

CHAPTER 1 The Unity Engine We are in a Golden Age of game development. It's amazing that, just 25 years ago, developing games looked completely different. The '16-bit' era was ending, the 3D era was beginning, and only large Studios had the money and muscle to create games for the masses. During those early days of 3D, Studios were researching & developing gameplay systems as they went. Every new 3D concept came with a new major game release. Third-person camera controls and analog movement were the hallmark of Super Mario 64. It was Halo: Combat Evolved that introduced bump/normal mapping, giving objects unparalleled detail and realism. Fable introduced the Bloom effect, giving lit objects a fantasy-like appearance. And, more recently, Minecraft introduced the world to the concept of Voxels – a way to use 3D cubes to create objects, characters, or even randomly generated worlds. Figure 1.1: Unity has been used to make blockbuster games such as Ori: Will of the Wisps, Subnautica, Cuphead, Hearthstone and Pokemon Go! Video games, and the code they were built upon, were inseparably bonded. The concept of a Game Engine – a set of code and tools to build games upon, available to all – was only a pipe dream. Of course, if you had the money, there were early 3D engines that could be licensed, but the upfront cost created a steep barrier-to-entry that most developers couldn't overcome. If you wanted to make a game, you were coding everything yourself – from scratch. Figure 1.2: Minecraft brought the concept of 3D 'Voxels' to gamers (and game designers). Luckily, that Golden Age is upon us, and developers are swimming in game engine options. Developers that want to make a 2D game can use GameMaker Studio or Pico-8. Unreal, the powerhouse 3D engine that originally cost millions to license, can now be downloaded by anyone for immediate use. There are many options, each with their own strengths and weaknesses. If you're looking for unrivaled flexibility, however, the most versatile option is, by far, the Unity Engine. Supporting 2D games, 3D games, VR and AR projects, PC, Mac, and Linux development, and even web games, Unity has the power to make any game you can imagine. Not only is it flexible – Unity's object-component system makes development and debugging easy and fun. The team at Unity is always working to improve their engine, releasing fresh updates to give you access to the latest technology. And the community of Unity developers continues to grow, eager to help other devs, and continuously creating impressive assets that can be downloaded from the Unity Asset Store. With the Unity Engine, it's never been easier to create your own games. So, let's get started! Structure Unity: The Broad Strokes Using the Unity Hub LTE vs Beta Builds Exploring Extras and Tutorials The Unity Editor Placing Objects in the Scene Moving Objects around The Inspector Window The Hierarchy Window Making Something Move Installing Visual Studio Visual Scripting Objective Chapter 1 is all about the basics of Unity. You'll be downloading the Unity tools and creating your first project. You'll learn about the different parts of the editor, where you'll create your first objects and add some simple Physics. By the end, you'll have written your first Scripts and hooked up some basic Input for changing the rotation of an object. Chapter 1 will give you both a strong understanding of the basics, a firm roadmap for what comes next, and a simple demo to build upon. Unity: The Broad Strokes Unity is one of the leading game engines, used by both independent and AAA Developers throughout the world. It can be used to make games for PC, Mac, mobile devices, consoles, and is the leading choice for VR/AR Developers. Unity allows you to make a game once, then build that project for almost any hardware platform out there. As for the toolset, Unity's Editor is where

you'll be spending most of your time. It allows you to quickly put together scenes, test gameplay in real time, and even pause your game if you need to debug a session in progress. The robustness of the toolset is – in a word – phenomenal. Coding is done using C#, a language that resembles C++, only a bit less complex (doesn't require header files, built in message handling, etc.). C# can also be compiled, helping it to run faster than most other scripting languages, and has its own memory manager so you don't have to worry about allocating and deallocating memory. To write our code, we will be utilizing Visual Studio, which comes built-in when we install Unity. The Unity Hub is another important piece of the unity ecosystem. Here you can manage different versions of Unity, browse and download your projects, and collaborate with others in the community. In fact, the Hub is where our journey begins.

Installing Unity Unity provides a single tool for installing Unity and managing projects called the Unity Hub. Figure 1.3: The Unity Hub – your one stop for managing projects, installing new Unity updates, viewing tutorials, and interacting with the expansive community.

