

# Sparky Game Engine - Notes

---

Date: 3/8/2015

## Sparky Engine

These notes capture the development of the Sparky Game Engine. The game engine will be developed as part of a new video series hosted at the following urls:

- <https://www.youtube.com/channel/UCQ-W1KE9EYfdxhL6S4twUNw> - you tube videos



Figure 1 - TheCherno youtube home page

- <http://www.twitch.tv/thecherno> - live videos of the game being developed
- <https://github.com/TheCherno> - GitHub location of source code for this project

The game engine will be built in 45-days in preparation for the upcoming Ludem Dare competition. This will allow game developers to concentrate on the game mechanics and not the common details that the typical game engine handles.

### *What is Ludem Dare?*

## Ludum Dare

Developers from around the world get together several times a year and spend a weekend building a game from scratch using the game theme presented to all the developers. The game theme is proposed and selected from the game development community. The goal is for the individual or a team to create a game in 72 hours. You can use any tools or libraries or start with any base-code you have. You can even use art work and music from 3<sup>rd</sup> party sources. You are encouraged to create games that can be played right in the browser. The source code created is made freely available to community. This practice encourages sharing and growth as others can learn new techniques and programming styles by reviewing code (of the winners and losers).

I will post the latest notes on github at <https://github.com/nyguerrillagirl/Sparky-Engine>. The repository will contain:

- The latest copy of these notes

# Sparky Game Engine - Notes

---

- A file named codeCommitSessions listing the commit codes corresponding to the code developed for the end of a video coding session. For example if you want to obtain the code as it was developed up to video session #5 then the commit number (e.g. d42371e6a7) will represent the code completed up to that session. So checking out that commit<sup>1</sup> will get you to that location in these notes and on the video.
- The code related to this project



I highly recommend that you follow TheCherno on the various locations listed above for more information and to view the videos relates to these notes and to donate to TheCherno.

You can follow me at @nyguerrillagirl on twitter or visit my web site [www.brainycode.com](http://www.brainycode.com) . Do not hesitate to provide corrections and suggestions to my e-mail address at nyguerrillagirl at brainycode dot

com.

## What is a Game Engine?

A game engine is a code package, framework or application tool set that implement the common tasks that all games perform – rendering the screen, physics, input, and collision detection, etc. so that the game developer can concentrate on game specific details like art work, and specific game logic.

A typical game engine provides an API or SDK (collection of libraries)<sup>2</sup> that the game developer will use as a starting point to build their own game.

You are probably familiar with popular and well-known game engines such as Unity (<http://unity3d.com/>) or the Unreal Engine (<https://www.unrealengine.com/>). These engines are rather complex and have a high learning curve. There are simpler game engines such as GameMaker:Studio (<http://www.yoyogames.com/studio>) and Construt 2 (<https://www.scirra.com/construct2>) that make quite easy to build 2D games.

Using a game engine (most are free) will provide an insight into what you want your own game engine to do, so I advise that you check out one or more game engines in order to get an idea of the typical functions that are abstracted away into APIs and how a game developer would interface with a game engine in order to build their actual game.

## Why build your own game engine?

The benefits of developing your own game engine is the appreciation you gain on what a decent game engine is doing and the technical complexity in implementing one. In addition, it will make is easier to learn and work with more complex game engines.

---

<sup>1</sup> See Appendix A for a quick tutorial on using GitHub.

<sup>2</sup> [http://www.gamecareerguide.com/features/529/what\\_is\\_a\\_game\\_.php](http://www.gamecareerguide.com/features/529/what_is_a_game_.php)

# Sparky Game Engine - Notes

---

## What Programming Language to use?

The first question being debated is what programming language to use. The two top choices in order to achieve maximum portability is Java or C++.

C or C++ has been the workhorse or primary language of choice for most in-house game engines and/or games. The advantages of using the programming language C++ are:

- Compiles into native machine code so the game engine will be fast
- Supported on all platforms
- Free IDEs and compilers are available (that run on all platforms)
- C++ is well-known by most developers and students
- C++ has many supporting libraries and frameworks

The downside of using C++ is that we probably will not get as many features as we would like completed for our game engine. The reason most game engine developers select C++ is because C++ has been the dominant language choice for many years (yes - use it because everyone else has!). The leading game consoles development tools use C++, many middleware packages are written in C++ and when you think about it if you want support, assistance or want to expand your work of art it will be easier to find C++ experts.

The major downside of using C++ is the plain ugliness<sup>3</sup> of the code since it has object-oriented concepts layered on top of C so the re-use of symbols makes it rather difficult to read.

