EWI3615TU - CS Minor project 2020/2021 [Q2]

Building your INDIE GAME

VERSION 1.2 (10/11/2020) - (possible later changes will always be indicated in red)

Rafael Bidarra

INTRODUCTION

This document describes the 'Computer Games' assignment for the project course EWI3615TU. It is part of the minor "Computer Science" and it is held during its 2nd quarter.

This document has been divided in two main parts: in *Assignment* we describe everything directly related to the software product - a game - that you will be developing; in *Process*, we describe the path you are expected to take and some tools you will use to achieve that result (the 2nd part will be shared shortly before the project kick-off).

Background

The "rise of the indies" is a very hot topic nowadays. Online shops, such as Steam or Google Play, are filled with indie games, since popular software packages, such as Unity, allow almost anyone to create a game and export it to the most common gaming platforms.

This also gave rise to many "game jams", in which individuals or groups of developers get together to create a game in a very short time. Often, themes are involved in order to inspire the creative minds of the game developer. Developers simply experiment with game mechanics and find great potential and inspiration in their designs. The best ones are often further developed, beyond the jam. Quite some of the indie games that are popular right now, were first created during a game jam!

Goals

During this project, you will make a complete, ***technically challenging*** and fun game using Unity in groups of 5 students. While doing this, you will...

- ...gain experience in developing software in teams
- ...gain experience in advanced programming
- ...gain experience in writing intelligent algorithms
- ...gain experience in gathering, storing and visualizing player data

In addition, you will also gain experience in: indie game development; (indie) game industry standard software; 3D modeling, texturing and shading; and finally, playtesting with game analytics. In short, you will experience game development and even become an indie game developer with enough experience to create your own indie game.

Prerequisites

Since the main goal of this project is software development, it is required for this project that you understand the basic concepts of, and have some experience with, Object Oriented Programming (OOP). Languages such as Java or C# are common OOP languages, and experience in either of these is a plus.

Finally, it is advantageous if someone in your team has experience with 3D modeling, texturing, or game design. In any case, we suggest you to do this (old) Unity practical assignment, based on "Roll-a-Ball", before the start of the project! For more tutorials check out Unity's website.

ASSIGNMENT

The assignment of this project consists of making a complete, *technically challenging* and fun game using Unity3D, working in a team of 5 students. This software development project is set up like an extended “game jam” to give you the full indie game development experience. The following subsections discuss some crucial parts of the assignment. Read them carefully before starting!

Your Indie Game

Finding your game concept is, of course, one of the crucial parts of the development. Thinking of a game from scratch is not that easy. Therefore, we help you along by the fact that it has to be based on a theme. However, before you start thinking of your game concept, we will first introduce the requirements that your game has to fulfil.

UNITY

Your game has to be made using the Unity3D game engine. You are advised to make use of its built-in components, which include Physics, NavMesh, UI Tools, Particle Systems, Audio Sources, Light Sources, Networking, etc. You are recommended to download and use the latest version.

YOUR OWN ASSETS

This project's main goal is to make a technically challenging game. Therefore, you will not not be graded for imported content/code/assets. Any kind of externally acquired library or code (snippet) has to be reported and sources of any other assets have to be acknowledged as well! We want to stimulate you to be technically creative rather than to put together a bunch of pre-made ingredients. You can import a JSON parser library if necessary, but if you create it yourself: bonus points!

This should not discourage you to e.g. do some elaborate coding using an external library; you won’t receive points for that library, but you may for what you do with it; depending on its complexity, obviously. In addition, you can make use of external tools (e.g. Photoshop/Blender) in order to create your assets.

In any case, just make sure that:

- ...whatever you import, or use to create your content, is totally legal
- ...whatever you import, or use to create your content, is reported and sourced

2.5D

We *advise* you to make a 2.5D game. This means your game would (partially) contain 3D content, but the playfield is actually 2D. Unity has great 2D tools! If, however, you think you can handle a fully 3D game, you are welcome to do so.

PLAY TESTING & ANALYTICS

During development, you are required to have your builds be playtested by players other than your team. In addition, you are required to somehow let your game gather data of your players and have this data be stored and processed (possibly online) automatically. This can be done using free online tools, such as Gameanalytics.com (mobile only), or Unity Analytics (start implementing in time, as it may take several hours before playthrough data reaches your control panel), but creating your own tool would give you more points.