1. Setting up the Hub is as easy as visiting <http://www.unity.com/downloads> and scrolling down until you find a Hub download link. Beta vs Non-Beta: While searching, you may be presented with "Beta" options for the newest version of The Hub. For this book, it's suggested that you download from a non-'Beta' link. It's often tempting to get the latest and greatest version of a Unity tool, but until you're more familiar with Unity as a whole, we'd encourage you to use the existing, well- tested options.
2. Once you download UnityHubSetup.Exe, locate it in your Downloads folder, and run it. This will install the Unity Hub to your computer. It will also automatically launch it.
3. Now that we have the Hub, it's time to install Unity itself. Press the Installs tab on the left side of the hub, then select the Install Editor button in the top left corner. Figure 1.4: Press the Installs tab, then the Install Editor button to bring up the list of versions of Unity you can download.
4. This will bring up a list of the available versions of Unity you can download. We will be downloading the 2021 LTS build. Figure 1.5: You will be presented with several versions of Unity that can be downloaded. Select the latest 2021 LTS version. About LTS and Beta Builds: There are two special types of builds you'll see listed – LTS and Beta. LTS stands for "Long Term Support". Unity provides two years of support for their releases, so 2021 LTS is a stable, feature complete version of the 2021 build. Beta builds, on the other hand, are the newest and coolest builds of Unity, but may contain features still being implemented (and the possibility of bugs) . Since we're looking for stability, we'll download the latest LTS build. Once you're more comfortable with Unity, you may want to jump into the Beta builds and help shape the future of the platform.
5. After selecting the Unity 2021 LTS release, press the Next button in the lower right corner.
6. You'll then get a list of optional Modules that you can install. For now, let's disable everything except for "Documentation" and press Done, which will start the download and Installation process. Figure 1.6: You can install Modules at any time, so don't worry about missing something during your initial setup. With our build of Unity installed, let's take one last look around the Hub before starting our new project. Exploring Extras and Tutorials While the unity Hub is primarily for managing your projects and builds of the unity engine, there are some tabs worth investigating. The Learn tab, for instance, gives you access to a deep well of knowledge directly from the team at Unity. Game-design tutorials, demos, and templates are all Made available with a click of a button. Each of these is highly polished and will always result in new knowledge. Figure 1.7: Unity provides a wealth of

learning content directly in the Hub. Let's say you want to develop your own first person shooter game. There is an FPS Creator kit option, designed with beginner developers in mind, that can be completed in under 2 hours. Or perhaps you're trying to get a younger developer excited about coding. Unity has teamed up with Lego to make a series of microgame tutorials. These allow novice developers to play with block building as they learn how to code games. This tab alone is a deep well of useful knowledge for any aspiring game developer. There is also a tab for the Community, where you can read up on what the Unity team has planned for upcoming features and releases. You can also get help directly from the team, or join the forums, where an incredibly passionate and active community of Unity Developers would be happy to help with any problems you run into. Figure 1.8: The Unity Community is very friendly, so don't be afraid to ask questions and look for feedback! And while these tabs have an exciting amount of content to dive into, we're going to start fresh with our own blank project. Return to the Projects tab and press the New Project button in the upper right corner. Figure 1.9: The 'New Project' Button will get us started. Our First Project 1. Pressing the New button will bring up the "Create a New Project" dialog, giving you several options to choose from, as shown in figure 1.10 Figure 1.10: There are many templates to choose from when starting a new project. 2. We will return to these templates in later chapters, but for now select the "3D" template option. 3. This is also where we will set our project name. Select the field that says New Unity Project and change that to Mastering Unity Demo. Let's also organize all our Unity Projects into a Project directory. Once you have that folder available, set that as our project location. Figure 1.11: Your first project is being created. This may take a few minutes, but it's worth the wait. 4. Press next, and Unity will start generating all the files and folders needed for your new project. This is a lengthy process, and depending on the template you've selected, setup will take between 2 and 4 minutes. Once complete, you'll be presented with Unity's main tool: the Editor. Editor Overview Welcome to the heart of game creation in Unity: The Editor. Figure 1.12: The Unity Editor There is a lot to take in here, so let's take a quick walkthrough of its key sections. Views Window: Directly in the center of the screen is the Scene/Game View window. This is where you can edit your scenes and levels, as well as test them in real time. Hierarchy Window: To the left of the Views panel is the Hierarchy list window. This contains all the objects in the current scene. Inspector Window: On the right is the inspector window, which shows all of the data related to any selected object. Toolbar: The vertical list of buttons in the Scene View is the Toolbar. This gives you the tools needed to manipulate objects. (move, resize, rotate, etc.) Project Window: All the assets related to this project. 3D models, 2D textures, sound and music files, code, and anything else required for your game to look, sound, and play as you want it to. Console Tab: Pressing the console tab brings up a debugging window called the Console. This window shows output and errors that may occur in our code. Play, Pause, and Step Buttons: Use these to test and debug your game project. Now that you know the basics of the editor, let's dive into the centerpiece: the Scene View window. The Scene and Game View Window Right in the middle of everything is the Viewport window, which consists of two parts: a Scene View and a Game View. Each of these modes have a respective tab. Figure 1.13: The Views window in "Scene" Mode. Pressing the Scene tab when you want to edit your game. This mode lets you see all the objects, select them using your mouse, and edit them in real time. You can drag objects into your scene, move them

