Introduction to Game Developement

Om Prabhu
19D170018
Undergraduate, Department of Energy Science and Engineering
Indian Institute of Technology Bombay

Last updated July 12, 2020

NOTE: This document is a brief compilation of my notes taken during a course in game design and development. You are free to use it and my project files for your own personal use & modification. You may check out the course here: `https://www.coursera.org/learn/game-development?specialization=game-development`.

Contents

1 Introduction

1.1 About myself

Hello. I am Om Prabhu, currently an undergrad at the Department of Energy Science and Engineering, IIT Bombay. If you have gone through my website (`https://omprabhu31.github.io`) earlier, which is probably where you found this document too, you will know that I love playing video games, story-rich titles in particular. I also listen to a lot of music and engage in a little bit of creative writing as and when I get time. With this brief self-introduction, let's get into what actually motivated me to pursue game development.

1.2 Motivation

Most of my motivation for pursuing game development came from playing games itself. I am talking less of titles like *Grand Theft Auto*, generic FPS/RPGs, etc meant purely for self-entertainment and more about games like *Life is Strange*, *When the Darkness Comes*, etc that actually give you some amazing stories and/or simple, powerful messages to be remembered for life. When one has experiences like this, the question naturally hangs at the back of their minds - why not create enriching experiences like this?

Now while playing games (of all genres) is vital to understanding what essentially makes a good game, they are two very different things - it's like comparing movie binging to actually making movies. Making games involves a lot of hardwork at different stages of the development process and there is a reason why good game developers take their time (often more than 10 years) before putting out a game on the market.

Nevertheless, I decided to give it a shot - the worst that could happen is I could end up hating it, but I hate it during quarantine anyway.

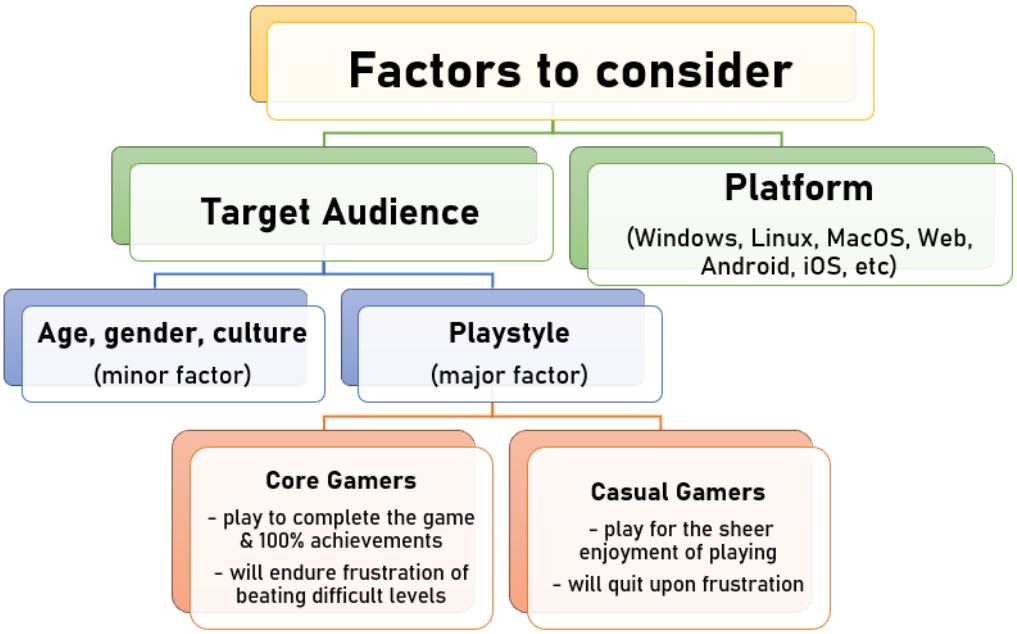
NOTE:

1. This entire course works with Unity3D as the game engine, however the exact same concepts apply to other engines like Unreal, Godot, etc as well.
2. The source files for course projects are available on my website for free use and modification. I will mainly be working with the 2019.4.2f1 and 2017.4.40f1 LTS releases of Unity, so you might need to install these versions of the engine before you try them out.
3. Most of the project builds will be for the WebGL platform. Many browsers do not support running local WebGL content out of the box. You might find this guide handy: `techwiser.com/enable-webgl`

