player collides with the checkpoint without fulfilling the requirement to move to the second level (i.e., less than 4 boxes collected).
Displaying the message for 3 seconds, and then hiding it.

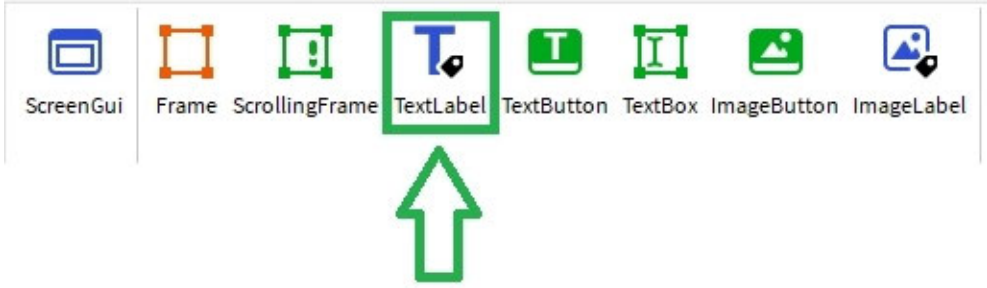
First let's create a text label to be used to display a message to the user.

Right-click on the object and select: Insert Object |
This will create a new object rename it
Switch to the

Using the top menu, click on the TextLabel button

If the UI panel below doesn’t appear, you can click on a different object in the and then click again on the object

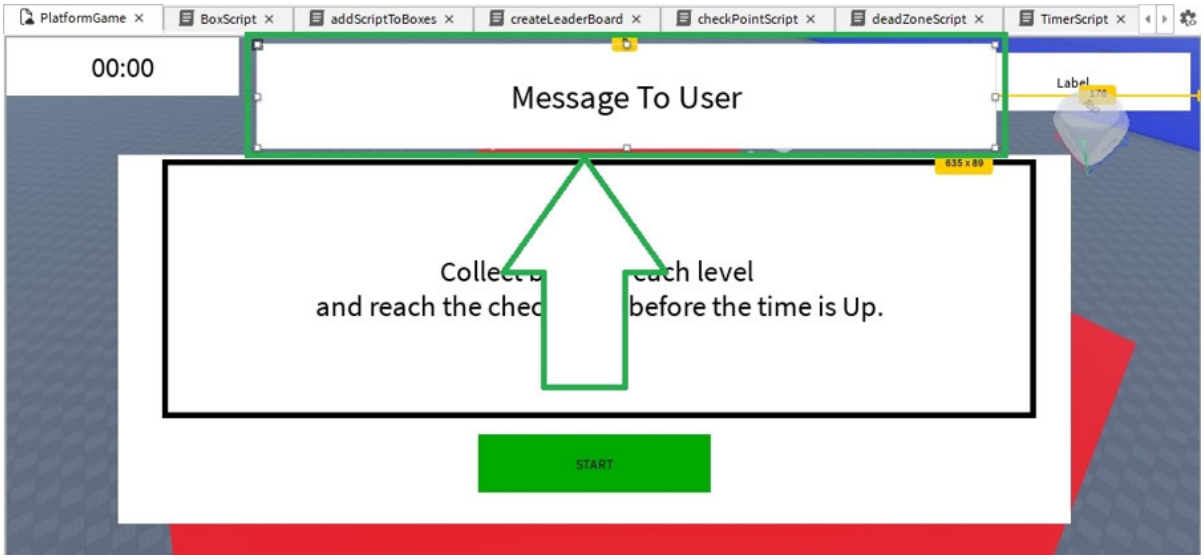
---



This will create a new TextLabel object that is also visible in the explorer.



Move the text label so that it is located in the upper part of the screen; modify its text to To



Once this is done, we just need to make sure that this message is hidden at the start of the game:

Right-click on the Text Label that you have just created and select: Insert | Object |
This will create a new script.
Open that script and replace the existing code with the following code:

```
script.Parent.Visible = false
```

In the previous code, we ensure that the text label is hidden at the start of the game.

The next and last step is to display this message for a few seconds whenever the player collides with the checkpoint in the first level without having collected enough boxes:

Open the script
Modify the script as follows (new code in bold):

---

function teleportPlayer(part)

player = game.Players:GetPlayerFromCharacter(part.Parent)

player then

Score is: "..player.leaderstats.Score.Value)

(player.leaderstats.Score.Value >= 4) then

= player.leaderstats.Level.Value + 1

= true

= "You Need to Collect more Boxes"

= false

end

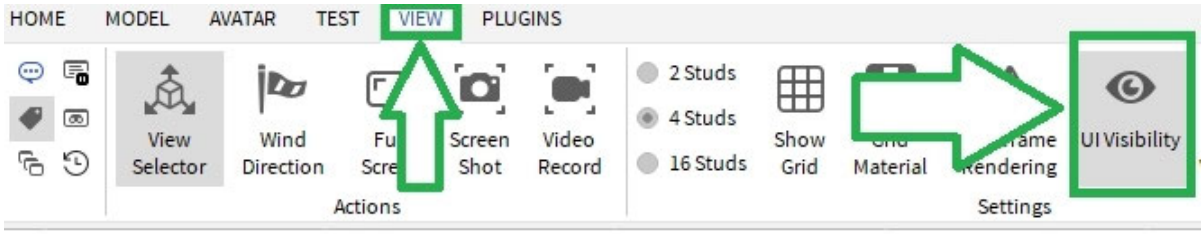
script.Parent.Touched:Connect (teleportPlayer)

In the previous code:

If the player hasn’t collected enough boxes yet, we display the text label that we created earlier.
We set the text of this text label to Need to Collect more
We then wait for 3 seconds, so that the player has enough time to see the message before it disappears
We hide the message.
You can now play the game and check that the message is displayed if you reach the checkpoint without enough boxes collected.

Note that if you would like to modify your environment and hide the UI in the meantime, you can temporarily hide/display the UI (i.e., frames,

buttons, labels, etc.) by pressing the button View | UI





OceanofPDF.com

Completing the End screen

In the previous section, we created a condition for the player to progress to the second level. In this section, we will apply a similar principle to check if the player has a high enough score upon reaching the checkpoint in the second level. If the score is too low, a message will inform the player to collect more boxes. If the player has collected enough boxes, the end screen will appear, displaying a congratulatory.

First, let's create the user interface that will be displayed when the player completes the second level. In the Explorer panel, right-click on and select Insert | Object | This will create a new object, which you can rename



One this is done, we will create the interface that will consist of a frame, a Congratulations text, and a button that the player can press to restart the game, similarly to the next figure.