around with the gizmos, and view their details in the inspector panel to the right. Pressing the game tab will bring up the game as it'll be seen by the player. This is the tab you press when you want to critique visual effects, user interface look and feel, and playtest your project. Let's keep the viewport in Scene mode until we have something worth playtesting. The Scene

Since we're in Scene mode, it would be a good time to talk about the scene itself. All objects in a Unity project will be placed into a 3D Scene. Think of it as a sandbox where all your enemies, power-ups, and level design objects will live. Scenes can be filled using the unity editor (by dragging objects into it) or at runtime (using code). Scenes are created and managed in the Project window. A project can have multiple scenes, with many game developers using a new scene for every level in their game. For now, we will keep things simple and only make use of our initial scene.

The Hierarchy Window On the left-hand side of the editor, you will find the Hierarchy Window. This is a list of everything in the scene, giving you a single spot to search for and select specific objects: **Figure 1.14: The Hierarchy Window** – the master list of the objects in the scene. At the top of this window is a Searchbar, where you can narrow down the listed object. For instance, if you click where the search field says All, and enter Camera, you'll see that the Directional Light object disappears and only the Camera remains. To the left of the search field there is a + button. This is a quick way to add more objects to the scene. To the right of the search field, there is a button that opens an advanced search window, where you can search for a given phrase across your entire project. Following the search bar is the name of the current scene, which is currently SampleScene. Under the Scene Name is a list of all the objects in the scene. Right now, only two objects are shown: the Main Camera and a Directional Light. If you look over at the Scene View, you can also find the listed objects shown there, represented by 2 small icons known as Gizmos. Select these objects in the hierarchy and note that they also get selected in the Scene view: **Figure 1.15: The default Camera and Directional Light in the Scene**. Note that selecting them in the Hierarchy will also select them in the Scene View Window. The Camera gizmo represents where the player will be viewing the scene from. The Sun gizmo represents the default Directional Light for this scene, storing the direction, intensity, color, and shadow settings for this scene. As your project grows, expect this list to get very long and very complex. Soon we'll learn about Parent/Child hierarchy, which will allow us to organize multiple objects in a single parent object. For now, however, let's work on editing one of these objects.

The Inspector Window

1. Select the Directional Light object by either clicking on it in the hierarchy, or by selecting the 'sun' gizmo in the Scene View window.
2. Once selected, the settings for the directional light will fill the window on the right-hand side. This is called the inspector Window, where you will be editing objects in your game: **Figure 1.16: Selecting the Directional Light gives you access to all its settings in the Inspector**.
3. Because we've selected a Directional Light object, many of the options listed will be directly related to lighting. As shown in **Figure 1.16**, you'll have access to the Type of light, Color, Intensity, Shadow options, and more.
4. Now select the camera object, either in the hierarchy panel or by clicking its Gizmo in the Scene View. The inspector panel will update accordingly, listing all the available data related to the camera: position, rotation, field of view, and so on: **Figure 1.17: The Camera Inspector**

For now, we will keep all these settings as they are. Instead, let's add some new GameObjects into your scene.

