BREDA UNIVERSITY APPLICATION

INFORMATION & REFLECTION

DAVID VAN SCHEPPINGEN

Project:

Compile_settings.txt	- compile information
ODTUENGINE/Dependencies	- some of the libraries used
ODTUENGINE/ODTUENGINE/src	- code written by me
ODTUENGINE/ODTUENGINE/ext	- imported code

Note:

Unfortunately the replay button after playing once doesn't work properly yet. I'm not managing the "levels" right and some memory seems to be reused. I didn't plan on implementing this, but did anyway, that resulted in this feature being broken.

Introduction

For me developing this game was mostly about developing the engine, the backbone of the game. Of course it isn't really an engine, more a sort of framework, but in the rest of this document I'll be referring to it by the term engine.

In my 4 years studying at Grafisch Lyceum Utrecht, and the years of experimenting before that I've build a lot of games and prototypes (which can all be viewed on my portfolio site https://davidschep.com/). I've never programmed a game this low level before though. Most of my games have been developed with Unity (C#), Unreal Engine, Java and JavaScript.

Because of this I've faced a bunch of new challenged I had to solve, which definitely gave me some insight at what is going on when I'm using the other methods to develop games.

A month before I started this project I started my internship at TaleWorlds Entertainment, working on Mount & Blade II Bannerlord, these 2 projects went hand in hand greatly, since I could apply all the techniques I learned in one of the projects to the other one. In the Bannerlord project I was mostly responsible for programming the animation system, combat and quests in the engine part of the project.

About the game

The game, named ODTUENGINE, after the university I was staying at while developing the project, is about driving around with a jeep, shooting of the bears, until the round of fighting is over. Intentionally the game was meant to be about dinosaurs, but since I couldn't find any proper sprites for those they turned out as bears (this is why in code they are referred to as dinosaurs).

There are 5 different bears, with 4 different weaknesses, specific ammo is used for specific weaknesses (as shown in the tutorial). These weaknesses show in the reflection of the water when a bear walks over it, to achieve this, the player will be leading the bears over the water. The player has to survive until the helicopter comes to pick him up, if he manages that, the game will be won.

I'm happy with how simple the concept of the game turned out, I would have loved to put more game design techniques into it though. Playing the game could feel like so much more fun if more time and effort would be dedicated to developing systems to support things like better feedback to the player, balancing and level design. The lack of clear art and animations at times has a negative impact on the clarity of the game as well.

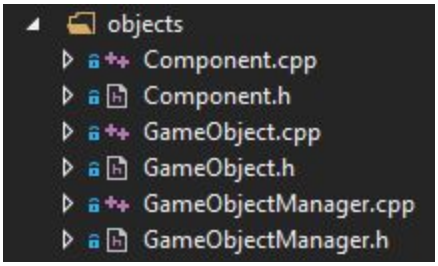
This all concluded in me not being able to show off my game design skills this project.

Official tutorials of API's were used, next to this ChiliTomatoNoodle's video on SSE Optimization and Direct3D and TheCherno's C++ lessons where needed.

Most of the art is made by others, credits and names can be found in the start screen of the game, here are some more detailed links:

Jeep - https://opengameart.org/content/military-jeep
Background – game: Rimworld
Bears - https://grandmadebslittlebits.wordpress.com/tag/rpg-maker-vxace-animals/
Armored bear - https://nl.pinterest.com/pin/787707791046383260/?autologin=true
Font - https://www.dafont.com/market-deco.font
Helicopter - https://i.pinimg.com/474x/46/cf/93/46cf930d8a02fcc496ac231b1c30d426--slug-helicopters.jpg

Code and development process



Component & GameObject system

After programming most of the core engine part I decided that I needed a component based system to avoid code duplication and provide easy expansion in functionality.

```
GameObject* tPlayer = new GameObject("Player");
tPlayer->AddComponent(new Player());
```