Let’s create this interface:

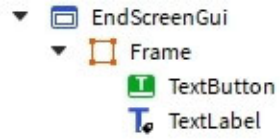
Right-click on the object select Insert | Object | This will create a new frame object as a child of the object



You can move the frame to the center of the screen, change its size | to (800,
Change its border | to black and its border size | to

We can now add the text label and the button:

Right click on the object Frame that you just created, and select Insert | Object | this will create a new TextLabel object as a child of the object
Right click on the object Frame that you just created, and select Insert | Object | this will create a new TextButton object as a child of the object



Change the color of the button | to and its label | to “>> RESTART
Change the label of the TextLabel object | Content to you can also change the font | Font I used the font bangers.



So, at this stage, the interface that you have just created for the end of the game should look like the next figure.



Since we want this interface to be hidden at the start of the game, we will create a script attached to the frame that will hide it accordingly.

Right-click on the object Frame that you have just created and select: Insert | Object | this will create and attach a new script.
Open that script and replace the existing code with the following:

```
script.Parent.Visible = false
```

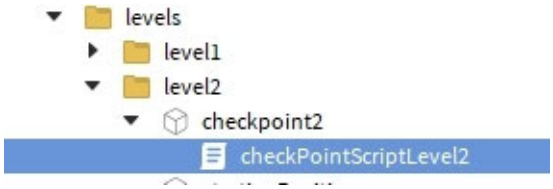
In the previous code, we hide the parent of the script, which the object called since the button and text label are children of the object they will also be hidden at the start of the game.

You can now play the game and ensure that the end screen is hidden at the start.

Once this is done, we need to handle collision between the player and the checkpoint (that we yet need to create) in the second level; for this, we will modify the script that is already connected to that checkpoint, to include the conditions needed to complete the level

Create a new part in in the right-click on the folder then select Insert | Object |

Rename it change its color to green and place it above a platform in
Add a script to that object: Right-click on the object and then select Insert | Object |
Rename this script



Open that script and replace the existing code with the following:

---

function checkLevelComplete (part)

player = game.Players:GetPlayerFromCharacter(part.Parent)

player then

(player.leaderstats.Score.Value >= 8) then

= true

= true

= "You Need to Collect more Boxes"

= false

end

script.Parent.Touched:Connect (checkLevelComplete)

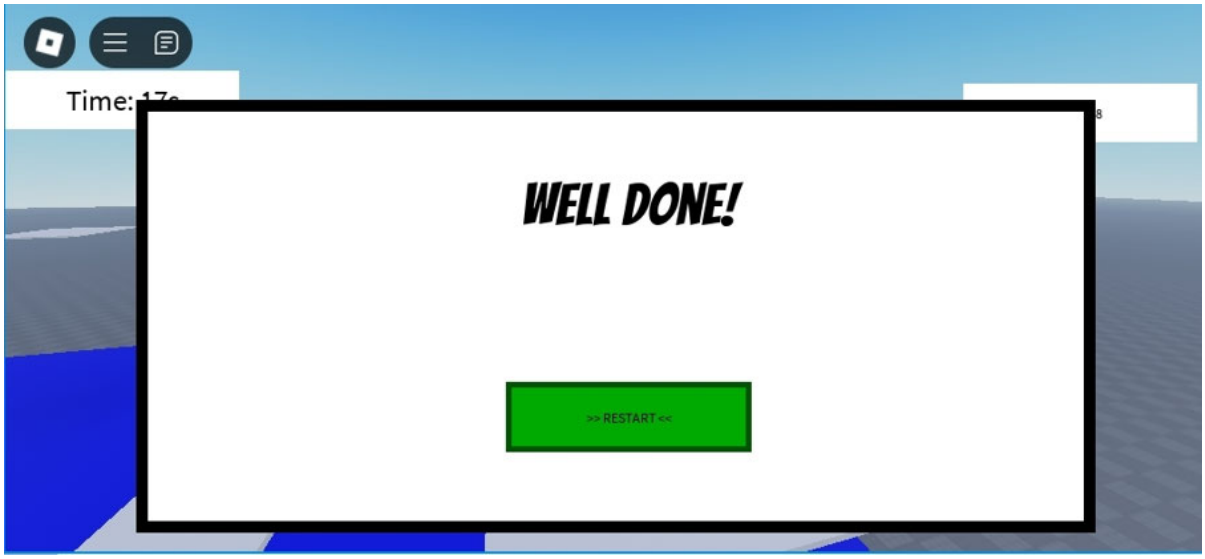
In the previous code:

As for the checkpoint in the first level, we create a function called checkLevelComplete that is called whenever the player collides with the checkpoint.
In case of collision, we check that it is indeed the player that collided with that checkpoint.
We then check whether the score is greater than 8. If that’s the case, we display the end screen interface (i.e., the message and the button to restart the game).

If the player’s score is not high enough, we display the message Need to Collect more for 3 seconds.

You can now play the game and test the code by reaching the second level and (1) reach the checkpoint in the second level with a low score (and you should see the message Need to Collect More in that case) and (2) reach the checkpoint in the second level with a high score of 9 or more (and you should see the message in that case).





So, all going well, you were able to see both messages.

So, at this stage, we just need to make sure that whenever the player presses the Restart button, that s/he is redirected to the first level, with a score of 0.

An easy way to implement this is to reload the entire game by teleporting the player back to the same place, effectively resetting everything to its initial state.

Theres several ways to achieve this:

Solution 1: Use the Teleport Service. This solution consists in using the Teleport service offered by Roblox, which reloads the entire game; however, this code will only run when the game has been published. To implement this solution, we could simply create and attach a new script to the RESTART button, and include the following code within:

```
local reStartButton = script.Parent

local TeleportService = game:GetService("TeleportService")

local placeId = game.PlaceId

function restartGame()
```

end

reStartButton.MouseButton1Click:Connect(restartGame)

In the previous code, upon clicking on the button, we just reload the current game. As mentioned above, this code will only be executed when the game has been published, and since it would be great to test your game before it is published, we can use a different solution.

Solution 2: In this solution, we clone a box and store the positions of all the collectible boxes in the game at the start. When the game is reset, we instantiate clones of these boxes at their original positions (which were saved in an array). Additionally, the player’s score is reset to 0, and the level is set back to 1. This method allows us to reset the game without reloading it entirely, simply by replacing the boxes in their initial locations..

So, will let’s implement the second solution:

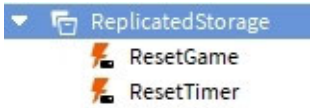
Create a Box Start by selecting one of the collectible boxes, and clone it to use as a template for future instantiations.

Store Box Write a script that loops through all tagged boxes at the start of the game, saving their original positions and which level they belong to in a table.
Create a Game Reset Define a RemoteEvent called ResetGame in ReplicatedStorage to handle resetting the game.
Implement the Reset Game Write a function that, when triggered, resets the score, sets the player's position to the start of level 1, and respawns the boxes using the saved positions and the box template.
Attach the Reset Connect the RemoteEvent to the function so that when the event is triggered, the game is reset.
Link the RESTART button to the Event: Create a script attached to the RESTART button that will fire the ResetGame event when clicked, ensuring the game reset function is executed.

