EWI3620TU - Minor project Software Ontwerpen en Toepassen 2015/2016 [Q2]

YOUR INDIE GAME

VERSION 4.2 (11/11/2015) - changes are indicated in red!

INTRODUCTION

This document describes the assignment for the minor project EWI3620TU. It is part of the minor "Software Ontwerpen en Toepassen" and is held during the 2nd quarter of this study year.

This document has been divided in two main parts: in *Assignment* we describe everything directly related to the software product 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.

Background

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

This also gave rise to many "game jams", where individuals or groups of developers get 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 firstly 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. Thus, you will...

- ...gain experience in software development
- ...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 usage data

In addition, you will also gain experience in: indie game development; (indie) 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 pure OOP languages and experience in either of these is a great plus. Note that Java is not the same as Javascript!

Finally, it is recommended that you have experience with 3D modeling, texturing, game design, and/or the use of the game engine Unity3D. You have probably already gained most of this experience during the course *EWI3610TU Computer Graphics*. If for some reason this is not the case, we urge you to catch up and *at least* do the Unity practical assignment "Roll-a-Ball".

ASSIGNMENT

The assignment of this project is to make a complete, *technically challenging* and fun game using Unity3D, in groups of about 5 students. This software development project is set up like an extended "game jam" for the full indie game development experience. The following subsections each discuss some crucial part of the assignment. Please read it all carefully before starting!

Your Indie Game

Finding your game concept is of course one of the most 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, see the next subsection. However, before you start thinking of your game concept, we will first discuss the requirements that your game has to adhere to.

UNITY

Your game has to be made using the Unity3D game engine. You are advised, to make use of the built-in components. This includes Physics, NavMesh, UI Tools, Particle Systems, Audio Sources, Light Sources, networking, etc. You are recommended to use the latest version. Download it here.

YOUR OWN ASSETS

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

This does not mean that if you want to do some elaborate coding using an external library to aid you, you won't receive any points. You simply won't receive it for that library, but you may for what you do with it. Depending on the complexity, obviously.

In addition, you can make use of external tools in order to create your assets.

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 users other than yourself. In addition, it is required to somehow let the game gather data of your players and have this data be gathered online automatically. This can be done using free online tools, such as Gameanalytics.com, but creating your own tool would give you more points.

Another great form of playtesting is observing other players play your game, without you telling them anything. *Just observe.* Make sure they are unfamiliar with your game!

After receiving your 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.

At about 3/4th of the project, you will write a test report where you are asked to include these graphs and report your conclusions.

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 280 hours on this project, which means that e.g. a *simple* Tetris clone would never be acceptable. Also, creating GTA VI or Roller Coaster Tycoon 4 is obviously impossible within the given timespan. Just make sure that no one will be picking their noses half the time! If this happens, increase the complexity!

COMPONENTS

Your game has to include bunch of technically challenging components, in order to prove your knowledge and skill. A list of possible components is shown below. You are allowed to use these, but it is **required** to think of a few of your own components as well.

In the first deliverable, the core project document, you should indicate which components you use, and for each decide how many difficulty stars you think they deserve in *your* game. Remember that your teaching assistant will review this, so he can reject your components 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*. The first deliverable has to be approved before you can continue working on your game!

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 be rejected.

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

Computer Graphics (CG)	10 stars
Artificial Intelligence (AI)	8 stars
Web & Databases (WD)	6 stars
Programming (PR)	10 stars

\+ allocate 6 stars over subjects of your choosing

Thus, **40** stars in total!

The actual amount of stars you assign each component will vary based upon the complexity of what you want to achieve. The list of component ideas, along with an *indication* of their difficulty:

- Computer Graphics (CG)
  - 3D Models:
    - Procedural generated* 3D meshes ★★[★]
    - Animated procedural 3D meshes ★★★[★]
    - 3D models ★[★★]
    - 3D animated models ★★[★]
    - Optimizing 3D models for games (e.g. high to lowpoly conversion) ★
    - ...
  - Textures:
    - Procedurally generated* textures ★★[★★]
    - Texture as input (e.g. for level/algorithm) ★★
    - 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 (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)
        - Start, pause, end screen ★
        - High scores ★
        - Options ★
        - Credits ★
        - 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 & Databases (WD), Game Analysis
    - Send playthrough data to a free online tool ★
    - Your own game analysis tool on your webserver ★★★[★]
    - Collect and show highscores from web server ★★
    - Online gamer accounts with avatars ★★★
    - 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 ★★★★[★]
        - …
    - 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 280 hours, make sure you minimize your boredom. Keep it challenging!