Java is up for consideration because

- it is easier to build a cross-platform game engine with it
- software development and debugging is faster
- has freely available tools and frameworks

Java comes with a host of libraries and tools (all free!) that make it easier to build and test programs. It will be cross-platform with no extra work or flags required in our code. The downside is the fact that if you are trying to prepare to get a job with a company that develops professional console games chances are that they DO NOT USE Java<sup>4</sup>.

## What platforms should it work on?

The main platform that the game engine will be designed for is Windows. A secondary goal will be to have it run on Mac and Linux environments.

---

<sup>3</sup> This is rather subjective since beauty is the eye of the beholder (and therefore so is ugliness). I think it is fair to state that the more experience one has with C++ the less ugly it seems.

<sup>4</sup> This may not be true if they are building games for Android devices

# Sparky Game Engine - Notes

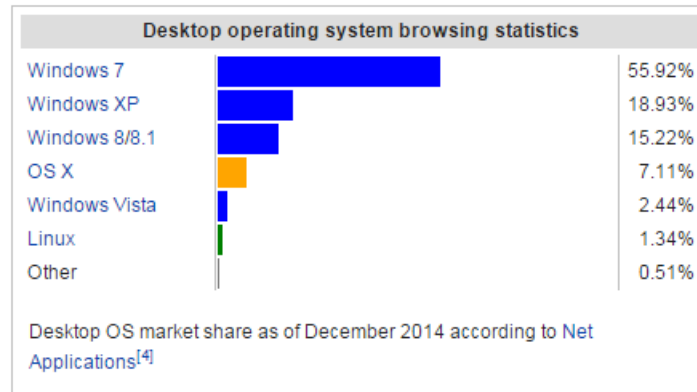


Figure 2 - Operating systems on desktops

As you can see Windows 7 has the largest share of users on desktop environments. The game engine will be built on a Windows OS. I myself will be using Windows 7.

As the project progresses I plan on using the same compiler and IDE that is available on Mac and Linux platforms in order to ensure everything works on as many platforms as possible. I plan on setting up my Mac and Linux machines with the same version of all the tools and provide any notes on differences in one of the appendices.

## What are the goals?

The goal is to build a 3D game engine. We will get to our goal by first focusing on building a 2D game engine and expanding it to support 3D.



Figure 3 - 2D game look-and-feel

The classic and most popular type of 2D game is the 2D platform scroller (as shown above). The game engine will support the basic elements of building a 2D game:

- Creating and laying out game objects (players and enemies)
- Collision detection

# Sparky Game Engine - Notes

---

- Input handing
- Scrolling
- Level transition
- Sound
- Heads up display

Many game engines (e.g. Unity) allow you to build either a 2D or 3D game.



Figure 4 - 3D Game, Off-Road Velociraptor Safari

It probably will be stretch but the ultimate goal is to have our game engine support the building of a 2D or 3D game.

## What technologies?

The graphics for our game engine will be rendered using OpenGL. For a quick introduction to OpenGL see Appendix C.

OpenGL stands for Open Graphics Library. It is an API (set of libraries) for defining 2D and 3D graphics images.

## What languages can developers use to build their games?

We would like not to limit game designers or developers to using the same programming language we selected for the game engine. If we select C++ as the programming language for the game engine it will still be possible for game developers to use Java, C#, C++ or other languages.

In addition, it is quite possible to make our game engine extendable by providing support for LUA or JavaScript as an internal scripting language.

# Sparky Game Engine - Notes

---

## Open Source – Available on GitHub

All material related to this project (even these notes) will be open source and available on GitHub. In addition, we will stick to tools and libraries that are open source and freely available.

## What about images and art work?

In order to test and exercise the portions of the game engine we will need to build and “see” things working. This will require game assets such as sprites and texture models. I will stick to freely available assets but may on occasion demonstrate a feature using art assets that we have purchased. I find the website <https://www.gamedevmarket.net/> a good place to obtain decently and well-priced art assets.

