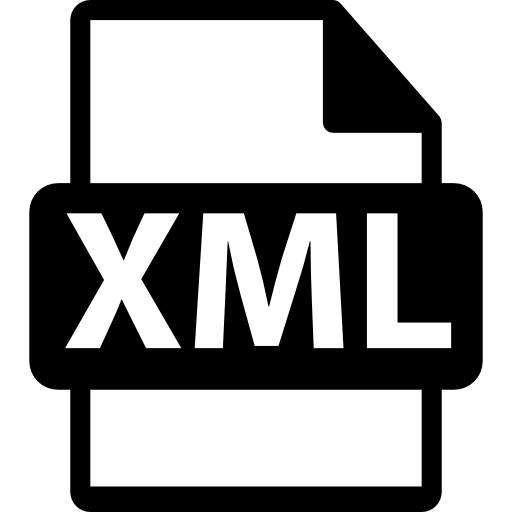
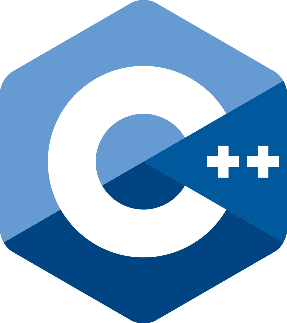
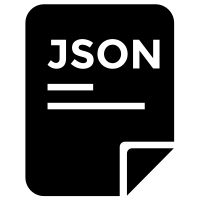
TDD.

//Index of tdd

1. Choice of Game engine
2. Code style (Code rules)
3. Platform (platform aiming for and minimum requirements)
4. Performance Budget (performance expectations)
5. Branch workflow (Branch organization of the Project on github) <https://cdn.discordapp.com/attachments/555497331848380447/681223665487970361/Sin_titulo-7.jpg>
6. Software and hardware needed (What does the team need to work on the projects as long as software (programs) or hardware)
7. Code reviews (How does someone report a bug in the code, use a template as <https://cdn.discordapp.com/attachments/555497331848380447/681230801668669447/Sin_titulo-10.jpg>)
8. Code organization overview (UML) (one UML that represents the main code structure [https://cdn.discor dapp.com/attachments/555497331848380447/681233926173360145/Sin\_titulo-12.jpg](https://cdn.discordapp.com/attachments/555497331848380447/681233926173360145/Sin_titulo-12.jpg))
9. Version list
10. Build delivery method

**Choice of Game engine**

We will use an own made game engine. We will work with the SDL2, a cross platform development library, which allows us to render graphics, process input events and much more. Some SDL2 libs such as SDL\_Mixer or SDL\_ttf allow us to work with sound and text fonts too.

C++ will be used as the main coding language. We will also read from external files such as .xml, .json or binary files for save and loading, map loading, entity info loading and more.

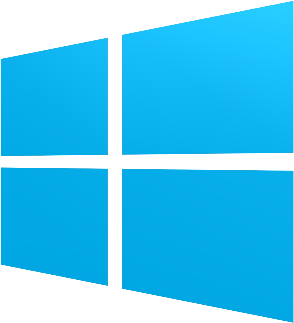
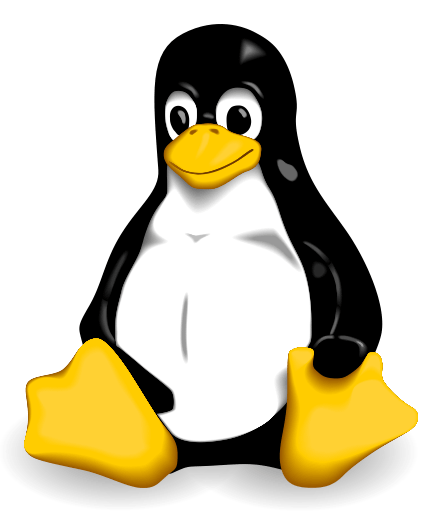
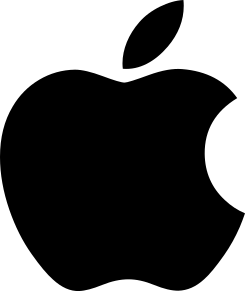
**Code style**

We’ll be working on a team, with multiple people working on the same code, that means that we cannot code “our way” and expect everyone to use time and effort to adapt to us. To avoid wasting other people time, we will follow a set of generic rules while coding, which will keep the code clean and easily understandable for everyone in the team.