* 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 generating, by means of functions that generate random numbers, available in Unity, or C#.

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 bad/ridiculous ideas can lead to great ones. Just keep in mind the previously discussed restrictions. Choose one of the following themes:

- You only get one
- Build the level you play
- An unconventional weapon
- 10 seconds
- Things you hate

In order to illustrate what we mean with "interpret the themes any way you like", we provide a few examples to give you a quick start:

> "*You only get one* bullet" – "*You only get one* glass of goat milk" – "*10 seconds* before he reaches his gun" – "goats as *unconventional weapon*" – "the player's mistakes are the *unconventional weapon*" – "*Things you hate*: comic sans"

You are allowed to use one of the above interpretations if you think it would allow 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 the concept of your game, and the assignment of roles. Once this is 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.

According to your findings of your prototypes, you will start working on the “Early Access” version. This build should be playable and contain the core gameplay features. In addition it should include game analytics code, as you will let people playtest your game.

Next up is finishing all the features of your game, and improving your game based on the feedback from the gathered game analytics data. You will release a beta version of the game, which still may contain bugs.

From this point on you will have a “feature lock”, so no features should be added to the game, you will only *polish* the game. You will then release the final version of your indie game and give a presentation to finalize the project! :)

Groups & Roles

For this project, you will be assigned to a group. Each group will consist of 5 students, where each will fulfill a role. These roles are for you to assign, and we’d like you to stick with them during the entire project. They simply define your responsibility, but don’t necessarily limit you with the tasks you can perform. A lead artist can also be assigned to program some component, or help with the game design. However, the tasks related to your responsibility should be a priority. Also, do not worry! If some role assignments turn out to work badly, you can discuss a change of roles with your teaching assistants. The roles you can assign are:

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, and feels juicy.

LEAD PROGRAMMER

Oversees programming tasks and code structure/quality of the whole team. Mainly focusses on programming the core components of the game. Also creates and manages the class diagrams.

LEAD ARTIST

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

WORLD BUILDER

Designs the levels and puts all components together so that the game works and there is a level to play.

PRODUCER

Watches over the planning, task assignment and fulfillment, the organization of gameplay test sessions, the deliverables, and takes care of all communication with the staff.

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

Scrum

The development of your game will be done by the iterative and incremental software development framework Scrum. Every few weeks you are expected to hand in a working version of the game, or components of it. We will call these proof of concepts. At each iteration you will test whether your ideas

and implementations work as expected and you will adjust your game concept and schedule accordingly. Therefore, 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. This also means that your choice of components may also change. If this happens, ***send an updated version of the Core Project Document to your student assistant 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. This means that 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 method 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.

Project Management

During this project it is *required* to use GitHub as the main project management software. 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 **required** to use the included issue tracking system, which is a very useful method to keep track of your todo's and their statuses. Create issues for each small and large 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.

So create your project page on GitHub, and start managing! We will need the link to your project page for the first deliverable. Here are a few links to get you started:

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

  * You need just one!

Obviously, if you don't have an account at GitHub yet, please create one as soon as possible. If your project is created, please add the following GitHub account to your repository:

EWI3620TU

Teaching assistants

Each group will have a teaching assistant (TA) who will help and guide you throughout the project. TA contact info for this year:

- Olivier Hokke, o.j.hokke@student.tudelft.nl
- Bas Dado, b.dado@student.tudelft.nl

Detailed Schedule

You will get 10 ECTS for this project, so keep in mind that you are required to spend 10 * 28 = 280 hours on this project.

Week 1 (9 nov – 15 nov)	*9 nov:* Read the assignment carefully. Think of base game design. *10 nov*: Define core prototypes and the goals to learn from them. ***Deliverable: Core Project Document*** *Then:* Start prototyping and testing components.
Week 2 (16 nov – 22 nov)	Prototyping and testing components.
Week 3 (23 nov – 29 nov)	*24 nov:* Prototyping and testing components. Elaborate on small prototypes with a Prototyping Report, and adjust Core Project Document according to conclusions. ***Deliverables: Prototypes, Prototyping Report, Revised Core Project Document*** *27 nov:* ***Deliverables: Game Design Document***
Week 4 (30 nov – 6 dec)	Start making your indie game! You may use your prototypes.
Week 5 (7 dec – 13 dec)	Build your indie game. ***Deliverables: peer reviews***
Week 6 (14 dec – 20 dec)	Build your indie game. Elaborate on tests with your game, and adjust your documents accordingly. *18 dec:* ***Deliverable: early access game***
	Happy new year!
Week 7 (04 jan – 10 jan)	Build your indie game. *6 jan*: **Deliverable**: **game analytics report**
Week 8 (11 jan – 17 jan)	Build your indie game. Adjust your game and documents according to the game analytics. *15 jan:* ***Deliverable: beta game***
Week 9 (18 jan – 24 jan)	Final bug fixes and tweaks. Prepare presentation! ***peer reviews*** *TBA:* ***Deliverable: report & your indie game!!! :D*** *TBA:* ***Presentations***