---

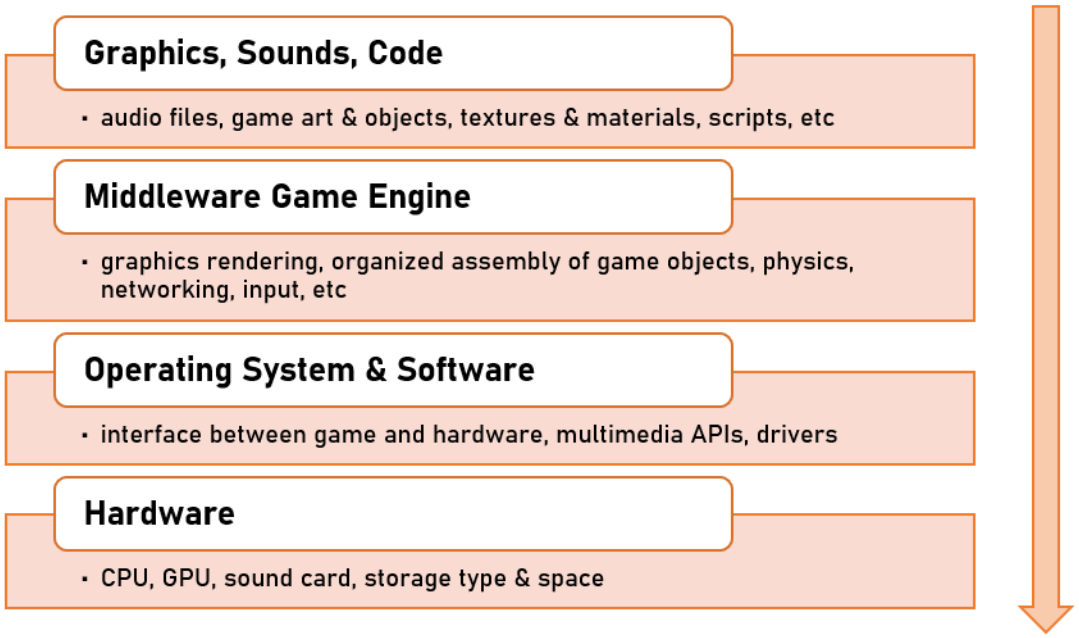
2 Overview of Game Development

2.1 Factors to consider before starting out



2.2 Hierarchy of game structure

Keeping in mind the above factors, we proceed to conceptualize the game structure.



While normally we would construct anything from the ground up, from a game design perspective things work better when conceptualized from the top down.

- We first create everything essential, i.e. 'game assets' - that would mean any art, textures & materials, game object models, audio/music and scripts (basically code that defines interaction between objects)
- Then, we actually construct the levels in the game using the above created assets in an engine like Unity, Unreal, Godot, etc
- We then build the game for the intended platforms
- Finally the game is tested for bugs and to determine the requirements for a system that could hope to run our game

2.3 Team structure and roles

Considering all the stuff from the previous subsection, it would be near impossible for a single person to carry out all the work required and make an end product that ticks all the boxes (individual developers do exist still). This is why there are often teams, ranging in size from as small as 2-3 people to well over 1000 employees, with each member having one or more roles to perform.

- Game Designer: execution of the idea behind the story, designing levels, gameplay mechanics, organizing assets and documents
- Story Designer: character design, story arcs, writing dialogues, etc
- Game Producer: monitoring project budgets, keeping track of deadlines, keeping the team together
- Programmer: writing actual code to determine how game objects interact
- Game Artist: create concept art, textures for game objects to give them unique looks
- Sound Designer: create and edit music to match the mood of various in-game scenes
- Game Tester: not the same as a player; requires sitting on a single game level for hours to identify bugs/inconsistencies

Depending on the size of the team, each person may assume one or more of the above roles based on what they can best do:

- a small team (1-5 people) may be able to make do with just general purpose designers, artists and programmers, and each person will be more or less actively involved in production work
- a medium sized team (6-25 people) may additionally require a manager/producer and specialized designers & artists
- a large team (25+ people) will have supplementary roles like scripters (people who have knowledge of programming as well as design), technical artists (programmers who can also work as artists) and level designers (who make the artwork and game design for separate levels)

---

3 The Unity3D Engine

All games need a few basic features - loading and displaying game assets, playing sound FX. receive player input and react to it, some code to define game rules and mechanics. One way to incorporate all of this is to build everything from scratch i.e. the core game software, renderers, etc. While this can certainly be done, it is much easier to use a game engine to do most of the work for us.

A game engine is a platform that provides basic game functionality, support for integration of game objects of multiple different formats. This allows us to focus on the actual unique gameplay elements.