The rules are the following:

* No #pragma comment(lib, “xxxx.lib”) to link libraries, libraries MUST be specified in the external dependencies fields in the project settings on Visual Studio.
* No functions definitions on .h files, definitions MUST be on .cpp files.
* The name associated with #DEFINE macros MUST be in capital letters and in case of any space, we’ll use a “\_”. Ex: #DEFINE SOME\_MACRO
* Function names MUST start with a capital letter and MUSTN`T contain ANY spaces or “\_” in the function name. Ex: DoSomething();
* If and IF statement only contains 1 line of code inside, this MUST be a line under the statement itself, we can`t define an if statement and the code inside it in the same line.
* We MUST comment functions in the .h file, at least explain in a line what the function does and what the parameters are.
* The names of functions parameters MUST clearly state what they are.
* The names of variables MUST start with a lower case letter and we MUST ignore any spaces, spaces will be omitted and variables wil be named like this. Ex: someVariable
* Variables MUST be identified with a clear/related name and/or a comment.
* When declaring a class, the code inside must follow this rules, constructor and destructor declarations, variable declarations, function declarations.

**Platform**

As stated before, SDL2 natively supports Windows, Linux, Mac OSX, iOS and Android, but our game is mainly focused to be played on Windows, Linux and Mac OSX.

A port to Android and iOS could be considered in the long run, but is not an option to be considered at the moment.

To run the game, we should have at least a minimum software and hardware requirements:

//2GB ram

//minim un i4

Windows 7 (vista?)