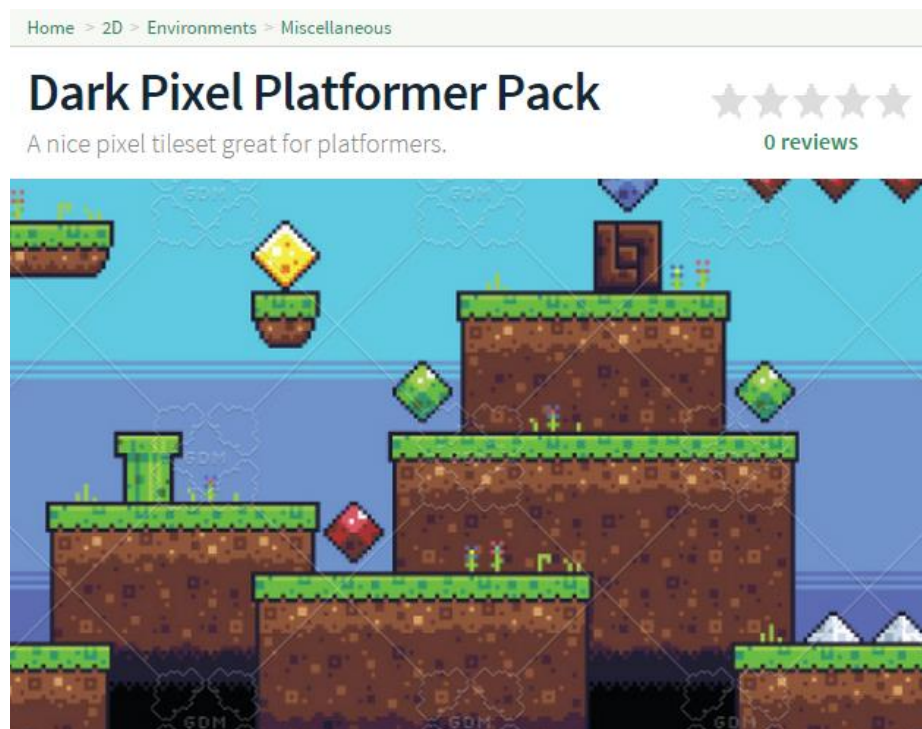


Figure 5 - My purchased art assets

I cannot make any purchased art assets available but I do encourage you to consider making a purchase and supporting the game art development community.



# Sparky Game Engine - Notes

## Appendix A – Using GitHub

GitHub (<https://github.com/>) is a popular and well-known website that developers use to host their projects. It supports individual and group efforts to manage version control for documents and software development projects.

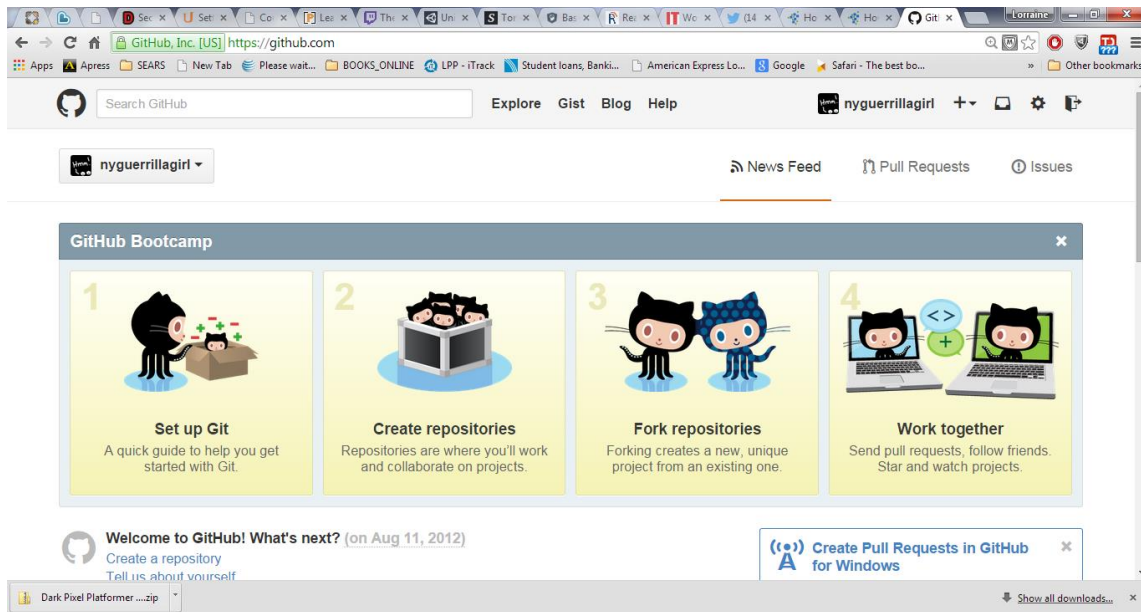


Figure 6 - GitHub Home Page

If you are a software developer still pondering the question “Why use version control?” then you either have very little experience or have yet to build software with a team of developers. If you ever had to go back and figure out “what changed about this file?” and are eyeballing it and going line-by-line looking for changes you really have to find better ways to waste your time. Version control software such as Git and a repository manager such as GitHub make these tasks manageable and seamless.