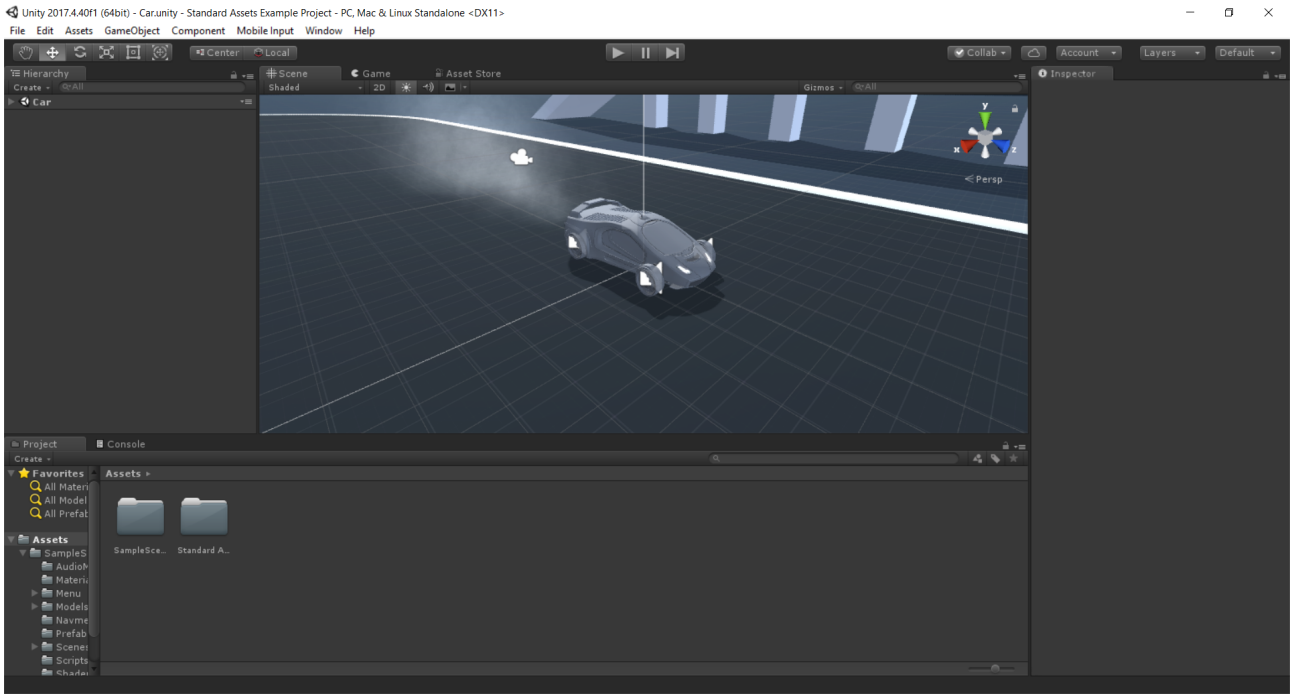
Some advantages of using the Unity3D game engine are:

- great documentation (by which I mean insanely great) and a very large dev community
- very fast build times and support for many platforms
- great asset pipeline (mechanism to import and use game assets)
- not as processor intensive as other engines like Unreal
- can actively switch between different editor versions using Unity Hub without needing to rebuild the entire project