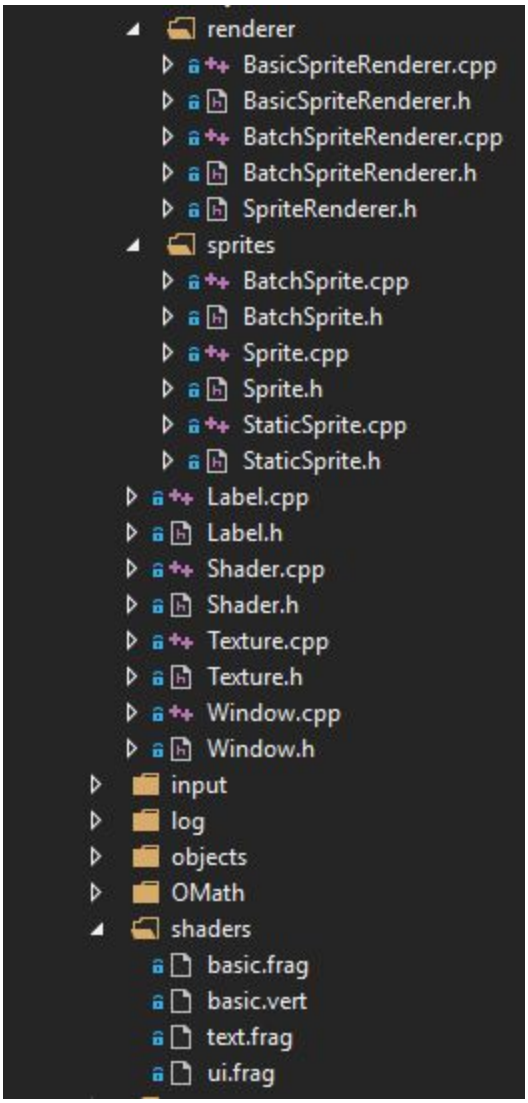
I based the logic of the system on how the game engine Unity does it, since I was always quite pleased with that. The trick was to keep this simple as possible, and I mostly succeeded in that, although, if I would have some more time I would refactor this system to work a bit better.

```
8   class Component
9   {
10  public:
11      Component(GameObject* iParent = nullptr);
12
13      virtual ~Component() { this->_parent = nullptr; }
14
15      virtual void Awake() {}
16      virtual void Update(float iDeltaTime) {};
17
18      GameObject* GetParent() { return _parent; }
19      void SetParent(GameObject* iParent) { this->_parent = iParent; }
20
21      int GetComponentId() { return _componentId; };
22
23      // I would like to do this a little more proper and not have it in the Component class
24      virtual void OnTriggerEnter(PhysicsBodyComp* iPhysicsBodyComp) { }
25
26  protected:
27      int _componentId;
28
29  private:
30      GameObject* _parent;
31      bool _active = true;
32  };
```

Rendering

For my sprite rendering I wrote a simple and a batch sprite renderer, resulting in my game running at a ***whopping*** 5000fps on this pc (i7-6700 & GTX 670). I mostly used the GLFW docs to help me out with this.

As soon as I build my component system I turned the BatchSprite class into a component. This still had a lot of improvements and extra features that I could work on, but I unfortunately didn't get around to do that.

