Technical Design Document (TDD)

# Technical Goals

We are trying to achieve a product that runs performantly on most Android recent mobile devices. For that we need to become familiar with the unique requirements and limits this platform poses. We for example cannot use many post processing functions in the same way as we would use it on a desktop device. To make our stylized art work the way we want to, we will have to find suitable solutions for mobile platforms.

To make the game feel great, we also want to make full use of the responsive capabilities of mobile phones, such as audio and vibration.

## Technical Risks

The obvious risk here is that we might be able to make it look as good as we expect. With the shader materials we are using at the moment, lighting becomes very difficult. We might have to rethink our approach to creating 3D environments for this project.

On the coding side, we will not encounter greater roadblocks. The game is fairly simple and we do not need any complex systems. The main obstacle would have been porting the product to mobile. Creating an executable .apk was a challenge and we continue to encounter problems with this as the development progressed.

## Platform and Software

|  |  |
| --- | --- |
| Platform | Android for Mobile |
| Engine | Unreal Engine 5.2 |
| 3D Software | Autodesk Maya |
| 2D Software | Adobe Photoshop CC  Adobe Animate CC 2023  Adobe After Affects CC (TBD) |
| Sound Software | Ableton Live |

## Code Conventions

### Naming Conventions

Every file being used in the context of the project, should be named in a specific way to offer information on the first glance. You might be able to see through all your FirstDraftFinal2 copy copy.jpg but somebody might have to work with it later on who isn’t. To avoid confusion and to locate files easier we will apply naming conventions. This is an open list and you are welcome to extent it at your own convenience.

|  |  |  |  |
| --- | --- | --- | --- |
| **Type** | **Usage** | **File Format** | **Conventions** |
| Unreal Blueprint | Code Snippet in Unreal | .uasset | BP\_WordsLikeThis |
| Texture File | 2D Sprites to use with materials or Paper 2D in Unreal | .png | Object\_Detail\_State\_Numerator  E.g.: Pine\_Leaves\_Red, Pine\_Bark\_3 |
| Animation Assets | Frame sequences to create an animation in Unreal | .png (sequence) | **Player\_**State\_MaxFrame\_CurrentFrame  **Enemy\_**Name**\_**State\_MaxFrame\_CurrentFrame |

### Source Management and Branching Policy

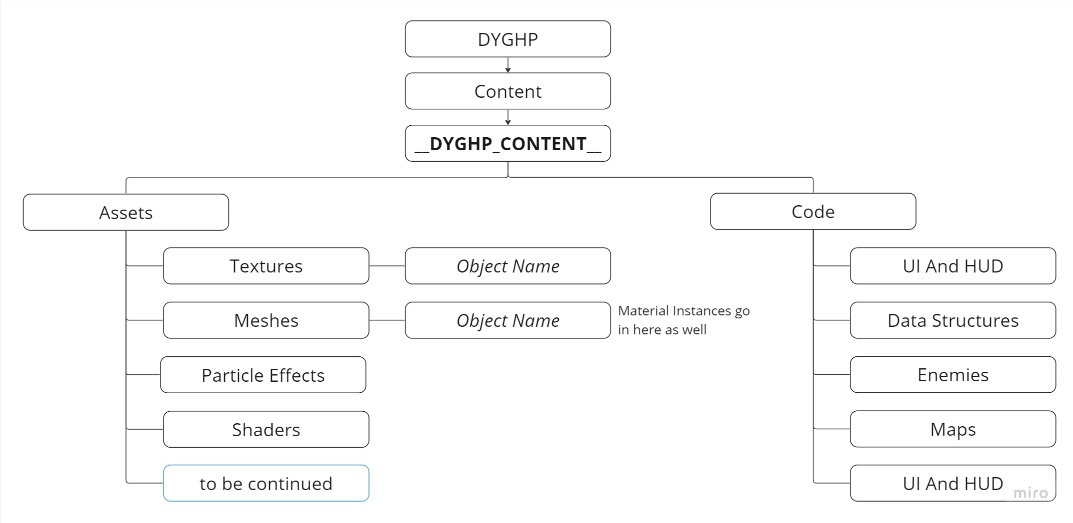
We are again using the double repository approach by having one rep for the game project files and one for the assets. We use GitHub and every member of the team has access to both repositories at all time. Especially for the project files, it absolutely necessary to maintain immediate communication about checking out files to avoid losing work progress. Please also check out the guide to using GitHub which is linked [**here**](https://crunchybits.atlassian.net/l/cp/1vFN1fBQ).

Starting from the first playable we will also introduce a new branching policy: We will maintain two different branches of the game project files. One is always kept as the latest working version of the game. It is gonna be called the **Main Branch**. Then we will have a **Development Branch** on which all development is being done. This way we will always have a presentable, working version while we can do whatever we want on the Development Branch.