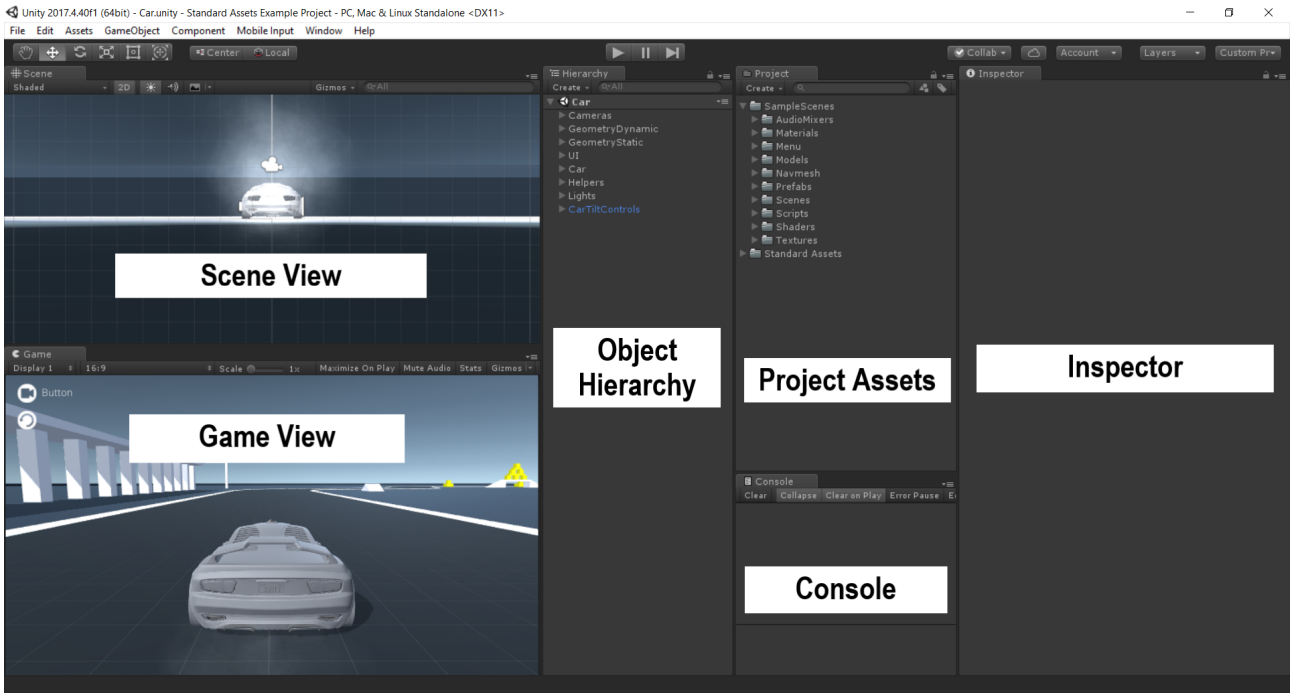
NOTE: If you take up this course, the instructor will mention that Unity3D offers flexibility in programming languages between C# and JavaScript. This is no longer valid after the 2019 LTS releases. However, many .NET compatible languages like C++ can be used if they are first compiled into a DLL (for which you can refer this - `https://docs.unity3d.com/Manual/UsingDLL.html`).

3.1 Unity3D interface and configuration

The process of downloading and installing Unity is very intuitive and I won't go over it. However, note that Unity has stopped distributing the Standard Assets Example Project from 2018 onwards, so you might need to use the 2017 releases if you want to try stuff out with the example project. On launching Unity, the editor window might look something similar to this:



I personally hate this layout (for the main reason that you can only see 4 panels at any given time which means you have to keep on switching between tabs). You can play around with this layout by simply dragging various tabs around the window to get the window configuration you like. I personally prefer this editor configuration, since now you can see all tabs within a single window:



Let us now take a brief look at the various elements in the Unity3D interface:

- Scene View: visual level editor for the currently open scene (edit mode)
- Game View: shows the view through the active camera object (the play button can be used to switch to play mode for playing and testing the game within the Unity editor)
- Object Hierarchy: hierarchical list of all game objects being used in the current scene along with parent-child relationships (is linked directly to the scene view)
- Project Assets: all the files and folders that make up the project (including the ones which are not being used in the current scene, but can be used later)
- Console: displays errors while building the scene (eg: lighting needs to be rebaked)
- Inspector: shows details of all game objects and components (like position, scale, scripts, materials, audio, etc) attached to them

You might now want to explore the Unity editor to look at various features. There are also many quick shortcuts in Unity that you can refer to here: `https://docs.unity3d.com/Manual/UnityHotkeys.html`.

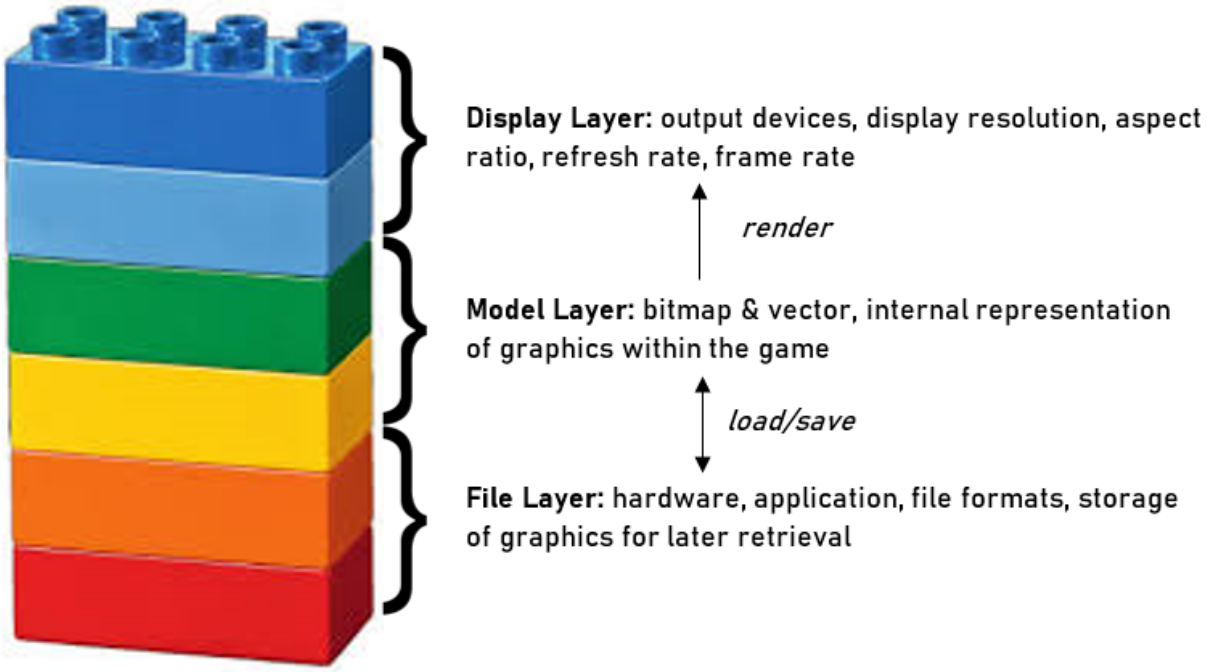
NOTE: If you ever need to move/rename/delete asset files, do it in the Unity editor itself (in the Project Assets panel) and *NOT* in the system file explorer - doing this may affect some connections between game objects and break your project.