Another great form of playtesting is by observing other players playing your game, without you telling them anything*: just observe them! And m*ake sure they are unfamiliar with your game!

After collecting your game data, you should plot this data in meaningful graphs, draw conclusions from them and figure out how you can improve your game based upon the received feedback.

In your final presentation, you are asked to include these playtesting data graphs, and to report on the conclusions you drew from them.

GAME COMPLEXITY

Please think well about the complexity of your game concept and how much time is required to build it. Each student has to spend a total of 140 hours on this project, which means that e.g. a *simple* Tetris clone would never be acceptable. Also, creating GTA IX or Roller Coaster Tycoon 7 is obviously impossible within the given timespan. In any case, make sure that no one will be twiddling their thumbs half of the time! (And definitely, increase the complexity, might that happen…)

COMPONENTS

Your game has to include a bunch of technically challenging components, which will demonstrate your knowledge, skills and proficiency. For practical purposes, we will divide these components into four categories: *Computer Graphics*, *Artificial Intelligence*, *Game Analytics*, and *Programming*.

In addition, we will represent the difficulty (or complexity) of a component by a number of stars (up to 5 stars). It is up to you to decide how difficult you will make your game, but remember that technically challenging games will be strongly rewarded. Games that are too simple will not be acceptable.

We require your game to have *at least* the following amount of difficulty stars per category, but more is encouraged:

Computer Graphics (CG)	6 stars
Artificial Intelligence (AI)	6 stars
Game Analytics (GA)	6 stars
Programming (PR)	6 stars

\+ allocate 6 more stars over subjects of your choosing

Thus, a minimum of **30** stars in total!

In the first deliverable, the core project document, you should indicate which components you choose and, for each, how many difficulty stars you think they deserve in *your* game. Remember that your teaching assistant will review this, so he can revise/reject your components and/or assignment of difficulty stars. Therefore, *you should indicate per component what you plan to use it for, who is responsible for it, and why you think it is as difficult/easy as you indicated*. This core project document has to be approved before you can continue working on your game!

The actual amount of stars you assign to each component will vary based upon the complexity of what you want to achieve.

Below we present a possible list of typical components per category, along with (just!) an ***indication*** of their difficulty (range). You are allowed to use these, but it is **required** to think of a few of your own components as well.

- Computer Graphics (CG)
  - 3D Models:
    - Procedurally generated[1] 3D meshes ★★[★]
    - Animated procedural 3D meshes ★★★[★]
    - 3D models ★[★★]
    - 3D animated models ★★[★]
    - Optimizing 3D models for games (e.g. high to low-poly conversion) ★
    - ...
  - Textures:
    - Procedurally generated[1] textures ★★[★★]
    - Texture as input (e.g. for level/algorithm) ★★

[1] By procedurally generated, we mean that something should be generated *in-game.* Thus, not by an external tool, and then imported into Unity. So every time the game starts, procedurally generated content will look different. You will have to write your own script that does the generation (e.g. by means of functions that generate random numbers, available in Unity or C#).

        - Animated Textures (e.g. fire, water, magic, TV screens) ★
        - Custom animated textures ★[★]
        - Baking normal maps from highpoly to lowpoly 3D models ★
        - ...
    - Special effects & Juiciness
        - Animations with eases: [In/Out][Cubic/Quad/Circular/Bounced] ★
        - Audio mixer effects (e.g. in the pause menu, music sounds muffled) ★
        - Camera shakes (e.g. with explosions, button clicks, hits) ★
        - Unsteady camera (e.g. to simulate looking through someone's eyes) ★
        - Particle Systems (e.g. explosions, dust particles, fire, smoke, magic) ★
        - ...
    - Rendering
        - Play with lights and shadows ★
        - Custom shader ★★[★★]
        - ...
    - User Interface (with new Unity3D UI tools)
        - Main menu screen ½★[★]
        - End/results/score screen ½★[★]
        - Pause ½★
        - High scores ★
        - Options ★
        - Credits ¼★
        - UI animations ★
        - ...
- Artificial Intelligence (AI)
    - Dumb enemy ★
    - Smart enemy ★★[★]
    - Huge amount of differently dumb enemies ★★★
    - Pathfinding using own algorithm ★[★★]
    - Some "consciousness" in enemies or the level ★[★★★]
    - Use genetic algorithms ★★★★
    - Use a neural network ★★★★
    - Enemies that learn ★★[★]
    - Smart enemies you always lose from ★★[★]
    - ...
- (Web-Based) Game Analytics (GA)[2]
    - Create your own remote server for storing data ★★★
    - Save relevant information from your game during play ★[★]
    - Collect and show highscores (remotely) ★[★]
    - Create gamer accounts (with avatars) ★[★★]
    - Create your own data analysis tools ★★★
    - Save and share game states with others through social media ★[★]
    - ...
- Programming (PR)
    - Game Mechanics
        - Procedurally generated levels/weapons ★★[★]
        - Dynamic difficulty based on player skill ★★
        - Moving platforms ★
        - Race against the clock ★
        - Local multiplayer ★
        - Split-screen local multiplayer ★★
        - Online multiplayer ★★★[★★]