### File Structure

In the asset repository every team member has a folder with their name on it. Inside, you should have folders for every file type you have saved there. Inside these, you can organize further if you see fit. Make it easily understandable for others, because they might need something from your folder when you are not there to guide them through a folder labyrinth.

In the engine you find everything we do in the content folder and then \_DYGHP\_CONTENT\_.



# Performance Budgets

Since we are unfamiliar with the platform and its limits, we have to tread carefully with this. I might be the safest bet to design with performance in mind first. That means we need the lowest possible amount of vertices on our models from the get-go and we can not rely on light shenanigans to make the game look cool.

Here are some estimates of what is within of our limit. Please expand this list with new information.

|  |  |
| --- | --- |
| **Type** | **Encouraged Limit** |
| 3D Models | 2000 Vertices |
| 2D Animation Sprites | 20 kb |
| 2D Background Sprites | 50 kb |

We had to reduce the polygon count of the 3D environments several times to save processing power, in the end we managed to replace a lot of the meshes with 2D planes which solved many of our issues. We also decided to reduce the resolution of the sprites and heavily compressed them to make them load faster.

So far, this seems to be sufficient to make the game run quite well on all recent devices.

# Art

## Programs

|  |  |  |
| --- | --- | --- |
| **Type** | **Name** | **Usage** |
| 3D | Autodesk Maya | Creation of 3D background assets and UVs |
| 3D | Substance Designer | Generating Materials |
| 3D | Substance Painter | Creating materials and applying textures |
| 2D | Adobe Photoshop | Concepting for characters, background and style Illustration for character designs Additional illustrations for backgrounds & press-kit |
| 2D | Adobe Animate | Frame-by-frame animation of characters and exporting image sequences |
| 2D | Adobe After Effects (TBD) | Possible adjustments of character animation with puppet animation / puppet pins (TBD) |
| Implementation & Level Design | Unreal Engine 5.2 | Implementing animation and assets Level Layout & Scene composition |
| 2D | Adobe Illustrator | Vector Graphics for UI Element designs |
| 2D | Figma | Documenting Design system for UI |

## Limitations

|  |  |
| --- | --- |
| **Type** | **Limit** |
| 3D Asset | 2000 Vertices |
| 2D Character Animation: Coloring | Flat colored, if necessary max. 1 layer cell-shading |
| 2D Character Animation: Framerate | 24fps |
| 2D Character Animation: Scale | Full canvas size: 2560 x 1440 Characters are not moved in the engine and simply animated in the scene in Animate. |

## TimeLines

A screenshot of a computer program

Description automatically generated

### Approval Pipeline

A yellow diamond with black text

Description automatically generated

### Crunch Timetables

A diagram of a environment crunch

Description automatically generated

A screenshot of a computer

Description automatically generated

## State Machines

A diagram of a flowchart

Description automatically generated

## Naming Conventions

|  |  |  |  |
| --- | --- | --- | --- |
| **Type** | **Usage / Definition** | **File Format** | **Convention** |
| Concept Art | Mass-production of greyscale sketches only for design iteration | .png; .jpg | / |
| Character Design | High resolution colored Illustration showing the subject from several side. Used as reference for animation. | .png | Type\_Name\_MM\_DD  E.g.: Enemy\_KasaObake\_01\_27 |
| Animation Asset | Sequence of animation frames, to create an animation in Unreal. | .png (sequence) | **T\_Player\_**State\_MaxFrame\_CurrentFrame  **T\_**EnemyName**\_**State\_MaxFrame\_CurrentFrame  E.g.: T\_Akaname\_AttackHigh\_0012\_0000 |
| 3D Asset | 3D background assets being used in Unreal to create the environment | .fbx | **Asset**\_Detail\_Level  E.g.: Asset\_House\_01 |
| Texture | 2D Sprites to use with materials or Paper 2D in Unreal | .png | Object\_Detail\_State\_Numerator  E.g.: Pine\_Leaves\_Red, Pine\_Bark\_3 |

# Guides

## GitHub

GitHub is our central data storage and source control. It is not only used to share assets and files efficiently, but also makes it possible for us to work on the same version of the game from different computers.

There are two repositories:

[awesomemobilegame](https://github.com/scrollmops/awesomemobilegame)

[awesomemobilegame\_assets](https://github.com/scrollmops/awesomemobilegame_assets)

The first one contains the unreal project with all its assets, the second one holds everyone's asset folder.

To access these, please follow this guide:

1: Log into your GitHub account.

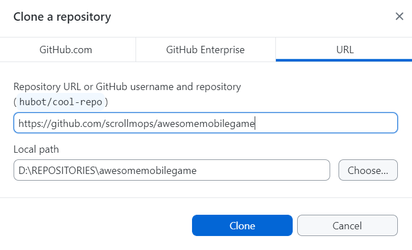
2: Download the GitHub Desktop App.