Place Scripts in the Right Add the box management script to a ServerScriptService and ensure the ResetGame event is connected correctly to handle player input for resetting the game

With this in mind, let's get started:

In the Explorer, right-click on ReplicatedStorage and select Insert | Object | Remote This will create a new event that you can rename



Create a new Local Script attached to the RESTART button and add this code to the script:

```
local restartButton = script.Parent

local resetGameEvent =
game.ReplicatedStorage:WaitForChild("ResetGame")

function onRestartClick()

= false  -- Hide the button after click

= false;

-- Fire the event to reset the game

end

restartButton.MouseButton1Click:Connect(onRestartClick)
```

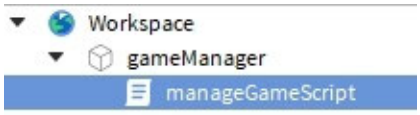
In the previous code we:

Assigned the restartButton variable to the script's parent, which is the UI button that triggers the reset.
Referenced the ResetGame event from ReplicatedStorage using WaitForChild to ensure it's loaded.

Created the onRestartClick function, which hides the restart button and its parent UI frame when clicked.
Fired the ResetGame event to initiate the reset of the game on the server.
Connected the MouseButton1Click event of the restart button to the onRestartClick function, ensuring the game resets when clicked.

Now that we have linked the RESTART button to the event we will create a game manager that will handle the storing and spawning of blue boxes, at the start and when the game is reset.

Please create a new part and rename it
Set its Transparency to its attribute CanCollide to
Create a new Script attached to this object, and rename the script



Open the scriptand add this code to it:

```
local CollectionService = game:GetService("CollectionService")

local ReplicatedStorage = game:GetService("ReplicatedStorage")

local ResetGameEvent = ReplicatedStorage:WaitForChild("ResetGame")
```

```
-- Table to store original box positions and their levels

local boxData = {}

local boxTemplate = nil  -- This will store the template box
```

In the previous code we:

Use CollectionService and ReplicatedStorage to interact with tagged objects and storage for the ResetGame event.
Create a boxData table to store the original positions and level data of the collectible boxes in the game.
Initialize boxTemplate as which will later store a cloned version of a collectible box that can be reused when resetting the game.
Use WaitForChild("ResetGame") to ensure the reset event is loaded from ReplicatedStorage before it’s used.

Now that we have defined the necessary services and variable to store the location of our boxes, we can create a function that will store information about these boxes.

Please add the following code to the script:

```
function storeBoxData()

boxes = CollectionService:GetTagged("collectibleItem")

_, box in pairs(boxes) do

box:IsA("BasePart") then

not boxTemplate then

= box:Clone()

= ReplicatedStorage


levelName = box.Parent.Name

{position = box.Position, level = levelName})



end
```

In the previous code we do the following:

Define the function storeBoxData that will be in charge of saving information abou the collectible boxes.
In this function, we go through all the objects in the game with the tag
We clone these boxes and save the clone in
We also ensure that we only clone the first instance of a box (used as the template) once, not the clone with the code not boxTemplate

We also store information about this clone in a table i.e., parent and position).

Now that this function has been created, we will create a function that will respawn the boxes, and that will be called when we hit the RESTART button.

Please add the following code to the script:

---

function respawnBoxes()

counter = 0;

_, data in pairs(boxData) do

= counter + 1

newBox = boxTemplate:Clone()

= data.position

data.level == "level1" then

= workspace.levels.level1

data.level == "level2" then

= workspace.levels.level2

end

In the previous code we go through the table that we have created and filled earlier, and we use the information stored to instantiate next clones of the collectible boxes based on the data stored (i.e., parent and position). Each new instance is then placed accordingly, and attached to their respective parent (i.e., level1 or This allows us to respawn the boxes in their original locations when resetting the game.

Next, we will create a function that will effectively reset the game.

Please add the following code to the script:

---

function resetGame(player)

player then

= 0

= 1

spawnPosition = workspace.SpawnLocation.Position + Vector3.new(0, 1, 0)

end

In the previous code we do the following:

Check if we are dealing with the current player before continuing with the reset.
Call the function respawnBoxes() to respawn the collectible boxes back to their original positions.
Reset the player's Score and Level values to 0 and respectively, by accessing the leaderstats folder in the player object.
Define the player's spawnPosition as 1 meter above the SpawnLocation in the workspace.
Finally, move the player’s character to the spawnPosition by using SetPrimaryPartCFrame to teleport them there.

You may notice that while the function resetGame take a parameter, we have not passed a parameter to that function when colling it from the script attached to the RESTART button; this is because the player is automatically passed as the first argument by the system when you call FireServer() from a

Now that we have defined the functions to store the data and to resetteh game we just need to add the following code:

ResetGameEvent.OnServerEvent:Connect(resetGame)

storeBoxData()

In the previous code we do the follwoing:

We connect the ResetGameEvent to the function resetGame using This ensures that whenever the event is triggered from the client, the resetGame function will be executed on the server.
We call the function storeBoxData() to store the initial positions and levels of the collectible boxes at the start of the game. This step is essential for respawning the boxes later when the game is reset, ensuring they return to their original locations.