[2] Game analytics can be done either locally or remotely, but a remote solution will reward more stars.

        - …
    - Mobile
        - Implement mobile controls (e.g. accelerometer) ★[★★]
        - ...
    - Game loop
        - FPS independent (use Time.deltaTime / Time.fixedDeltaTime) ★
        - Game speed can be changed by player (e.g. fast forward) ★[★]
        - Player can go back in time (like an “undo”) ★★★★
        - ...
    - Physics
        - Use Unity’s triggers to trigger certain actions ★
        - Use Unity’s full physics simulation for all movement, collisions etc ★★
        - …

Remember, that, as mentioned previously, difficulty boosts your grade! You have 140 hours, make sure you minimize your boredom. Keep it challenging!

Themes

Your super-fun game will have to be based on a theme. The use of themes both gives you the experience of a game jam, and helps the flow of ideas. You may interpret these themes in any way you like. Be creative! You will probably already have a few game ideas popping up while reading them. Don’t stop there. Brainstorm together, have fun with the themes and the ideas that come to you. Throw them in the group. Even poor/ridiculous ideas can lead to great games. Just keep in mind the previously discussed constraints. So for this year, you can choose from one of the following themes:

- *Not the only way to win*
- *Different yet familiar*
- *A powerful vehicle*
- *Water is cool*
- *A very versatile tool*

In order to illustrate what we mean with ‘*interpret the themes any way you like*’, here are a few examples to give you a quick start:

- “*Reaching the finish line first is not the only way to win*”
- "*This room is different yet quite familiar, have I been here before?*"
- "*This giant mech can walk and kick buildings around*"
- *“Water is pretty cool, it expands when you freeze it”*
- *“This grappling hook allows you to traverse the environment and pull others towards you”*

You are allowed to use one of the above interpretations if you think it would help you to make a great game. But bear in mind that you are also graded for both originality and uniqueness of your game.

PROCESS

Overview

At the beginning of the project, you will brainstorm together with your group about your game concept, and the assignment of team roles. Once this has been approved by your TA, you will start working on prototyping your game in order to get a feel of whether what you are creating is indeed what you hoped for. The deadlines can be found in the 'Detailed Schedule' below.

According to your findings with your prototypes, you will start working on the **Early Access** version. This build should be playable and contain **only the core gameplay features**. In addition it should already include game analytics code, which you will use after letting people playtest your game.

Next up is completing all the game features, and improving your game based on the feedback from the gathered game analytics data: this is the **Beta** version of the game, which may still contain bugs.

In the weeks after Beta, we will have a "feature lock", meaning that no more features may be added to the game: you will only finalize and *polish* the game. This is called the final or **Release** version of your indie game. Finally, your team will give a plenary presentation on both game and process.

Groups & Roles

For this project, each group will consist of 5 students, in which each member will fulfill one or more roles. It's the whole team's task to assign these roles and you should stick with them during the entire project. Team roles define your responsibility, they are meant to give you focus, and to give the team structure and assurance, not to forbid you to do other tasks. So for example, the lead artist might also be assigned to program some component, or help with the game design. However, the tasks related to your responsibility should be a priority. Might some role assignments turn out to work really badly, you can discuss a role switch with your teaching assistant. The roles you will assign include, at least:

GAME DESIGNER

Oversees all game design aspects: game mechanics, game feel, gameplay, and game flow. This person should make sure the game is fun to play, feels right, and looks juicy.