### History of Git

The version control software named Git was started by the same team that worked on the Linux kernel. The goals were to develop a tool that was fast, simple in design, supported the ability to create branches, was fully distributed and could handle large projects – Git was born.

If you are familiar with other version control systems (e.g. SVN) Git may take some getting used to.

### History of GitHub

GitHub is a web-based Git repository. It is free for use but does have paid services for companies who want to use it for their software development projects.

You can create your own repository on GitHub and then clone that repository on your own desktop.

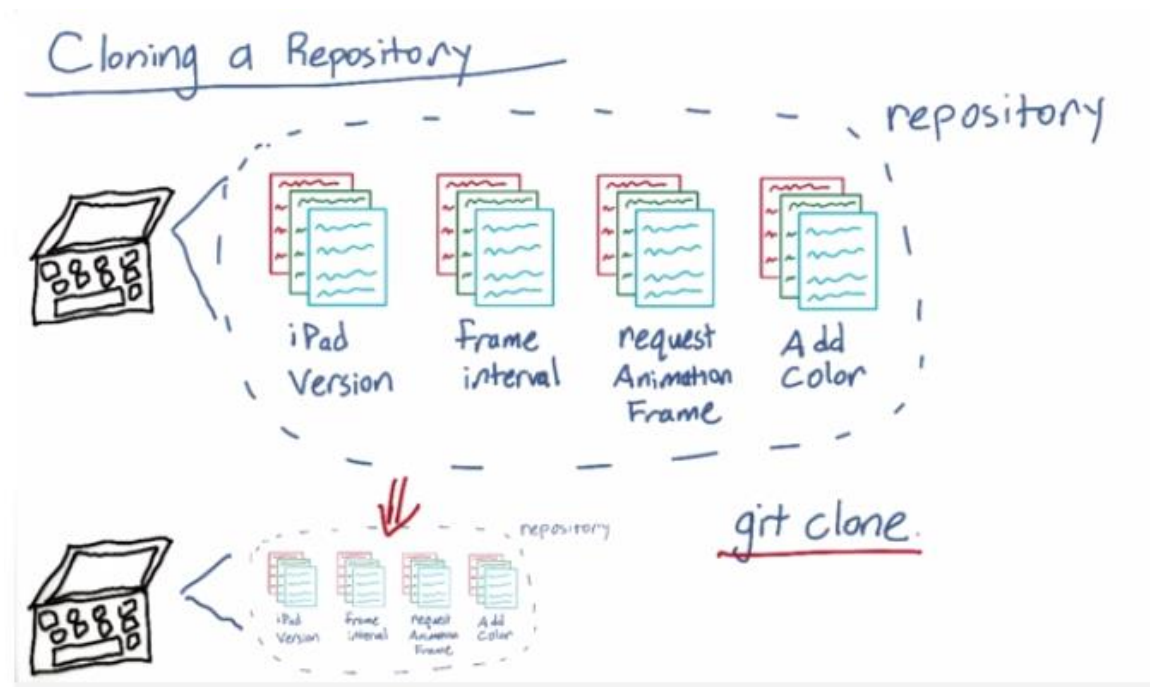


Figure 7 - Cloning a repository

If you find a repository project you like (e.g. TBD)

One typically works on a local/desktop copy of a project (usually a branch) and then push or integrates the changes into their GitHub repository.

## Installing Git Bash

## Obtain GitHub Account

## Fork or Clone a Repository

## Where to find good Tutorials

The best places on the Internet I have found to learn more about Git and GitHub are:

- <http://git-scm.com> – the host a great book on this website that you should spend a couple of hours reading.

## How I typically work on a simple project

I created a repository on GitHub to host these notes and any code I develop as part of this project - <https://github.com/nyguerrillagirl/Sparky-Engine>.



# Sparky Game Engine - Notes

---

1. I cloned the repository on my desktop (one time)
  - a.** Obtain HTTPS clone URL
  - b.** Start Git Bash on desktop
  - c.** Navigate to location of where you want to clone GitHub repository
  - d.** Issue “git clone <url>”
2. Whenever I update the software or document
  - a.** Update in git repository
  - b.** Example changes “git status”
  - c.** Add changes “git add <file>”
  - d.** Commit “git commit”
3. Upload changes to GitHub
  - a.** Issue “git push origin master”

Things get a bit more involved if I decided to branch off and test some new concepts (rather than adding to main branch).

## **Appendix B – Installing a C++ Development Environment**

## Appendix C – A Quick Introduction to OpenGL