All About GameObjects The GameObject is the basic building block of all Unity games.

Whether it's a character, a weapon, a building, an Interface Element, or even the camera or directional light – each of these is built from the basic `GameObject`. While most objects in your game will be imported from external asset creation tools (Maya, 3dsMax, Photoshop, and so on), Unity comes with a handful of basic objects to play with. Cubes, spheres, planes, and cylinders will give us all the basic objects we need to get our demo game started.

Placing Objects in the Scene

1. The most straightforward way to place an object into your scene is to use the right click menu in the Hierarchy panel. Bring up the menu, navigate down to 3D Object, and select Sphere:
2. Selecting Sphere from this menu will place a 3D sphere object into the scene directly in the center of the viewport. You can also place a sphere from the top Menu Bar by selecting `GameObject>3D Object>Sphere`. Either way you do it, your scene will now have a Sphere object in the center of the Scene View:
3. The 3D Objects that come packaged in Unity also have several pre-assigned components. Select our new Sphere object to see all its information listed in the inspector panel:
4. At the top you will see the Object Name. This is for internal use and will never be displayed to the player themselves. Feel free to come up with a naming system to give your objects instantly identifiable names. For example, let's name this one `PlayerObj_Sphere`. Click on the existing name and enter `PlayerObj_Sphere` as our new one.
5. To the left of the name is a small check box to enable or disable this object: **Figure 1.21: Object Name Panel**
6. Click on it once to see the Sphere disappear in the Scene View. You'll also see the object name turn gray in the Hierarchy panel. Note that, when you toggle it again, the sphere reappears, and its name is no longer greyed out.
7. This top section also includes options for Tag, Layer, and Static. These are advanced options that we will dive into later, so keep them at their defaults for now.
8. Following this top section is the Transform panel. This is an important section that shows where the object is in 3D space, as well as its rotation and scale information: **Figure 1.22: Transform panel Local vs World Space**: The location listed here is in "Local Space". **This means that these positions are relative to the parent object.** If an object says it's X position is 5, but it's parent Object is at -5, The "World Space" position would be 0. We'll dive deeper into Parent/Child objects in a later chapter. While the transform data is laid out in text fields, you can also click on one of the 3 axes (X, Y, Z), hold the left Mouse button down, and drag your cursor to change this value. This is a quick way to make broad changes to these values without needing the keyboard.
9. Below the Transform component, we have several areas with Mesh and Rendering information: **Figure 1.23: Mesh Rendering Components**

Rendering Meshes

A Mesh is a collection of triangles that represents a 3D object. Rendering is the process of taking those triangles and drawing them to the screen. The Mesh Filter is simply a `GameObject` component that specifies the 3D mesh to display. The Mesh Renderer component has all the material, lighting, visual settings to be used when rendering the specified mesh. Again, for now, we will keep these default settings. Below the rendering data, you will find a Sphere Collider panel. Colliders are components that tell the physics system what the shape of a `GameObject` is. In this case, our sphere mesh makes use of a Sphere Collider: **Figure 1.24: The Sphere Collider Panel**

The last panel in the inspector panel shows us the material being used for this game object. Included `GameObjects` – such as the Sphere – will use the default material, which is a white plastic looking material. We will learn more about materials, textures and meshes in our chapter on Graphics, but for now our demo game will use default visuals.

Moving Objects Around

Now that we have

our sphere placed in the 3D World, The Next Step is learning how to move it around in the Scene. The