LEAD PROGRAMMER

Oversees programming tasks and code structure/quality of the whole team. Mainly focuses on programming the core components of the game; also creates and maintains the class diagrams.

LEAD ARTIST

Is responsible for the artistic choices that define the game, in terms of visuals, audio and any other content.

WORLD BUILDER

Is responsible for the design of the game levels and puts all their components together so that the game works and there is a level to play.

GAMEPLAY TESTING

Designs how/when/where every week gameplay test sessions will take place, and guarantees that their outcome gets analyzed and is really taken into account in the game.

PRODUCER

Watches over the whole planning, the task assignment and fulfillment, the deliverables, and assures that all communication of/with the group works: notifications, plan changes, deliverables submission, appointments, etc. (incl. communication with the course organization, any external parties, etc…)

In addition to these typical roles and responsibilities, it speaks for itself that everyone will occasionally help any other team members with their assigned tasks.

Scrum

The development of your game will be done following the iterative and incremental software development framework Scrum. Every week you are expected to show a working version of the game, or components of it. We will call each of these a *proof-of-concept*. At each iteration, you will test whether your ideas and implementations work as expected and you will adjust your game concept and schedule accordingly. (In this sense, your game concept will never be really finished…)

Make sure you stay flexible and remove the elements that break the game, and include those that make it strong. This also means that your choice of components may also change. In that case, ***always send an updated version of the Core Project Document to your TA and let it be approved***.

Code Quality

It is important that the code you create is well organized, according to the principles that you have learned during this minor. This means that you should think about the classes and interfaces you create, and you should be able to explain why they interact the way they do.

Apart from the main class structure, the code itself should be readable and well documented, e.g. you should use meaningful names for methods, properties and fields. Also make sure that all methods have small, well-defined jobs; e.g. there should never be one huge method that does everything. The functions of all non-trivial methods should be described using comments in the code. In general, you should make sure that any developer should be able to understand what happens in your program. Please try to ensure the whole team uses consistent code formatting throughout the project.

Project Management

During this project it is ***required*** to use Git (Gitlab) as the main project management tool. We will be able to supervise all groups easily and you will have a wide variety of tools to optimally manage your project. You are also ***required*** to use the included issue tracking system, which is a very useful method to keep track of your todo's and their status. Create issues for each bug, component/feature that you want to see implemented. Assign the right person, and according to the scrum development framework, add milestones to either "next meeting", or "undefined". When somebody finishes a task, be sure to update it and close the issue. Perhaps add a screenshot. This way, everyone can follow the progress in your project.

The Gitlab repositories that you are going to use are hosted by the TU Delft. This means that we will create the repositories and invite you after the groups have been created. In the meantime, you can look at these links to get you started with git:

- Getting started with SourceTree, Git and git flow
- How to use Git for Unity3D source control?
- GitHub client *
- SourceTree (a nice alternative Git client) *

  * You only need one of these!

Teaching assistant

Each group will have a teaching assistant (TA) who will help and guide you throughout the project. Your TA will be available on Mattermost. Similar to Gitlab, Mattermost is hosted by the TU Delft, and you will receive an invite link. You can also contact your TA by mail:

- Shaad Alaka: s.alaka@student.tudelft.nl
- Max Lopes Cunha: m.a.lopescunha@student.tudelft.nl
- Dixit Sabharwal: d.sabharwal@student.tudelft.nl
- Tijmen van Graft: t.r.d.vangraft@student.tudelft.nl
- Shashank Anand: s.anand-1@student.tudelft.nl

Detailed Schedule

This project is worth 5 ECTS, which is quite a deal; so keep in mind that **each member** of your team is required to spend 5 * 28 = 140 hours on this project, over the coming 10 weeks.