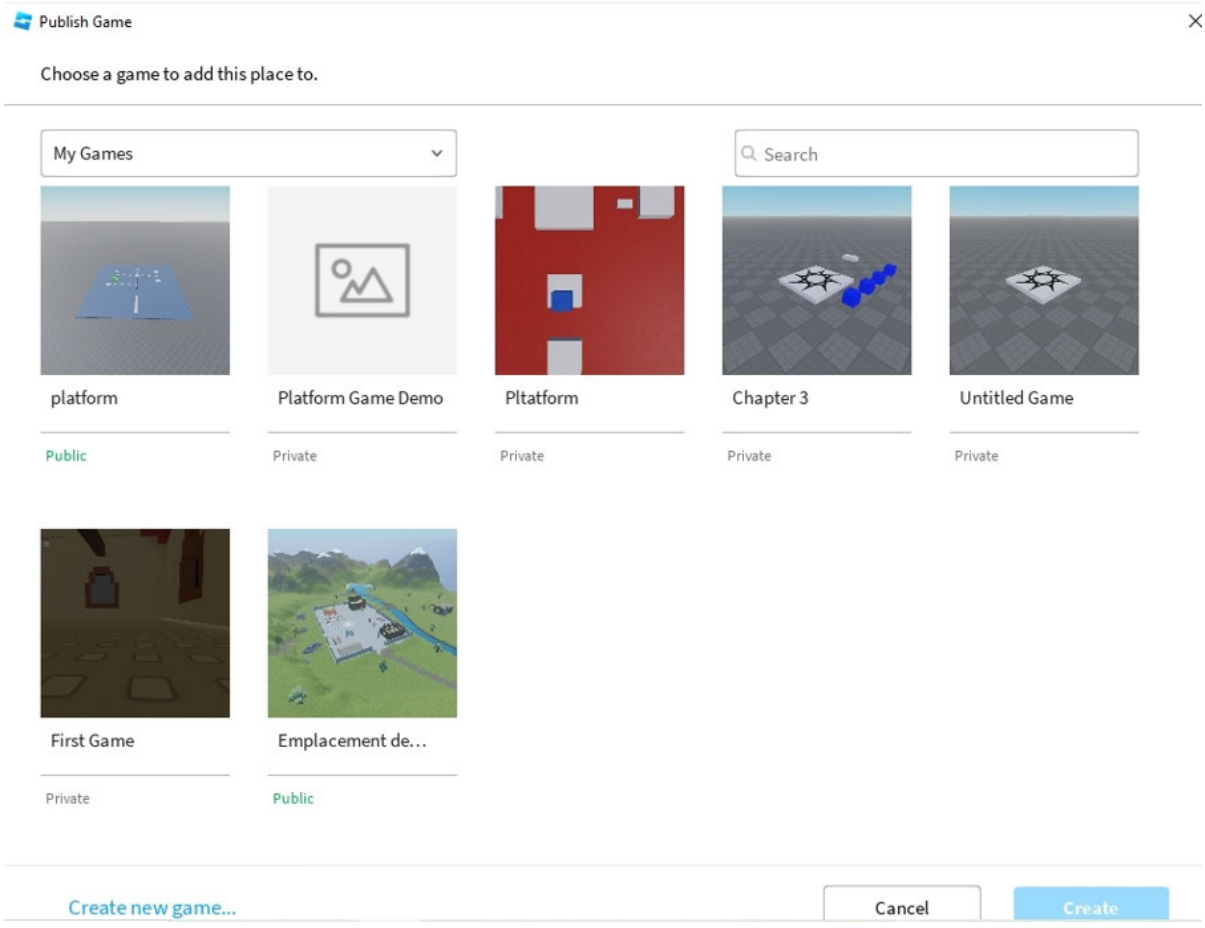
You can now save your code, play the game, reach the second level after collecting at least 8 boxes; the End screen should be displayed, and after clicking the RESTART button, the game should restart.

OceanofPDF.com

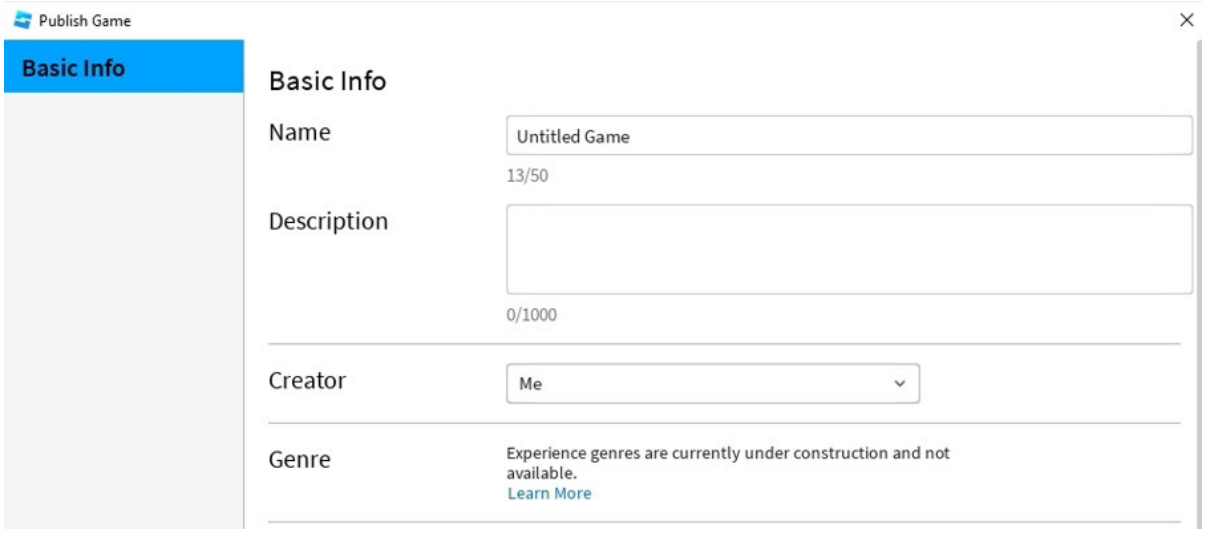
Publishing and Sharing the Game

Now that the game is complete, it is time to publish it and to share it with you friends:

In Roblox select File | Save to to save your game.
Then select File | Publish To Roblox As…
A new window will appear.



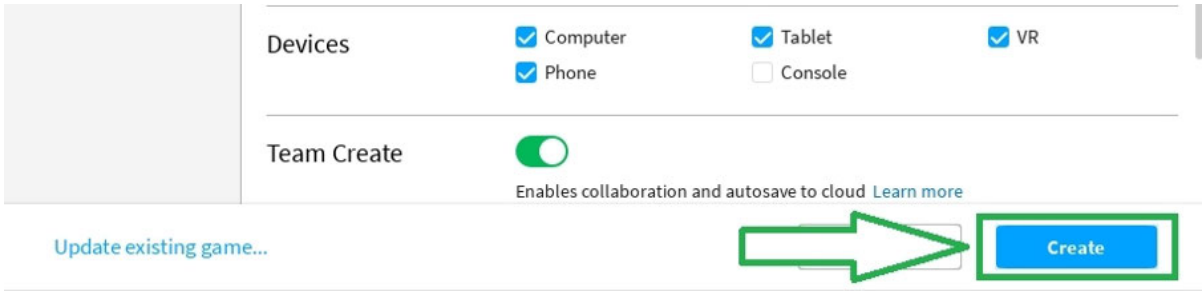
Press the option to new at the bottom of the screen.
A new screen with more information about your game will appear.



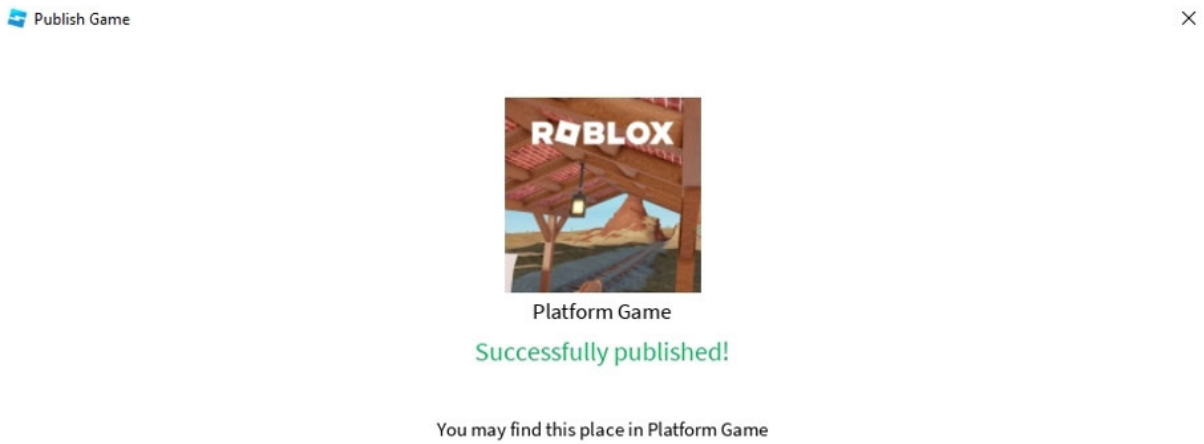
Give a name to your game as well as a description, leave the other options as default.