Editor gives us two ways to do this: The transform panel and the movement Gizmo. First, select the object you want to move around the scene. In this case, you can left click on the Sphere we just added. If you know precisely where the object should be in 3D space, you'll want to use the transform panel we mentioned earlier. This is the area in the inspector panel right under the name of the object. Here you can type in the specific X, Y, and Z location to place the object at. You can do the same thing with scale and rotation. There's also the transform Gizmo, which are the arrows that appear when you select the object. Each of these arrows corresponds with one of the axes that you can move the object to: X axis is red, Y axis is green, Z axis is blue. Use this opportunity to drag the movement Gizmo around and get an idea for how it feels. If you click directly on an arrow, you will move only in that axis. You can also select the object and drag it around on all axes, but it's an imprecise option, so use it sparingly: Figure 1.25: The transform gizmo, in 'Movement' Mode. In this case, we actually know the precise location we want our object placed. With the sphere selected, look at the right side of the screen to find the Transform Panel. Select on the entry field to the right of X and enter '0'. Set Y to '3' and Z to '-2'. When done, the position values should look like this: Figure 1.26: Our Sphere's position, with the updated values. Now, to ensure we entered the information correctly, let's check the Sphere in the Game View. The Game View is the scene as viewed through the Main Camera, with all lighting effects the player will see in the game. 1. Select the Game tab in the Views Window. If we did everything right, the sphere should be just above the halfway point in the Game View window. Figure 1.27: If the Sphere looks like this, then you've entered the data correctly. With the Sphere object in place, we're ready to put our little demo to the test. The PLAY Button At the top of the screen, almost directly in the middle, there are three very important buttons: Play, Pause, and Skip. Play will start (and stop) running your project. Pause will pause and unpause a running project. The last button is Skip, which will pause a game and advance to the next frame – useful when debugging. Press the Play button now. Figure 1.28: The Play, Pause and Skip buttons. What you will see is the sphere...doing nothing. How terribly unexciting. I apologize for the buildup. This is because the Sphere needs a Rigidbody component to officially hook into the physics system. Let's first stop the simulation by pressing the Play button, which will turn from Blue to Grey. Once the game is stopped, select the Sphere. At the bottom of the Inspector panel, you'll find an Add Component button. Press it to bring up the Components panel. Figure 1.29: Adding a Rigidbody to our Sphere. Select the Physics option, then Rigidbody. This will place a Rigidbody component on your Sphere object, officially making it a 'Physics' object. While we'll be diving into Physics in a later chapter, you can take a moment to look over the component we just added. Figure 1.30: Rigidbody Inspector Panel There is a lot of options here. Some are pretty obvious, like "Use Gravity". Others less obvious, like "Is Kinematic". Like with most components, let's keep everything at their default value until we understand them better. Now let's press Play again. Just like magic, we see the Rigidbody component in action. The Sphere, now being affected by the physics system, is pulled downwards by gravity. One second later it's gone, and we're stuck staring at a blank screen. Now what do we do? First we press the play button again to stop the game. This returns all the objects in the scene to the way they were positioned before the game started. Second, we need some ground. So let's place our second object.

The Importance of Stopping your Game: It may seem like a silly thing to be warned about but remembering to press that play button to stop a running game is VERY important. Because everything reverts to it's starting state, it's easy to accidentally make – and lose – changes to your project by implementing those changes during run-time. Your best practice is to ALWAYS stop your game, and verify it's no longer running, when you need to make changes to your project. This is an important concept that, if ignored, will lead to lost work and frustration.

Your Second Object To keep our player object from plummeting into the void, let's add some ground for them to interact with. 1. Like with the sphere, right click in the hierarchy window and add a Cube to the scene. Figure 1.31: Our new Cube, in its 6-sided glory! 2. Once placed, left click the cube to select it. We can now edit its position, scale, and rotation – directly in the Inspector window – to flatten the new object more ground-like. Table 1.1: The Transform of the new Cube Object. 3. With the position values, we've moved the Cube to be located below the Sphere. The Rotation has a very slight tilt to it, as set by the 1 in the Z Rotation field. And the Scale has the cube stretched in the X and Z axes but flattened in the Y axis. Figure 1.32: Your ground object Cube should now be flattened to look like this. 4. While we're here, let's give this object a more informative name. Click on its current name "Cube" and rename it to LevelObj_Ground. 5. Now that we have a ground object, as well as our player sphere, let's see what happens when we press the play button. 6. With a ground object beneath the falling sphere, the downward motion of our ball is almost instantly stopped by the ground. Then, because the ground object has a slight incline to it (thanks to Z = 1 in the Rotation panel) the sphere will slowly roll to the left, as it continues to be affected by gravity. How exciting!