Week (start day)	**Tasks**	**Deliverables**
1 (9 Nov)	*9 Nov:* Read the assignment carefully. Think of base game design *10 Nov:* Kick-off meeting and Q&A with TA *11 Nov*: Define core prototypes and the goals of each of them *Then:* Start prototyping and testing components	*13 Nov*: ***Core Project Document***
2 (16 Nov)	Prototyping and testing components	
3 (23 Nov)	*24 Nov:* Prototyping and testing components Elaborate on small prototypes, and adjust Core Project Document according to conclusions	*24 Nov:* ***Demo: Prototypes*** *24 Nov:* ***Revised Core Project Document*** *27 Nov:* ***Game Design Document***
4 (30 Nov)	Start putting together your game! You may use your prototypes	
5 (7 Dec)	Build your indie game	***Peer reviews***
6 (14 Dec)	Build your indie game. Elaborate on tests with your game, and adjust your documents accordingly	*18 Dec:* ***Early Access game***
	Merry Christmas and Happy new year!	
7 (4 Jan)	Build your indie game	
8 (11 Jan)	Build your indie game. Adjust the game and documents according to the outcome of your game analytics	*15 Jan:* ***Beta game***
9 (18 Jan)	Final bug fixes and tweaks. Prepare Game Release!	***Peer reviews***
10 (25 Jan)	Prepare your final presentation	*26 Jan:* ***Release your indie game*** *28/29 Jan:* ***Final Presentations***

ALL DEADLINES ARE STRICT, ENDING AT 23.59

Communication with course staff

For this course, a Mattermost group has been created. You will receive an invite link after the groups have been created. Please use your netid for username and **fill in your full name**. Mattermost will be the main channel for communication. The TAs will be monitoring the Mattermost channels and Rafa Bidarra is present as well. The **#General** channel is used for course-wide announcements. Questions

can be asked in **#Questions**, grade-related and team-specific stuff can be asked in your own team channel or in a private message to your TA.

Grading

Your grade will be determined based on the following points:

- Product (55%)
- Process (35%)
- Presentation (10%)

Deliverables

There are three types of deliverables:

1. **Project files** (source code, 3D models, art)
   These should be added to a Gitlab repository, see chapter on project management.
2. **Binaries** (game releases)
   These must be uploaded to your Gitlab "Tag" page.
3. **Documents and reports**
   Reports and other documents must be published to your Gitlab repository, which can be done in the following ways:
   1. Create a 'Deliverables' folder in the root of your repository, and upload there your reports.
   2. Create a 'Tag' in your Gitlab repository and add the document as an attachment.

   Please organise these files properly by using the naming convention:
   Group#_'REPORT_NAME'.pdf, where 'REPORT_NAME' is equal to the deliverable mentioned in this chapter, e.g. "Group1_Core_Project_Document" or "Group2_Game_Design_Document".

   You are free to use your preferred tools for working on the reports, but for collaborative work in LaTeX we recommend the use of Overleaf.

CORE PROJECT DOCUMENT – 13th NOVEMBER 2020, 23:59

In this special game pitch document you will specify:

1. Group name and number
2. Theme and interpretation
3. Game idea in about 100 words
4. Student names, emails and role assignment
5. Which features you are thinking to be implementing in your game

This document has to give a clear idea on what game you are going to be creating. The next two weeks you will be working on creating prototypes to see which features are viable or not. The *revised* core project document will contain the 'final' list of features you will work on and how difficult the task is (1-5 star).

REVISED CORE PROJECT DOCUMENT – 24th NOVEMBER 2020, 23:59

1. The key components to be implemented in your game, and for each indicate:
   - the difficulty stars you expect it to take (1 to 5)
   - *detailed* description of the component, including e.g. how many? what for? why 4 stars?
2. A rough schedule/timeline for the development (complying to the schedule on page 8)

In addition, for overview purposes, please summarize the *total* amount of stars you allocated, and also the *subtotal* amount of stars per category (see pages 3-4).

For new versions, please indicate all changes since the previous version in a red color.

Whenever something technical in your project is changed/removed/added, please update the document accordingly and notify your TA for approval.

PROTOTYPE DISCUSSION WITH TA – 24TH NOVEMBER 2020

You have made one or several proof of concepts for your game in the first weeks, each of which consists of one or more prototypes. These prototypes are the components in your game that you wanted to try out first in a very basic form, and figure out whether:

- Do they work as expected?
- Does it take more time to develop than expected?
- Are you satisfied with the prototype?
- Will you use it in your final game?
- Does it need improvement, and why?

Prototypes can be e.g. preliminary AI design, interfaces, levels or level generators, models and textures. Anything can be a prototype!