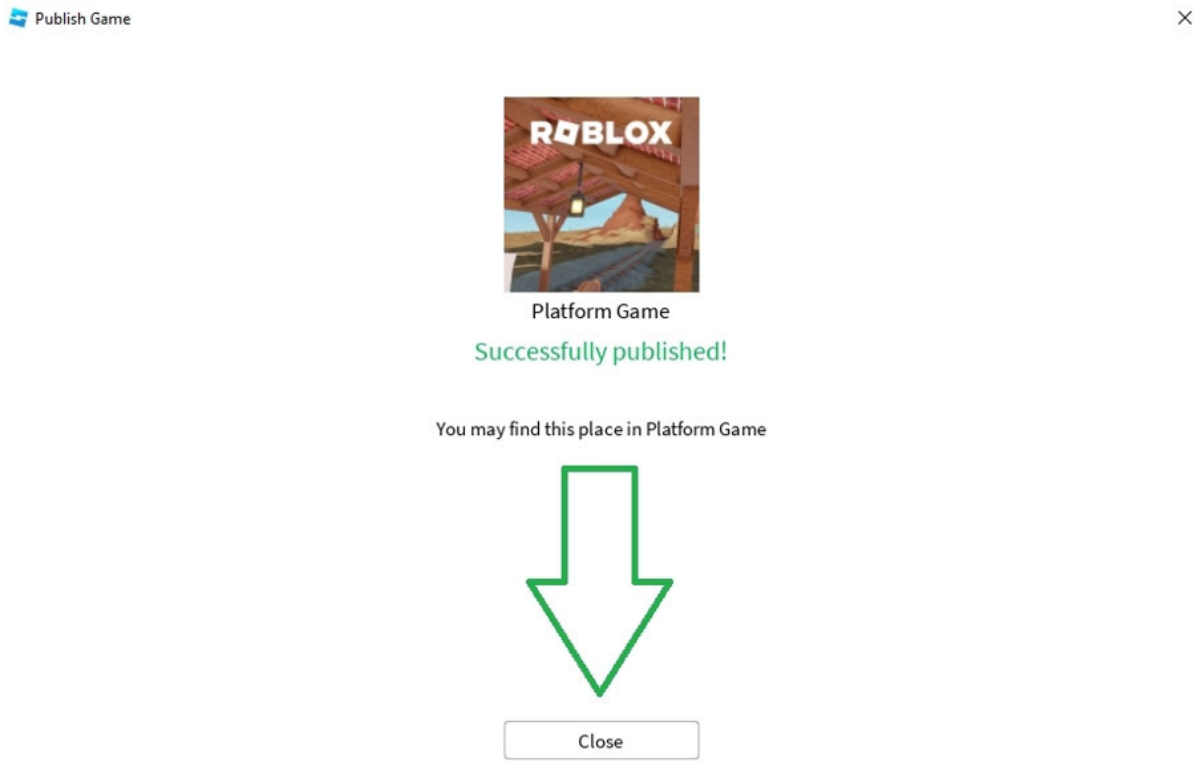
Once this is done you can press the button.



A confirmation screen should then appear, as per the next figure.

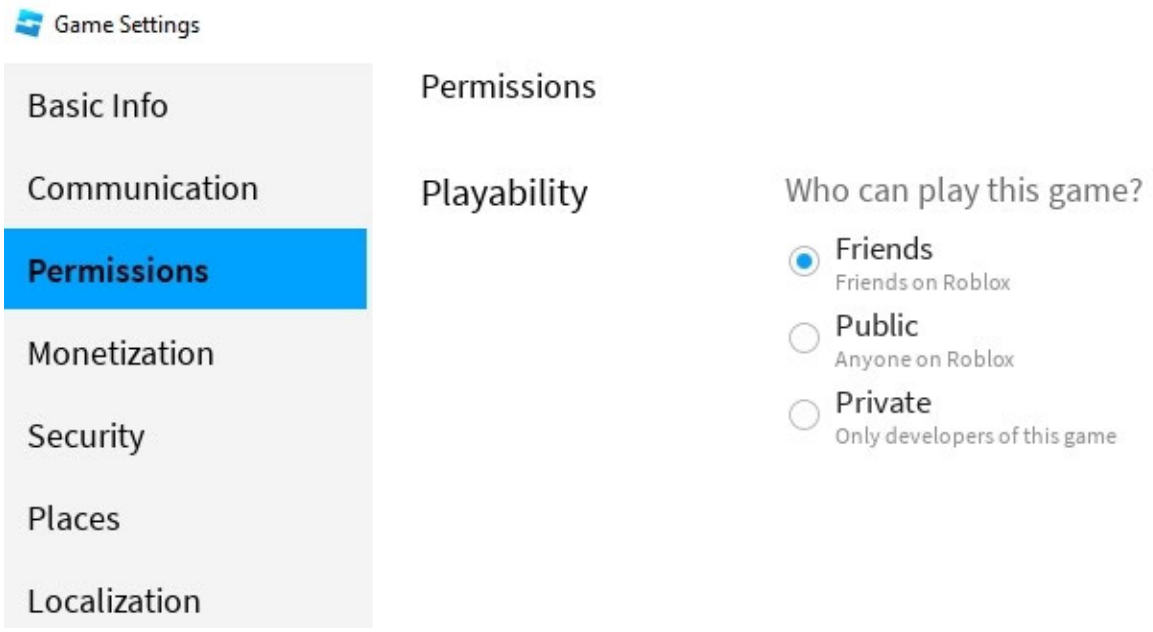


Click the Close button.



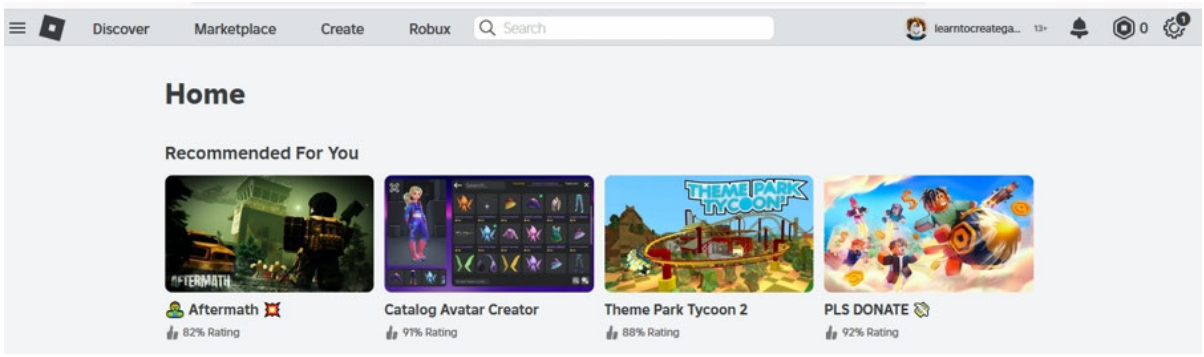
Before your friends can access this game, you just need to make sure that its access has been set-up accordingly.