---

4 Basic Concepts

4.1 Game graphics concepts

Before we start using Unity, let us briefly discuss the basics of computer graphics using the graphics display model:



1. Display layer:
    - display devices: laptop, smartphone, tablet, VR glasses
    - display resolution: measure of how accurately a display approximates continuous images using finite pixels (pixel dimensions like 1920×1080, dots/pixels per inch)
    - aspect ratio: proportion of pixel map width vs height (eg: 1920×1080 and 1280×720 both have aspect ratio of 16:9)
    - refresh rate: maximum rate at which the hardware can refresh the display image (expressed in hertz)
    - frame rate: actual rate at which the hardware updates the display images (eg: a complex game might run at 60 FPS even though the display has a refresh rate of 240 Hz)
2. Model layer:
    - bitmap image: stores color information about each individual pixel (zooming degrades image quality since we are not creating new pixels, but rather zooming in on the available pixels)
    - vector image: stored internally as mathematical equations representing certain geometric aspects of the image (can be scaled up without loss of quality)
    - vector images take longer to load (additional step of converting vector information to the corresponding pixel map before rendering it to the display), hence usually give lower frame rates
    - 3D graphics: essentially vector graphics in 3 dimensions (i.e. 3D model defined by geometric polygons, usually triangles, to create a 'mesh' of the model) - for realtime rendering at high FPS, we use low polygon modeling (i.e. limit number of polygons)
3. File layer:
    - Unity supports nearly all file formats (support for 2D vector graphics also added recently)
    - for 3D formats (eg: native Maya, 3ds Max, Blender files), Unity converts to FBX files while importing

NOTE: As a general rule of thumb, try to maintain highest achievable image quality for as long as possible - reduce it only if it is adversely affecting the gameplay.

4.2 Game audio concepts

We can apply the analogy of the graphics display model here as well:

- instead of the display we have a device which converts digital data signals back into analogue waves (7.1 surround system, headphones, etc)
- instead of image storage algorithms we have a process called sampling (approximation of continuous analogue input to discrete digital pulses)
- sample rate: number of times per second the sound wave is 'sampled' (typically 44.1 kHz)
- sample size: amount of data stored per sample (typically 8, 16 or 32 bits)
- typical file formats include WAV, AIFF, OGG, MP3, etc (software like Adobe Audition or Audacity have their native file formats)

Audio is very important to games, and also one of the tougher things to actually make or synthesize digitally. There are 3 main types of game audio:

- voice: easiest to come by - voice actors need not be professionals, they can be friends, family, etc (pro tip: keep background noise as low as possible, otherwise noise removal is a very difficult task in production)
- sound FX:
  - reactive FX: effects that occur as things happen in the game (royalty free, creative commons licensed effects can be easily found in places like the Unity Asset Store)
  - ambient FX: ambient noises that add a feeling of immersion (natural sounds often work better than digitized ambient effects)
- music: adds emotional impact and complements the atmosphere (again free resources exist; commercially produced music will need you to buy a license in order that you do not violate copyright law)

END OF WEEK 1

This is the end of the documentation from Week 1 of the course. Continue reading forward for further weeks, or head over into Unity and try out some stuff yourself.

---

5 Solar System Simulation

(*This section is not complete yet, but it should be up within a week's time at max.*)