In this meeting, you will discuss conclusions and progress on all the prototypes that you have created. Explain what worked, what didn't, what you changed to improve the component during development and why, and if you are still going to use it and in what form. So, note that the prototypes you have developed don't necessarily need to be included in your final game. If a prototype does not work or is too hard to work with, drop it! Cover the list of questions mentioned above and draw conclusions based on your findings!

GAME DESIGN DOCUMENT – 27TH NOVEMBER 2020, 23:59

In this document you give a detailed description of what will be included in your final game, but this time the focus is less technical. It focuses on the game design aspects. It is the player side of the game that should be covered (the **WHAT**?), not the developer's side (the **HOW**?); e.g. What are the game mechanics?, What will engage the players to play the game? If it helps to make your document more clear, include sketches or images that illustrate your game design.

Additionally, give an overview to describe the role of each game component, and how they all fit together in the game. For example, for a set of models, detail why they are created in that way, how do they fit in the game, etc; for an algorithm, indicate why do you need a pathfinding algorithm? Etc.

The following sections should be included in your game design document (if applicable):

- Introduction
- Target Audience
- Platform & Controls
- Story, characters and setting of the game
- Artificial Intelligence
- Level/environment design
    - How will the levels look? What is the feel of the levels?
    - Are there multiple levels?
- Gameplay and mechanics
- Art
    - Style, include a collage of images to paint a picture of it.
    - Include a list of all models, textures, sprites, etc. that will be in the game
    - How do they fit into the setting?
- Sound and Music
- User Interface, Game Controls

The game design document should be no longer than 2000 words.

Every game component should be described in enough detail for the respective developers to implement them. You can take inspiration from other Game Design Documents: just search for them, either online or in the Resources folder at Brightspace.

Also include in the document the priorities of the things that need to be implemented or made. Think of MoSCoW! This means:

- What **M**ust be done, is crucial for your game to function, and thus has the highest priority?
- What **S**hould be done, but maybe is not too bad if it is not implemented in the end?
- What **C**ould be done: what could be nice to have, if you have the time at the end?
- What **W**on't be done? What features are too hard to implement, and therefore won't be included? For example, what prototypes did not work at all or were too difficult?

The game design document is a living document. This means if you change your mind about what needs to be included in your game, or if elements in your game change, update the descriptions in your game design document. This way your team is always up to date about the plans for the final game. **Please also notify your TA if you update your document later on.**

EARLY ACCESS GAME – 18TH DECEMBER 2020, 23:59

*The game should be playable with respect to its core gameplay and contain **finished** game analytics code*! The game will contain all other features although possibly still in an unpolished state.

During the Christmas holidays you are encouraged to let the game be playtested by other people, e.g. family, friends, acquaintances. Because, immediately after the holidays, you will have to start analyzing the received data and feedback, and adjust your game according to your findings. Your TA can possibly ask you about your findings during these playtests. Please hand in your early access game using the Gitlab tag system. And send the staff, through your Mattermost channel, a link to the said page.

BETA RELEASE – 15TH JANUARY 2021, 23:59

The goal here is to have a game that contains ALL features with only 'minor' bugs remaining. Where 'minor' in this case, means bugs that are easy to fix before the release. This means that from this point on there is a *feature lock*![3] Please hand in the beta release using the Gitlab tag system. Send the staff, through your Mattermost channel, a link to the said release page.

FINAL RELEASE – 26TH OF JANUARY 2021, 23:59

This is the fully-polished FINAL version! All game-breaking bugs should be gone. Of course, no software is bug free, but with software development the total amount (and impact) of issues should be minimized. Your game will be graded on many aspects, such as: 'technically challenging' (most important), 'bug-free', 'completeness', 'gameplay', and 'juiciness'.

If you decide to implement remote game analytics, the final version of your game is required to run both OFFLINE and ONLINE. We expect the servers to only run for a limited amount of time (either self hosted or server provided by us), but you may want to keep them up for longer.

Please hand in the final release using the Gitlab tag system. Send the staff, through your Mattermost channel, a link to the said release page (*and to the main page on your server, if you created one)*

Final Presentations

Each presentation will take no longer than 15 minutes and will consist of 3 parts. You are expected to:

1. give a presentation where you introduce the game, and talk about what technically challenging components your game contains and how you approached them (5 minutes)
2. let a student from another group play your game under your guidance (5 minutes)

[3] With a feature lock we want to prevent you from adding a lot of new gameplay mechanics that can possibly break your game. After the lock, we focus our efforts on polishing the existing mechanics.