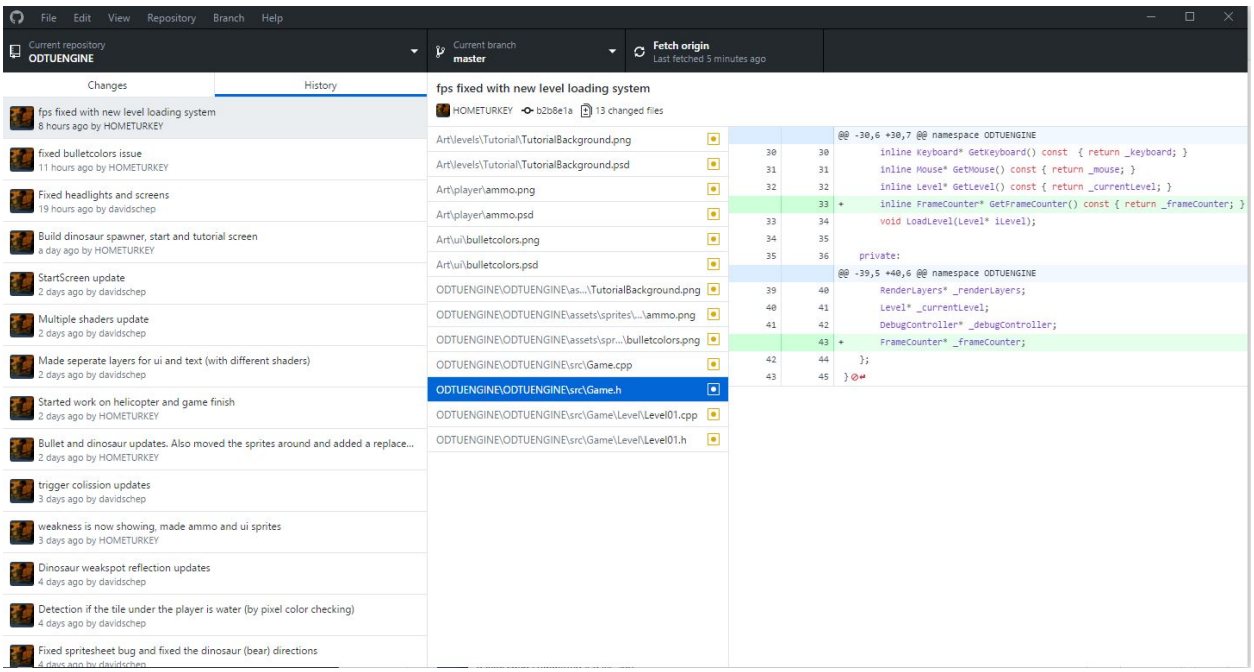



```
void Player::Awake()
{
    _machineGun = new GameObject("Machine Gun");
    _machineGun->AddComponent(new MachineGun());

    GetParent()->AddComponent(new BatchSprite(GetParent()->position.x, GetParent()->position.y, 1.f, 1.f, RenderLayers::INSTANCE->texturesCar[_previousJeepRotation]));
    GetParent()->AddComponent(new PhysicsBodyComp(Game::INSTANCE->GetLevel(), GetParent()->position, Vector2(0.7f, 0.7f), Vector2(1.5f, 1.5f)));

    _lastDirection = Vector2(0, -1.f);
}
```

Source control

I started out with SourceTree, since I used this when I was a programmer at GainPlay Studio. I used sourcetree for a few days before I switched over to Plastic SCM, since I became such a big fan of that software after working with it at TaleWorlds Entertainment.