Nvidia de 2gb de vram I alguna de intel incorporada

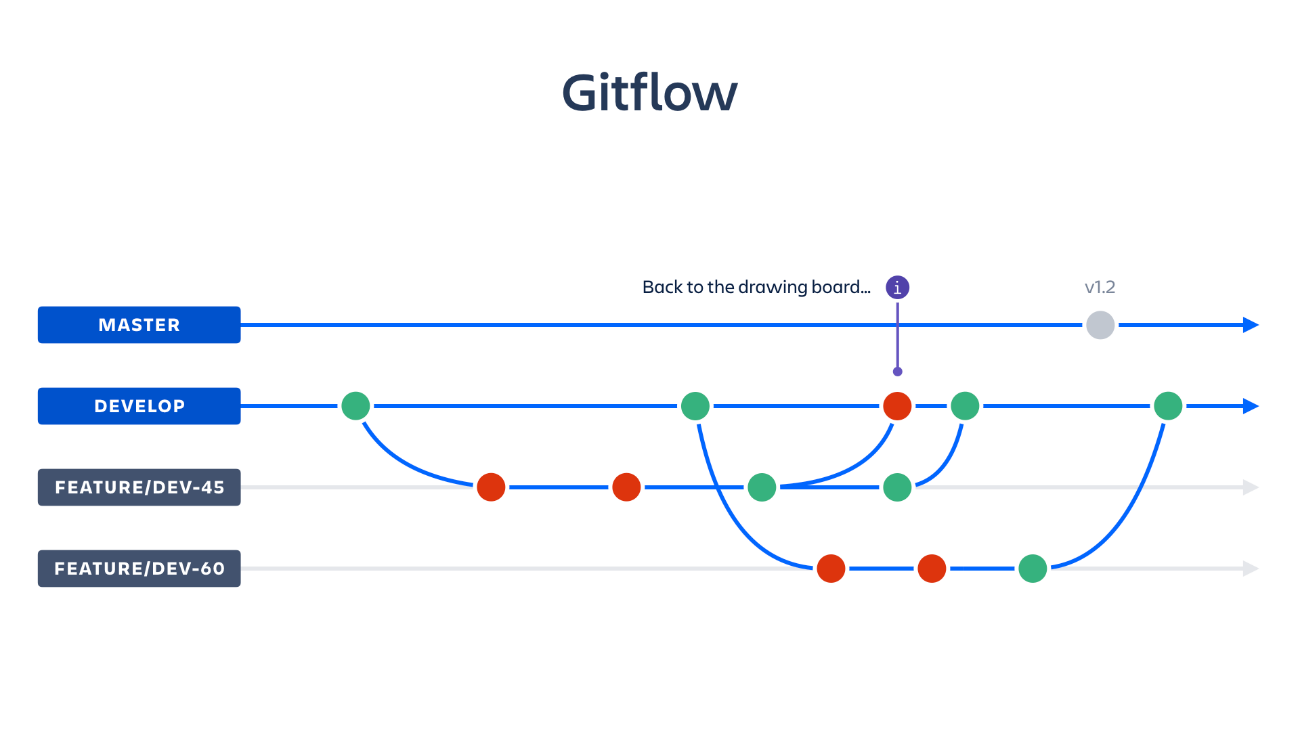
**Performance Budget**

The game is expected to run at stable 60fps with computers with the minimum requirements at software and hardware level. It is also expected to run at the same FPS across all the platforms.

The game will have the option to enable or disable VSYNC, which will cap the fps at the monitor’s refresh rate.

There is also the option to change between screen resolutions to adapt to the player needs.

**Branch workflow**

The team will work with a feature branching development system:

In this branch workflow, all the features and implementations are made externaly of the main branch and are only integrated when they are complete.

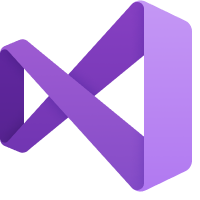
**Software and hardware needed**

**Software**

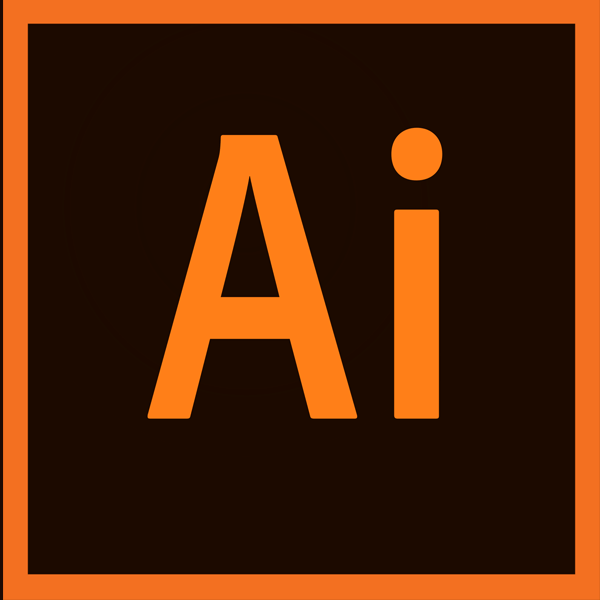
The software the team needs heavily depends on their role in the team, the software will be structured as following:

There are some general software used by all the members in the project, like Github, Slack and HacknPlan.

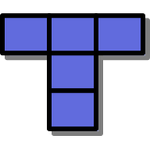
Programming

For the code we’ll use Visual Studio Community

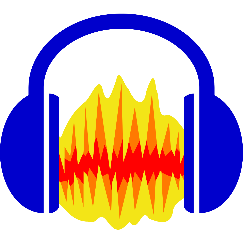
Art and UI

Art will use Photoshop, Adobe Illustrator

Design

Design will use Tiled for the maps

Sound

For the sound we’ll use Audacity

Management and QA

Managers will use Word for the files and will manage the HacknPlan and Slack pages.

**Hardware**

If we talk about hardware, all the team members will need at least a computer (Laptop or PC) to work, in the case of a PC, they will also need a keyboard, a mouse and a screen display.

**Code reviews**

When a bug is found, the person who found it has to fill and submit an Excel page like this with the following information:

The fields that need explanation are the following, the rest can easily be filled without any explanation:

**Bug ID:** Is the name of the file, which will follow the next structure “priority\_severity\_assignedTo\_date”.

**URL:** The url to the last github commit where this bug happens.

**Project build ID:** The ID of the commit on the URL.

This excel page must be exported as PDF and named like the Bug ID section and placed in the desired bug folder defined by the team. This folder will contain all the bug reports for the people programming to solve the most important ones.

**Code organization overview**

The code will be organized in the following way, this UML can be subjected to change while the project evolves:

**Version list**