3. defend your product by answering questions (5 minutes).

This means each group needs to fulfill the following roles: 1 presenter, 1 guide, 1 defender, 1 player. You are free to assign those roles in any way you like. If you would like multiple people to present or defend that is fine, but keep in mind that 5 presenters may not be effective given the time.

The slides used in your presentation will also have to be made available at the repo, like all other documents and reports.

Keep in mind that the order of the presentations will be random, so make sure you are there from the start! Attending the final presentations is mandatory.

ATTACHMENT 1

Links

On the Ludum Dare Compo tools page, you will find an awesome list of free tools!
On CGTextures you will find a huge amount of freely available textures.
On Freesound you will find a large collection of free to use audio files.
On Gameanalytics.com you find great free game analytics tools for your game.
There is an official unity tutorial page. For instance UI tutorials!

You are free to use resources from any (legal) source, but remember to always credit the rightful creator of the resource you are using.

Tips & Tricks

PLAYTESTING

Put 2 people in front of your game at once, they will start discussing and narrating what they see to each other, which is a great way for you to figure out how they experience your game. Players playing alone tend to clamp up so you are unsure what goes on in their mind.

UNITY AND GIT

In order to properly work together you will use Git, however it won't always work out-of-the-box. There are many files that you do not want included in your repository. To exclude these files and folders, you have to add them to your .gitignore file. A template for Unity can be seen below:

You *should not* ignore *.meta files, as they are used for references within the scene.

```
[Ll]ibrary/
[Tt]emp/
[Oo]bj/
[Bb]uild/
[Bb]uilds/
Assets/AssetStoreTools*

# Visual Studio cache directory
.vs/

# Autogenerated VS/MD/Consulo solution and project files
ExportedObj/
.consulo/
*.csproj
*.unityproj
*.sln
*.suo
*.tmp
```

```
*.user
*.userprefs
*.pidb
*.booproj
*.svd
*.pdb
*.opendb

# Unity3D generated meta files
*.pidb.meta
*.pdb.meta

# Unity3D Generated File On Crash Reports
sysinfo.txt

# Builds
*.apk
*.unitypackage
```

Finally, working together in the same scene is a nuisance. The only way this can work with the least amount of issues is by setting asset serialization to "Force Text". Conflicts might still arise however, so you manually have to merge or solve those in an editor, such as the one provided by TortoiseGit or SourceTree. However, we'd recommend using prefabs as much as possible, as they will only form a short reference in the scene file, and become a file on their own in your asset folder (as long as you apply all the changes you make to the prefab. All the bold properties within your prefab are not stored in the prefab itself, but the scene file. In which case conflicts might still arise)

Important: Another way to avoid conflicts is by separating parts of your game in different scenes. Make sure that everyone creates their prototype in a new scene, possibly creating a unique folder for scripts to avoid file and class naming collisions.

HELP

If you want to check the documentation of Unity, you can do so by clicking the little help icons in the Unity interface (the icon looks like a blue book with a question mark). Or, you can just check the online manual and scripting API. Finally, if you can't find an answer to your question, you can ask your TA for some guidance. We encourage you to try and find a solution with your own team first, but do not linger on a problem for too long. The time we have is limited and is better spent improving the game than reading 4 year old documentation and blog posts.

JUICINESS

Making your game "juicy" is a great way to improve the entertainment value of your game. It might not fix a broken game, but it greatly improves the game feel and appeal, which basically means how smooth and pleasing your game is to play. Snappy or slow movements often block the fun (unless such crappiness is part of your gameplay of course). The prettier your game, the more aesthetically pleasing and attractive it is to play.

We suggest you to take a look at this video and/or at this lengthier talk by Jan Willem Nijman, game designer at Vlambeer, which will also help you spice up your game using simple tricks!

MANAGERS, CONTROLLERS

In Unity, it is often advised to make manager classes that centralize the management or control of certain aspects of your game. More information can be found here.

TIME MANAGEMENT

Whenever you think that implementing a new feature takes an x amount of time, assume it actually will be at least $2x$ to $3x$ as much time.