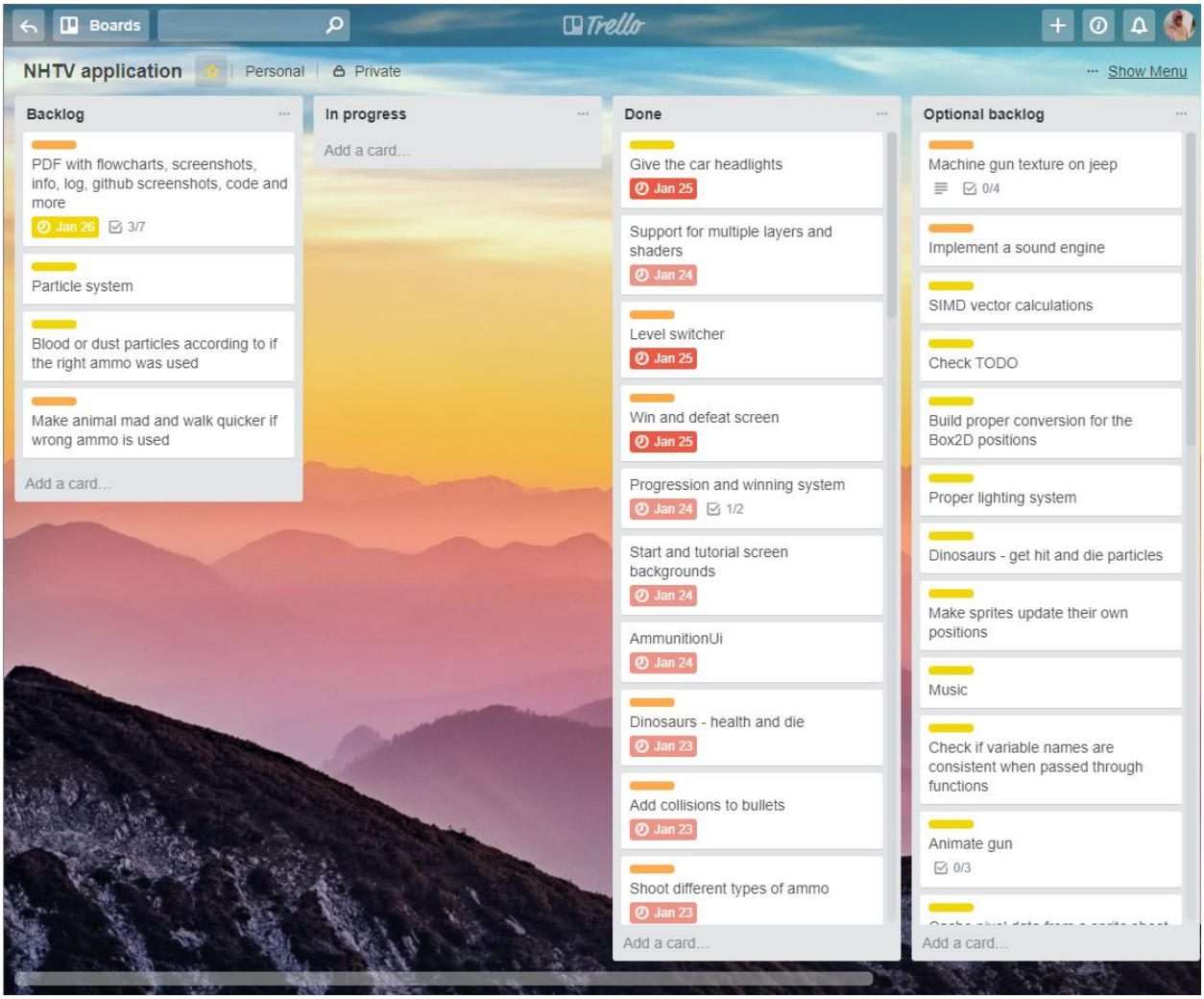
Later I found out that using Plastic SCM for a small, 1 man project like this wasn't ideal. Plastic SCM needs a server that's running, maintenance, licensing and more. After a few days I switch the GitHub desktop client, and kept using that for the rest of the project. I preferred GitHub here since the software is so simple and quick, sometimes it needs a little help from the command line (to go back to an old commit for instance), but other than that it is great!



Scrum using Trello

I decided to use Trello as scrum tool, which for me is the software I usually use for projects. I looked into some other stuff, but didn't find anything that worked out of the box and that didn't require a lot of time each time I wanted to change something. Especially in later stages of development Trello proved itself to be very useful.

At TaleWorlds we use Jira, which is also a fantastic tool in my opinion, but doesn't serve a small project like this very well.



Work diary

Wednesday November 1st, day 1 of 87

Applied at NHTV

Not much time to work on the project yet, especially on week days because of the 10-hour work days at TaleWorlds Entertainment, and the cleaning, cooking and washing I had to do since I moved to Turkey.

My laptop broke down this week so I wasn't able to work on the project before the 4th of November.