In Roblox Studio, display the game view and then select File | Game Settings | Permissions and set the permission to Public or Friends as per the next figure, and press



Once this is done, you can access your game through Roblox and test it entirely:

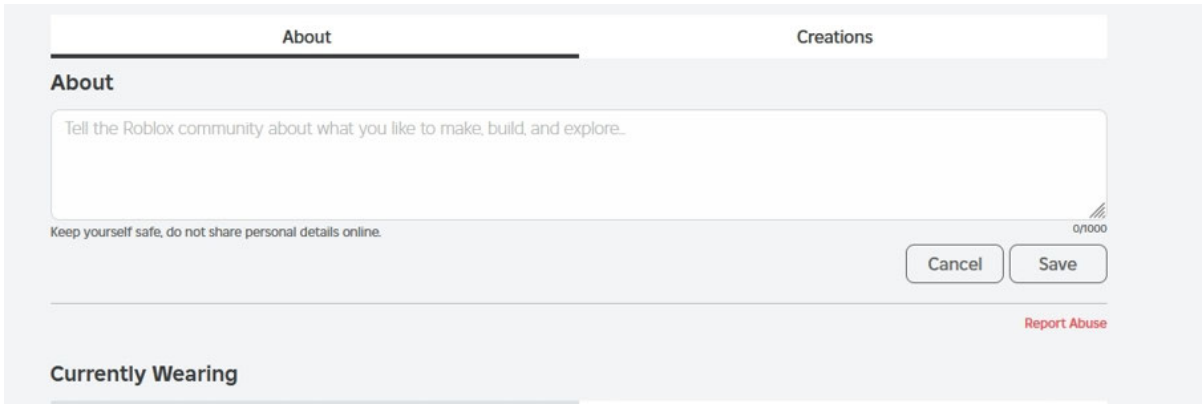
Open
Enter your login and password.



Click on your profile, as per the next figure.



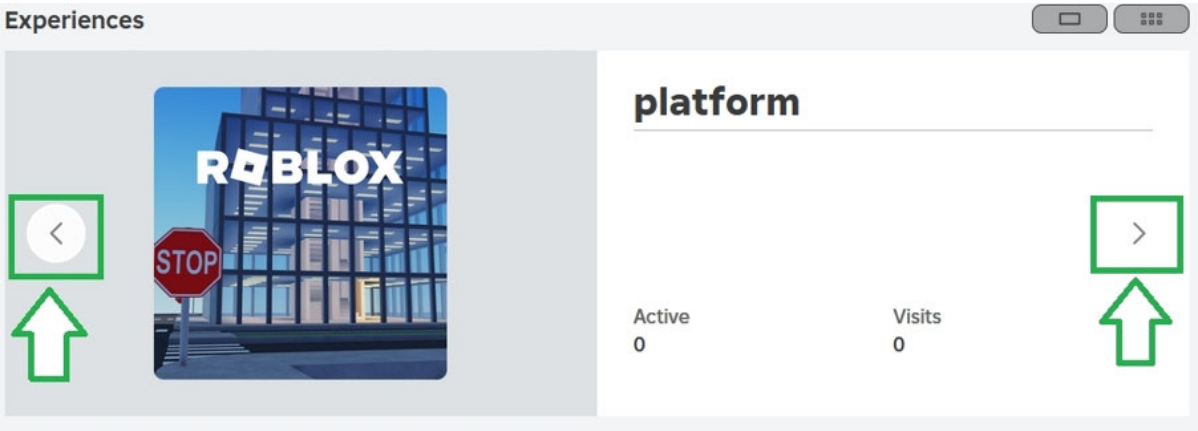
A new window will appear with two tabs: About and



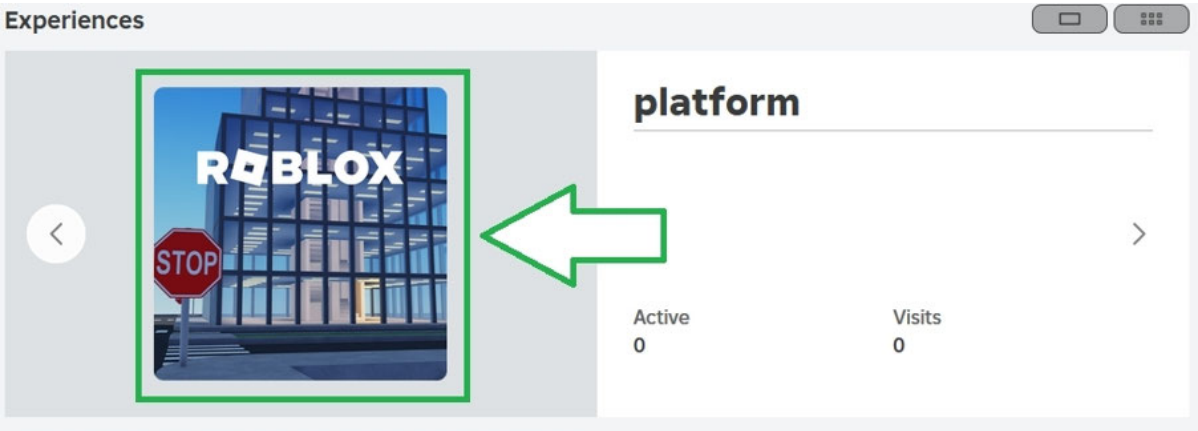
In the About window, you can scroll down and customize your character if you wish to do so, as per the next figure.