3: Set up the app and connect it to your account.

4: Click on the links above to get to the online repositories. Copy the URL in the browser (repeat the following steps for each)

5: In the app, click on “File” and then “Clone repository”.

6: Choose URL in the top and paste the link. Pick a folder on your pc to contain your local files.



### **How to use GitHub**

There are a few things you have to be aware of using a repository.

The files are saved at a central location and if someone takes one of these files to work on it, they apply their changes to a locally saved copy of that file. When done, they will go to the GitHub app to replace the centrally stored version of the file with their local one. This will delete the centrally saved version.

So this is important to understand: When someone works on a file and someone else decides to work on that same file at that same time, problems will occur.

If the first person finishes their work and uploads their version and then the second person does the same, the changes of the first person will be overwritten. For that reason, it is absolutely necessary to communicate about what you are doing, before doing it.

* **If you need to work on a file, talk to the person who has worked on it before first**

The app will help you not fucking anything up by telling you when there are issues. If that happens, never try to fix the problem yourself (unless you know what you are doing), always ask somebody for help.

Now let’s talk about how you use the GitHub App:

A screenshot of a computer

Description automatically generated

1: Make sure you picked the right repository.

2: Always fetch first to see if any changes have been made in the central storage before you do anything else. If so, the button will change to “Pull”. Click it to download all changes from the central storage to your local one. If there are any issues with pulling, talk to someone to get help.

3: The changes will be listed here. If needed, you can discard individual or several changes by right-clicking them. They will then not be added to the central storage, but your local files will also be overwritten with the central version. Only do this if you want to revert those changes.

4: If you have more than one change, you will have to put in a summary in this field, before you can commit them. Try to be brief but somewhat descriptive. You can leave the description field empty.

5: At last, you have to commit your changes and click push, which will then appear on the button where you fetched and pulled in step 2.

## Light in Unreal

#### Light Components

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Description** | **Mobility** | **Performance** |
| Directional Light | Works like sunlight. Usually applied in all scenes. Only use one. | Static for better performance, stationary if you want to cast shadows for movable objects | Great when static |
| Point Light | A small dot that spreads light in all directions. Your first choice. | Can be anything, depends on use case. Don’t use more than 3 close together when movable or stationary | Decent, better when static |
| Spot Light | A small dot that spreads light in a cone. Specific use cases. | Same as Point Light | Decent, better when static |
| Rectangle Light | A rectangle which emits light. Good for big lights like neon tubes. | Should be static | Not great when not static |
| Sky Light | Filler light, lightens up the shadows for a realistic ambient light. | Normally static, movable if you want to use distance field ambient occlusion | Decent when static |

#### Mobility

|  |  |  |
| --- | --- | --- |
| **Static** | **Stationary** | **Movable** |
| A static light gets baked into the environment once the game is loaded. That means once it is built, It does not affect performance anymore. It also means, that a static light does not cast shadows on moving objects. Only use to light out fixed geometry. | The go-to if you want to cast shadows on moving objects. Stationary means the light itself is fixed in place, but it will cast real-time shadows with every solid geometry. Avoid stationary lights to overlap cones, there should be a maximum of three overlapping. | Movable lights can change position and will cast real-time shadows on moving objects. Don’t use this unless you have a good reason. Typical use case are hand-held torches. Movable lights are very performance hungry. |

### Lighting a Scene: Workflow

* Add **skybox** (Plugin: GoodSky)
* Add **directional light**
* Add **sky light**
* Add **exponential height fog** and **atmospheric fog**
* Add **post process volume** → set minimum and maximum brightness to 1 and check “infinite extend”
* Add **light mass importance volume**
* Adjusting the skybox
  + Choose a light color fitting for your environment
* Adjusting the directional light
  + Intensity → Night time: ~0.5, Daytime: over 9000
  + Adjust rotation
  + Check “atmosphere sun light” for light shafts (needs fog)
  + Check “light shaft bloom” for light shafts
  + Adjust bloom scale
* Adjusting the sky light
  + Set the light so it fits with your setting if there was no direct light in it.
  + Adjust intensity scale
  + Choose a light color fitting for your environment
  + Choose lower hemisphere color (bounce light) → Color picker on floor textures
  + Set to movable if you want to use distance field ambient occlusion
  + Adjust DFAO → Max distance: as low as possible, Contrast: cheap way to increase effect, Exponent: low
  + Add **sphere reflection capture** → put in center of scene
* Adjusting the exponential height fog
  + Fog density
  + Fog height falloff
* Add **static lights**
* Add **stationary lights**
* Adjusting static and stationary lights
  + Uncheck “inverse squared falloff”
  + adjust or turn off volumetric scattering (fog)
  + adjust attenuation radius (the smaller the better for performance)
  + adjust light falloff exponent
  + adjust intensity
  + readjust attenuation radius
  + readjust light falloff exponent
  + check “use temperature”
  + adjust temperature (or use light color if needed)
* Readjusting the sky light
* Adjusting the post process volume
  + Adjust color grading
  + Rendering features → Ambient occlusion: low radius (~50), play with intensity (For better view: View mode → Buffer visualization → Ambient occlusion)
  + Adjust gamma
  + Optional: Uncheck “infinite extend” and adjust box size
* Readjusting the sky light again