Using Visual Studio At this point our game consists of a ball and a flattened cube. It's actually not that exciting. Let's add some interaction to our objects. This is a good time to introduce the second most important tool in your game making Arsenal: Visual Studio. Automatically set up when you first installed Unity, Visual Studio is the preferred code editing tool of the Unity Engine. In it, you can create classes that expose important data back into the unity editor, allowing you to actually extend the functionality of the editor by simply adding the right tags. Visual Studio also has a system called Intellisense, that will expose functionality as you write it. Well experienced Developers will have used intellisense before, beginner coders will be happy to know that all the functionality you'll need the right will be shown to you through intellisense. It's pretty awesome. To get started, we need to create our first script. Scripts are the text files that contain code for our Unity games. specifically, scripts are written in the C# language (pronounced "C Sharp"). We're going to write a script that allows us to tilt our Cube with the arrow keys, giving our sphere something to roll around on. Since we'll be writing code that affects the Ground cube, we'll want to add this new script directly to our new "LevelObj_Ground" object. Select the cube to bring up the inspector panel. scroll down to the bottom of the inspector panel and press the Add Component button. Figure 1.33: Add Components to your objects with this aptly named button. 1. A panel with a list of options will pop up. Scroll all the way down in this list and select New Script. When it asks for the name of this new script, enter "GroundObjectController", then press the Create and Add button. 2. Pressing the enter key finalizes the name of your script, creates the file and class, and automatically adds it to the selected object. Figure 1.34: This is the new script, as added to the Ground object. 3. To start editing this code, click on the grey box in the lower right corner that says

“GroundObjectController”. This will open that script in Visual Studio. 4. If you’re new to Unity development, much of what you see in the file is going to be unfamiliar. That’s expected, which is why we will dive deeper into Components in a later chapter. 5. For now, simply find the line `void Update()`. This is the update function of any `GameObject` script. As you may have guessed any code present in `Update()` will be called every frame, giving you the perfect spot to tell this object how to move and react.

6. In this case, we simply want to let the user rotate the ‘ground’ object in two directions. 7. Add the following code into the `Update()` function... `// the speed that this object rotates float speed = 15f; // get the current rotation of the ground object // based on the keys pressed // we will be altering this value and re-applying it Vector3 newRotation = transform.localEulerAngles; // go through each of the direction keys: up, down, left, right // if any of them are pressed, alter the rotation of the ground plane if (Input.GetKey(KeyCode.RightArrow)) newRotation.z -= Time.deltaTime * speed; if (Input.GetKey(KeyCode.LeftArrow)) newRotation.z += Time.deltaTime * speed; if (Input.GetKey(KeyCode.UpArrow)) newRotation.x += Time.deltaTime * speed; if (Input.GetKey(KeyCode.DownArrow)) newRotation.x -= Time.deltaTime * speed; // apply the updated rotation to the ground object transform.localEulerAngles = newRotation;` 8. This block of code will check if any arrow keys have been pressed by the player. If so, we apply a simple rotation to that object: in this case, our ground cube. Commenting your Code: If you’re new to game development, you may wonder what those `//` lines are for. Those are called COMMENTS, and they allow the developer to explain code in a way that’s easy to read. Comments are ignored when the code is run, and since code can be hard to read, it’s suggested that you comment your code whenever possible. An impressive feature of Visual Studio is that it’s always checking to see if the code has any errors (if it has errors, the code will not run).

If we wrote these lines correctly, we should be able to click on the output window in Visual Studio and see that no errors have been found. If errors have been found, Visual Studio will direct you to the problematic code. Simply go to any line with errors and double-check that it matches the sample code above. Putting it All to the test Our demo now has objects to play with and scripts that manage player interaction. Now it’s time to put your hard work to the test. Select the Game tab in the viewport panel and click the option “Play Maximized”. Now, when we press the play button, the Game View will fill the screen, bringing us closer to how the game will feel when exported to the final build. Pressing the arrow keys should rotate our ‘flat cube’ as expected. The ball will roll based on how you’re tilting the ground object. And even without weapons, enemies, music, or fancy graphics, trying to keep that ball from falling into the abyss shows the distant glimmer of a fun game. Optional Tool: Visual Scripting In this chapter, you were introduced to Visual Studio and C#. By writing a short script, you were able to take input from the keyboard and use it to rotate an object. Using Visual Studio for writing and debugging your game code is the preferable method of development. Once you’re familiar with writing code, it’s an incredibly useful skill that can lead to a professional career in software development across hundreds of industries. It’s also a skill that – at first – can feel daunting. To help ease incoming developers into the world of coding games, Unity now comes pre-packaged with a Visual Scripting system. Figure 1.35: A visual way to code, for those that prefer it. For visual learners that want to see their logic graphed out, the Visual Scripting tool is a node-based solution for writing code. By connecting these nodes, you get