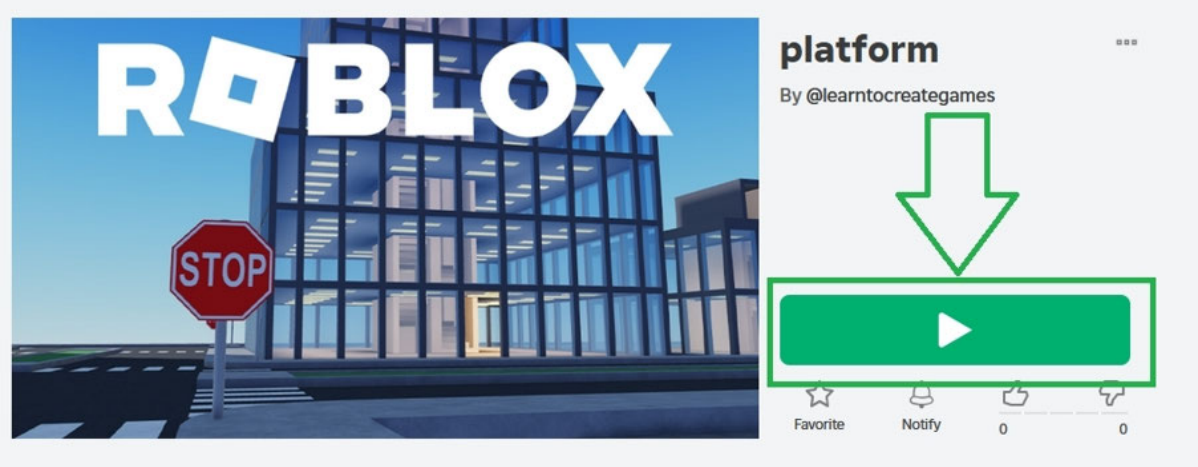
To access the game that you have published, click on the tab
Browse through the creations (i.e., the games that you have created so far) using the left and right arrows as per the next figure.



Once you have found the experience (i.e., the game) that you want to play, click on the corresponding logo.



This will open a new window with more details about the game, and you can press the Play button to start playing the game.



This should launch by default, if you have it already installed; and if you don’t, then a link to download the player will be provided. Once downloaded, you can launch the installer and then open the Roblox once it has been installed. As the Roblox Player opens, you will need to provide your login and password (the same login and password as for the Roblox website), you will then be able to start playing the game.



OceanofPDF.com

Summary

Great job! You've made incredible progress in this chapter, building on your knowledge by managing multiple levels, implementing a countdown timer, and ensuring that players face challenges under time pressure. You’ve structured your levels to create seamless transitions, set up triggers and win conditions, and learned how to reset the player when time runs out. These features add complexity to your game, making it more engaging and fun.

OceanofPDF.com

Quiz

Please guess whether the following statements are True or False (the solution is in the section Answers to at the end of the book).

You can use TextButtons to create clickable buttons in Roblox GUIs.
ScreenGui objects are automatically visible to all players when added to the
GUI elements are stored in StarterGui and cloned to each player’s
The TextLabel object can display both text and images.
You must use LocalScripts to control player-specific UI elements.
You can add interactive buttons using the Frame object alone.
TextBox objects allow players to input text, which can then be processed by the game.
ScreenGui elements scale automatically across different screen resolutions.
ImageLabel objects can only display static images and not animations.
You can create a custom HUD using GUI elements such as TextLabel and

OceanofPDF.com

Challenge

Try these challenges to put your newly acquired skills to the test:

Challenge Create a game with three levels where each level must be completed within 60 seconds. If the player runs out of time, they should be returned to the starting point of the level. Ensure that each level introduces new challenges to increase difficulty.
Challenge Implement a score system that tracks the player's performance across all levels. Add bonus points if the player completes a level before the timer runs out. Test the score display to ensure it updates correctly across levels.

OceanofPDF.com

8
Answers to Quizzes

OceanofPDF.com

Chapter 2

Can Roblox Studio be installed on both Windows and macOS? (True, Roblox Studio is available for both platforms.)
Is the Menu Bar located at the top of the Roblox Studio interface? (True, the Menu Bar is at the top for easy navigation.)
Can you find pre-made models in the Toolbox? (True, the Toolbox contains pre-made models for easy use.)
Is the Explorer panel used to manage game objects? (True, the Explorer panel is used to manage all game objects.)
Does the Properties panel allow you to change object properties? (True, the Properties panel lets you modify object attributes.)
Can the Select tool be used to move objects? (False, the Select tool is for selecting, not moving objects.)
Are the arrow keys used to navigate the workspace? (False, the arrow keys are not typically used to navigate the workspace.)
Can you customize the layout of your workspace? (True, Roblox Studio allows you to customize your workspace layout.)
Is it possible to save your custom workspace layout? (True, you can save your workspace layout for future use.)
Does the Move tool resize objects? (False, the Move tool is for moving objects, not resizing them.)

OceanofPDF.com

Chapter 3

The Move tool allows you to resize parts. (False, the Move tool is for moving parts, not resizing them.)
Grouping parts helps manage multiple parts as a single unit. (True, grouping simplifies managing multiple parts together.)
Anchored parts can move freely in the game. (False, anchored parts stay fixed in place.)
The Scale tool is used to change the size of a part. (True, the Scale tool is for resizing parts.)
You can apply textures to parts using the Toolbox. (True, the Toolbox offers textures and other assets.)
The Properties panel is where you modify part properties. (True, you adjust part properties in the Properties panel.)
Unchecking the CanCollide property prevents parts from colliding with others. (True, disabling CanCollide stops parts from interacting with others.)
Materials are applied in the Explorer panel. (False, materials are applied in the Properties panel, not Explorer.)
The Rotate tool is used to change the color of a part. (False, the Rotate tool only rotates parts.)
Parts cannot be grouped together in Roblox Studio. (False, you can group parts in Roblox Studio.)

OceanofPDF.com

Chapter 4

In Lua, the = operator is used for assigning values to variables. (True, the = operator is used for assignments in Lua.)
Lua tables always start their indexing at 0 by default. (False, Lua tables start indexing at 1 by default.)
You must declare the type of a variable when using Lua, just like in C#. (False, Lua is dynamically typed, so variable types aren't declared.)
In Lua, the local keyword limits a variable's scope to the block where it is declared. (True, the local keyword restricts scope to the block.)
Lua does not support multidimensional tables, so you cannot create grids or matrices. (False, Lua supports multidimensional tables.)
The Touched event in Roblox is fired when a player interacts with a GUI element. (False, the Touched event is triggered by physical part contact, not GUI interaction.)
In Roblox Lua, the Connect method is used to associate an event with a function. (True, Connect is used to link functions with events.)
Lua requires semicolons at the end of each statement, similar to C#. (False, semicolons are optional in Lua.)
In Lua, nil represents the absence of a value, similar to null in other languages. (True, nil in Lua is like null in other languages.)

In Roblox Lua, you can use the BindableEvent object to create custom events that other scripts can respond to. (True, BindableEvent allows custom events in Roblox Lua.)

OceanofPDF.com

Chapter 5

Variables in Lua can store different types of data like numbers, text, and objects. (True, Lua variables are flexible and can hold various types of data.)
The if statement is used to create loops in Lua. (False, if statements are for conditions, not loops.)
The Touched event is triggered when a part is clicked by a player. (False, the Touched event is triggered when a part is physically touched, not clicked.)
Lua is the programming language used in Roblox Studio. (True, Lua is the main scripting language in Roblox.)
Functions in Lua allow you to group code into reusable blocks. (True, functions are used to create reusable blocks of code.)
The Output window is used to change the color of parts. (False, the Output window is for debugging, not for modifying parts.)
The + operator in Lua is used for addition. (True, the + operator performs addition.)
Conditional statements allow your script to make decisions based on certain conditions. (True, conditional statements control decision-making in code.)
Loops in Lua are used to repeat tasks multiple times. (True, loops repeat code multiple times in Lua.)