Screenshot 'finished' game:

Saturday and Sunday November 4th and 5th, day 4 of 87

This weekend I was researching what libraries, API's, etc. to use, eventually I made a short list:

API's & libraries:

Direct3D	Graphics library
OpenGL	Graphics library, performance is a lot better than SDL graphics
SDL	Cross-platform development library
GLFW	OpenGL Framework, does input and such like SDL but less unwanted stuff like rendering (since I want to do this myself)
GLEW	OpenGL Extension Wrangler Library, loads OpenGL functions
SFML	SFML provides a simple interface to the various components of your PC, to ease the development of games and multimedia applications. It is composed of five modules: system, window, graphics, audio and network. Multi-platform and multi programming language, works with OpenGL as well After a quick look at it I concluded that this is not really ideal for game programming
CLI	Makes a bridge between C++ and C# code

Combinations interesting for me:

SDL	SDL already does 2d rendering
SDL + Direct3D	
SDL + OpenGL	
GLFW + OpenGL	GLFW doesn't do rendering

I made the decision that using SDL in combination with Direct3D would be the best choice, and immediately started programming with SDL.

I created a project did some setup and experimenting, but after not that long of programming and doing more research, I found out that Direct3D might not be the ideal interface to do this 2D game.

Monday 6th until Friday 10th of November, day 6 of 87

This week I experimented a little with the use of Direct3D, the more I worked with it, the more the idea came to mind to switch over to GLFW in combination with OpenGL.

Saturday and Sunday November 11th and 12th, day 11 of 87

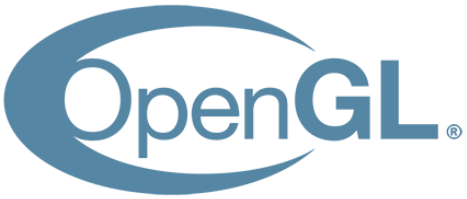
I made the decision to indeed switch over to GLFW in combination with OpenGL.

I started over, created a new project and did all the setting up.

I did this with use of the GLFW Getting started guide:
http://www.glfw.org/docs/latest/quick.html

I also installed GLEW to access OpenGL functionality.

I build functionality for the keyboard and mouse input, I setup a basic log system and created a resizable window.



Monday 13h until Friday 17th of November, day 13 of 87

I shook the system around a little and experimented the flow of the program, which would instantiate what, and how components would access each other. Eventually I settled down on a constructor where the core components would talk to each other through singletons, meaning that for instance the **mouse**, **keyboard** and **window** are each a singleton.

Saturday 18th until Friday 24th of November, day 18 of 87

In this time, I started writing some math classes, vector and matrix calculations (adding subtracting, multiplication etc.). I also switched my source control, I initially started the project with Sourcetree and Bitbucket, since I was familiar with this combination from my work at GainPlay Studio. Although I never really liked Sourcetree because of its many bugs I used it anyway (because of the easy setup and many features).

While using Sourcetree I began to annoy myself about the bugs even more, having used the fantastic source control Plastic SCM at TaleWorlds Entertainment, and I decided to switch over to Plastic. Plastic brought a new range of issues along with it though, main being that Plastic SCM needs its own dedicated server. After some experimenting I realized I was spending too much valuable time on Plastic SCM setup, that I could have spent developing the game.

It was because of this that I decided to switch to straight forward GitHub, since this works nicely, and doesn't need any work done to it.

Saturday 25th until Thursday 30th of November, day 25 of 87

In between this time, I created my 2D renderer, a simple on, not really focused on performance. I had some issues along the way, but nothing too bad of course. This renderer also made use of opengl vertex and fragment shaders.

Friday 1st until Monday 4th of December, day 31 of 87

In this time, I made a more efficient sprite renderer that rendered the sprites in batches, hugely increasing performance.