almost all of the flexibility of writing code, without the headaches of reading and parsing it. Be warned, however, that while this is a great tool for learning and writing smaller systems, the more complexity you add, the more difficult it becomes to manage this style of development. Conclusion Congratulations! You now have a project with two objects: a sphere that rolls around, and a ground plane that can be tilted as arrow keys are pressed. It's a humble beginning, but all major releases start with a basic foundation.

Every game that has captivated players began simply – with a just few objects and a sprinkling of interaction. So...it may be simple, but we're in good company. And while we're appreciating this milestone, let's list the game development systems you're making use of: Object Rendering Physics Simulation Input Handling Real-time Scene Editor The demo you've made, along with the subsystems you've tapped into, are built upon years of work done by teams of professional, passionate developers. Work that would have taken a solo developer months – if not years – to perfect on their own. Everything they've made – every system, mechanic, and tool they've built – are now yours to play with and create with. The Golden Age of game development is here. And we're just getting started. Questions 1. What are some of the benefits to using Unity over building your own Game Engine? 2. True or False: If you want the most stable build of Unity, you should use a Beta or Alpha release? 3. What are the differences between LTS and Beta builds of Unity? Why will we be using a LTS build while learning the tools? 4. What gets listed in the Hierarchy Window? What about the Inspector Window? 5. What happens when you press the Play button when the game is running? What about the Skip button? 6. The Transform of a game object contains three things: the Position of the object, the Rotation of the object, and what else? 7. Colliders are important components that define what aspect of an object? 8. Unity comes with several 3D Objects that you can place in your Scene. And while each comes with a Collider component, an additional component needs to be added for the object to respond to the Physics system. What is the name of this Physics component? In the Unity Editor, you can press Play to start running the game. If you make changes to objects while the game is running, do those changes get permanently applied? Or do they get reset when the game is stopped? Key Terms Game Engine: A set of code, systems, and tools that have been developed with the single purpose of creating Video Games. Some engines are tied to a specific series, while other are game agnostic, providing a great foundation for building any new project. Unity Engine: The Unity Engine is a modern Video Game engine that was designed to be used by anyone. It has the tools, systems, and raw power to create amazing projects without needing a large team. Unity Hub: A app that organizes both your projects and Unity builds. The Hub is also where you'll start a new project and connect with the Unity Community of developers. Unity Asset Store: A website where you can find sample projects, models, textures, gameplay systems, and anything else you can imagine needing for a game. As you learn more about Unity and build a skillset, you can even sell your own creations on this storefront. The Editor: The Editor is the main tool for creating games in Unity. Here, you will see a 3D scene where you can place objects, adjust lights and cameras, and build objects, levels, and even entire worlds. Game Object: Any object placed in a Unity is based on a core object type called a GameObject. GameObjects can have Components added to them and have a position in 3D space, known as a Transform. Transform: The position, rotation, and scale of a game object in 3D space. An object's transform can be changed in the editor or by adjusting it in code.

Hierarchy: The list on the left side of the editor. It shows all the individual objects that make up the current Scene. At the top of the Hierarchy, there's a useful search bar that can narrow down the visible objects in this list. Inspector: All the components that make up the selected GameObject are listed on the right side of the editor, in a window called the Inspector. After clicking on an object, you can edit its parameters in the Inspector Window. Project Window: This window shows the folders and assets that make up this project. Console: A window that shows errors and debug output from the game code. Visual Studio: The program used to write your game code in. Gets installed automatically when you .install Unity