Debugging involves fixing errors in your script. (True, debugging is the process of finding and fixing errors.)

OceanofPDF.com

Chapter 6

You can create a part in Roblox Studio using the Home tab. (True, the Home tab has tools for creating parts.)
The Touched event in Roblox is triggered when a player clicks on a part. (False, the Touched event is triggered by physical contact, not clicks.)
Vector3 is used in Roblox to represent 3D coordinates like position or size. (True, Vector3 is used for 3D coordinates in Roblox.)
The PlayerMovement script allows players to move their character using the arrow keys. (False, player movement is typically controlled with WASD or custom inputs.)
The Destroy method in Roblox permanently removes an object from the game. (True, Destroy removes objects permanently.)
A script attached to a part can detect when the part is touched by another object. (True, a script can detect touches if attached to a part.)
Collectible items are created by resizing a block and adding a destroy script. (True, collectibles often use a destroy script for collection.)
The ChangeState method is used to move parts around the game environment. (False, ChangeState is not used for moving parts; movement is managed by other methods.)
Interactive doors can be scripted to open when a player gets close. (True, doors can be scripted to open based on player proximity.)

The Insert Object menu in the Explorer panel is used to add scripts to objects. (True, the Insert Object menu allows adding scripts to parts.)

OceanofPDF.com

Chapter 7

You can use TextButtons to create clickable buttons in Roblox GUIs. (True, TextButtons are commonly used for clickable elements.)
ScreenGui objects are automatically visible to all players when added to the Workspace. (False, ScreenGuis are player-specific and must be in PlayerGui, not Workspace.)
GUI elements are stored in StarterGui and cloned to each player’s PlayerGui. (True, GUI elements are stored in StarterGui and cloned for each player.)
The TextLabel object can display both text and images. (False, TextLabels can display text, not images.)
You must use LocalScripts to control player-specific UI elements. (True, LocalScripts are required for player-specific interactions.)
You can add interactive buttons using the Frame object alone. (False, a Frame itself isn't clickable; you need buttons like TextButtons for interaction.)
TextBox objects allow players to input text, which can then be processed by the game. (True, TextBox objects are for user text input.)
ScreenGui elements scale automatically across different screen resolutions. (True, ScreenGuis adapt to various screen resolutions.)
ImageLabel objects can only display static images and not animations. (False, ImageLabels can display animated GIFs.)

You can create a custom HUD using GUI elements such as TextLabel and ImageLabel. (True, HUDs can be built using TextLabel and ImageLabel.)

OceanofPDF.com

9

Frequently Asked Questions

OceanofPDF.com

Getting Started with Roblox Studio

What is Roblox Studio, and how is it different from Roblox itself? Roblox Studio is the tool used to create and develop games, while Roblox is the platform where players play those games. Roblox Studio provides the environment and tools to design, build, and script games.

How do I install Roblox Studio, and is it free to use? Yes, Roblox Studio is free to use. You can download it from the Roblox website. Once installed, sign in or create a Roblox account to access the development tools.

What are the basic tools in Roblox Studio I need to know to start building? The key tools you'll use are the and Rotate tools to manipulate objects. Additionally, the Explorer and Properties panels are essential for organizing and modifying objects in your game.

How do I create and manage my first simple game in Roblox Studio? Begin by creating a new project, inserting parts into the workspace, and using scripts to add interactivity. Follow tutorials or use pre-made assets from the Toolbox to build your first game.

OceanofPDF.com

Scripting and Game Mechanics

What is Lua scripting, and why is it important in Roblox development? Lua is the programming language used in Roblox to script interactions, animations, and game mechanics. Without Lua, your game would lack interactivity and advanced functionality.

How do I create a simple script in Roblox Studio? Right-click on a part in the Explorer panel, select Insert Object | and start writing Lua code inside it to control what happens when the player interacts with that part.

What’s the difference between a LocalScript and a Script in Roblox Studio?Scripts run on the server and handle game-wide logic, while LocalScripts run on the client and are used for player-specific actions, like camera control and GUI interactions.

How do I create a simple scoring system for my game? Use a leaderstats folder with an IntValue named Score to store the player’s score. Update the score using a script when the player collects items or completes tasks.

How do I debug my scripts when something goes wrong?

The Output Window is your best tool for debugging. It displays errors and warnings generated by your scripts. You can also use print statements to track the flow of your code.

OceanofPDF.com

Section 3: Building and Object Manipulation

What's the easiest way to manipulate parts and objects in Roblox? Use the and Rotate tools from the Home tab to adjust the position, size, and orientation of parts in your game. The Properties panel lets you fine-tune settings like transparency, color, and material.

What is anchoring, and why is it important? Anchoring locks a part in place, preventing it from being affected by physics like gravity or player interaction. It's crucial for objects like walls, platforms, and scenery that need to stay in place.

How do I make an object disappear after the player interacts with it? Use a script with the Touched event to detect when the player touches the object, and then call the Destroy() function to remove it from the game.

What's the difference between Anchoring and Collisions when building objects? Anchoring prevents objects from moving, while Collisions determine whether objects can pass through each other. You can disable collisions for decorative elements but keep them enabled for barriers or interactive parts.

OceanofPDF.com

Section 4: Visual and Audio Enhancements

How can I add and manipulate lighting in my game to create different atmospheres? In the Explorer panel, select the Lighting object. Adjust properties like and Ambient to create various effects, such as a night scene or a sunny day.

How can I add sound effects or background music to my game? Insert a Sound object into a part or the workspace, set the SoundId property to the sound file you want, and use a script to trigger the sound when specific events happen in the game.

How do I create particle effects like smoke or fire? Right-click on a part in the Explorer, select Insert Object | and customize the settings (rate, size, and lifetime) to create effects like smoke, fire, or sparks.

OceanofPDF.com

Section 5: Publishing and Sharing Your Game

How do I publish my game to Roblox and share it with friends? Go to File | Publish to then set the game permissions (public or private). After publishing, share the game's link with friends so they can play it.

Can I share my games with others even if they don't have a Roblox account? No, players need a Roblox account to access and play games on the platform.

OceanofPDF.com

Thank you



I would like to thank you for completing this book. I trust that you are now comfortable with Roblox and that you can create interactive 3D game environments. This book is the first in a series of three books on Roblox, so it may be time to move on to the next book for the intermediate level where you will learn more about Roblox and Lua scripting. You can check for all the books in the series on the official page:

Before you do so, please leave an honest review on your favorite e-store, this would mean the world to me and it would also help other people to assess whether this book could help them.

So that the book can be constantly improved, I would really appreciate your feedback and hear what you have to say. So, please leave me a helpful review on your favorite e-store letting me know what you thought of the book and also send me an email with any suggestion you may have. I read and reply to every email.

Thanks so much!!

- Pat

OceanofPDF.com