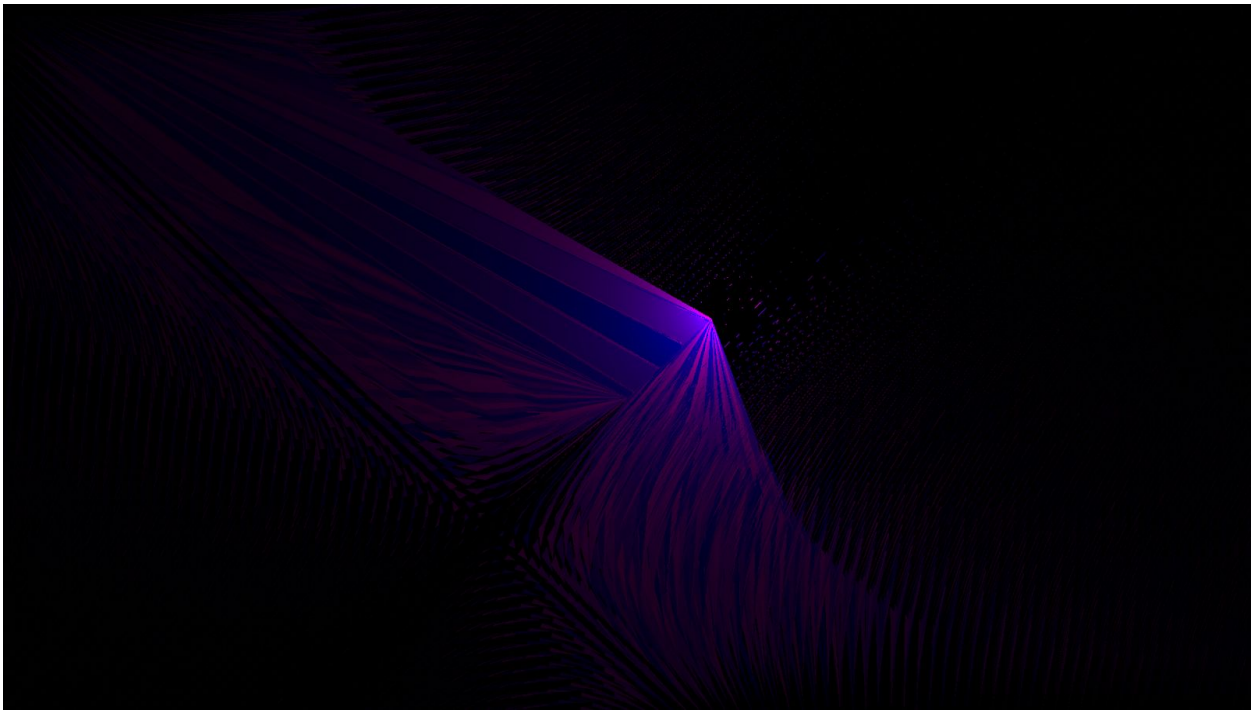
Tuesday 5th and Wednesday 6th of December, day 35 of 87

I added the possibility to make sprites children of other sprites, so that the transform (position, rotation and scale) are based on the one of the parent(s). In the future I want to create a system where this is managed by game objects instead of the sprite.

Thursday 7th until Sunday 10th of December, day 37 of 87

I implemented the FreeImage library (also used by the game engine Unity) and added code to be able to load files as images, in the future I should add support for more types of images, currently it only supports 24-bit images.

Screenshot of the broken sprite renderer, caused to a wrong matrix calculation, resulted in quite a pretty picture if I may say so myself



Monday 11th until Thursday 28th of December, day 41 of 87

In this time, I implemented string rendering, for this I used the freetype and freetype-gl libraries. This took a lot longer than expected because of a lot of problems with outdated code (deprecated OpenGL functions).

I also had a hand injury that made me unable to use my right hand for about a week. Because of the programming downtime I had a little more time to focus on the design of my game. I brainstormed a lot, and eventually came up with an idea that I liked, that fitted in the theme and that seemed possible to realize in the month that I had left.

Thursday 4th until Tuesday 9th of January, day 65 of 87

In these days I was mostly busy with my internship paper about my internship at TaleWorlds Entertainment for Grafisch Lyceum Utrecht, so I didn't get around to do too much programming.

Wednesday 10th until Thursday 11th of January, day 71 of 87

I did a bunch of updates on the rendering and file loading.