### Optimization and Analytical Tools

* Open the view modes in the upper left corner of the viewport (“Lit” is default)
* Go to optimization view modes
  + Light complexity → Blue good, green okay, yellow to orange still alright, dark red to white: no good. The more light cones overlap (stationary/movable), the less performant.
  + Lightmap density → Same color codes, but also balance performance with shadow detail, performant: ugly. Change light map resolution in mesh details.
* To see the effect of static lights and many of the different components in the workflow, you will have to built the lights manually. Whenever you change a setting, go to the “Build” drop down menu at the very top of the unreal window (left side) and click on “Build all levels” or only build the lights in the option below. It will then prerender all lights.
* The quality is set to “Preview”. To see what it would look like in game, you can change the lighting quality in the beforementioned drop down menu.

## How to setup a Location

Go to **Content/\_\_DYGHP\_CONTENT\_\_/Code/Maps/GameMaps** (Subject to change) and rightclick **MapTemplate** then click on duplicate. Name the file as you want it and save it.

Open the map and you are greeted with a mostly black screen. Follow the next steps before you do anything else:

##### **1.: Set up Materials and Meshes**

Create a Folder for your location in **Content/\_\_DYGHP\_CONTENT\_\_/Assets/Textures/RuntimeVirtualTextures**

and create instances of

**MM\_RVT\_Foliage** and **MM\_RVT\_Landscape**

and name them according to your location (MI\_RVT\_Foliage\_YourLocationName)

then put them into the folder you just created.

Then make two copies of **RVT\_Template**, call one RVT\_Color\_YourLocationName and one RVT\_Height\_YourLocationName and put them in your folder. Go to their details and make sure that

Open your Foliage Material and **activate the colorRVT** checkbox. Apply your RVT for the color here.

You can also apply a wind effect for the foliage in the parameters above.

Now for the meshes: Go to

**Content/\_\_DYGHP\_CONTENT\_\_/Assets/Meshes/Grass**

and create a folder for your location named GrassType\_YourLocationName. Then pick the type of grass you want and **make a copy of each of the MAKECOPY meshes**. Name your copies according to your locations name and put them in your newly created folder. Go into the folder and assign each mesh the MI\_RVT\_Foliage material instance that you created earlier.

##### **2.: Create a landscape**

A screenshot of a computer

Description automatically generated

Click on the drop-down menu next to the save icon and choose **Landscape Mode**. Click on “New” and choose your **landscape material** you created in step 1. Then look beneath and change the number of **Components** to 4x1 or 4x2, depending on the depth you need. I also recommend to increase the **Section Size** to 127x127 for better painting and sculpting (skip for better performance). If you do this, you have to scale the landscape down by 50% in the X and Y scales. Make sure that the Z scale stays at 100.

A green grid with white dots

Description automatically generated with medium confidence

Move the landscape so that it begins right behind the camera. You also have to move it down a bit**( -0.25)**, so the player character does not sink into the ground. At last, you move the landscape to the right to **8500**. You can apply some random material. If everything looks like on the picture below, click on “Create”.

A screenshot of a computer

Description automatically generated

Next you have prepare the landscape for being painted on. For that, you go to the “Paint” section of the landscape editor (at the very top). At the bottom you will see the available colors and next to them there is a plus icon. You will have to click each of them and **create a Weight-Blended Layer (Normal)**. Make sure the asset is saved in the folder you created in step 1 and click save (ignore the “sharedassets” folder). Also be aware that the first layer you create, will fill up the whole map. You can then paint over it by selecting the icon of the paints.

When your landscape is created, the first thing you should do is doublecheck if you have the right material applied (MM\_RVT\_Landscape\_YourLocationName). Then click on the landscape in the outliner and look for “draw”. **Add an array element and apply your RVT\_Color texture**.

Uncheck **Cast Shadow** while you are there.

A screenshot of a computer

Description automatically generated

Next you have to look for the folder RTV in the outliner and **apply your virtual textures** to the according volumes. You also need to go to the **Transform from Bounds** section in both volumes and click on the color picker icon to then click on the landscape in the viewport. Click **Set Bounds** afterwards.

At last you can go to the Foliage Editor and pull all your grass meshes into the panel. Make sure to **deactivate Cast Shadow** for all of them in the options below.