Friday 12th until Tuesday 16th of January, day 73 of 87

There wasn't a lot of time left at this point, I did most of the things needed for the engine, and started working on the game itself. I would have liked to do more programming on the engine before I started working on the game, since this would have resulted in cleaner code and less duplication, but at one point I had to start working on the game.

I programmed the driving jeep and machine gun on the jeep.

Wednesday 17th, day 78 of 87

I improved my component system, making the code more modular.

Thursday 18th, day 79 of 87

Here things became a little busier, at TaleWorlds Entertainment we entered a 2 week crunch period so that we could improve on the combat in Mount & Blade II Bannerlord. This resulted a switch from a 45 hour workweeks and plenty free time to a lot less free time…

Although the crunch was going on I still managed to finish up my component system and implement Box2D as physics system.

Friday 19th, day 80 of 87

I build a simple wrapper around Box2D so that it would work with my component system (if I would have had more time I would have done this more properly, like mentioned in *"Friday 12th until Tuesday 16th of January"*).

I also created a debugger for my Box2D stuff, it isn't fully operational and has some bugs, but it does the job.

Then I build the colliders in my scene, needed for the player, bears and bullets.

Saturday 20th and Sunday 21st, day 81 of 87

This weekend I started working on the collision for object. Next to this I've started on the basic ai for the bears.

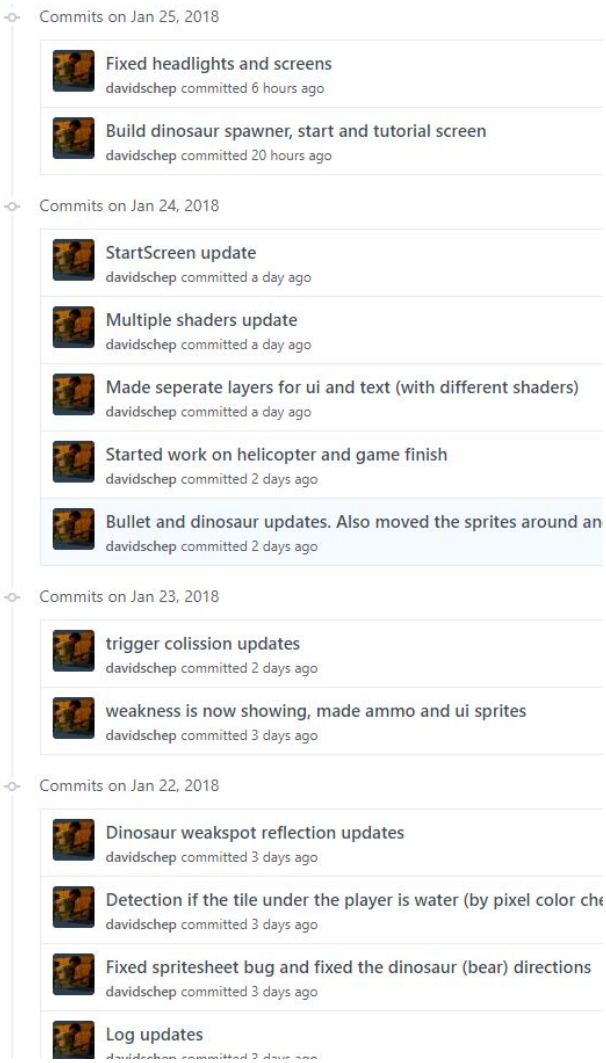
I also added spritesheet support to the engine, this should be made more efficient at some point in time.

Monday 22nd, day 82 of 87

I worked on making the bears actual enemies and making sure they got spawned and were affected by their weaknesses.

Tuesday 23rd, day 83 of 87

Trigger collision is now supported by the physics body components. Same as before, I could do this a whole lot better but there was no time for that.



Wednesday 24th, day 84 of 87

Created start and end screens, separated the game, ui and text layers and added support for multiple shaders. Also made an ending to the game.

Thursday 25th, day 85 of 87

Added the lighting and the car headlights to the game, this could be programmed better but I needed quick results. Also build the dinosaur spawner.