Grading

Your grade will be determined based on the following points:

- Product (40%)
- Process (30%)
- Presentation (15%)
- Final report (15%)

All deliverables should be met.

Deliverables

There are three types of deliverables:

1. **Project files** (source code, 3D models, art)
   These should be added to a github repository, see chapter on project management.
2. **Binaries** (game releases)
   These must be uploaded to your github "Releases" page.
3. **Documents and reports**
   You are required to create a Google Drive folder in which you store and edit your reports. This allows you to work together on the same reports, at the same time seamlessly. In addition, you and the TA's can place comments directly in the document. It also allows the TA's to check the history and make sure that a finished document was available at the mentioned deadline.

   The names of these documents should be equal to the associated deliverable mentioned in this chapter, e.g. "Core Project Document" or "Game Design Document".

   Your Google Drive ***folder*** should be shared with the TAs:

   - ➔ immortaly007@gmail.com
   - ➔ ojhokke@gmail.com

CORE PROJECT DOCUMENT – 10TH OF NOVEMBER 2015, 23:59

As a group, provide a special game pitch document where you specify:

1. Group name and number
2. Theme and interpretation
3. Game idea in about a 100 words
4. The key components to be implemented in your game, and for each indicate:
   - *o* The difficulty stars you assign to it (1 to 5)
   - *o* *Detailed* description of the component. E.g., how many? what for? why 4 stars?
   - *o* The name of the student responsible for this component
5. Student names, e-mails and role assignment
6. A rough schedule/timeline
7. A link to the GitHub project page

In addition, for overview purposes, please summarize the *total* amount of stars you allocated, and also the *subtotal* amount of stars per subject.

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

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

PROTOTYPING REPORT – 24TH OF NOVEMBER 2015, 23:59

You have made one or several proof of concepts for your game in the first weeks, which each consist 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 preliminary AI design, interfaces, levels or level generators, models and textures. Anything can be a prototype!

The report will contain details, conclusions and progress on all the prototypes that you have created. For each, write what worked out, 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!

Note: the prototype report does not need to be lengthy.

GAME DESIGN DOCUMENT – 27TH OF NOVEMBER 2015, 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, not the developer's side. How will the players be engaged to play the game? What are the game mechanics?. If it helps to make your document more clear, include sketches or images that illustrate your game design.

Additionally, give an overview to describe how all pieces of the puzzle 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. The same goes for algorithms. 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.
  - ◆ Make 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 1500-2500 words.

Every game part should be described in enough detail for the respective developers to implement them. You can take inspiration from other Game Design Documents, available online. Just do a Google search, you will find many: http://goo.gl/4GTI0t

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; what is crucial for your game to function, and thus has 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 the TA's if you update your document later on.**

EARLY ACCESS GAME – 18TH OF DECEMBER 2015, 23:59

*The game should be playable with respect to its core gameplay and contain **finished** game analytics code*! The game will contains all other features but most in still an unfinished state.

During the holidays you are required to let the game be playtested by other people; for instance 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. Note that the game analytics report deadline is already the second day of the first week!

Please hand in your early access game using the GitHub release system. Send your TA, tudelft@orbitgames.nl, and r.bidarra@tudelft.nl a link to the said release page.

GAME ANALYTICS REPORT – 6TH OF JANUARY 2016, 23:59

More details will follow soon, but it should contain meaningful graphs from which you draw conclusions that hopefully will improve your game, and/or confirm its awesomeness. Also, reports on playtesting observations should be included. Example questions you might want to answer in this report:

- Where did players fail to understand the goal?
- Where did players miss necessary feedback to continue reaching their goal?
- Did players reach their goal? If not, why?
- What are the players' motivations for playing your game? Is it what you expected?
- Where did the game break?
- What bugs did occur?
- Could players easily find their way through the UI/game?
- Did players find the controls intuitive?
- ...

BETA RELEASE – 15TH OF JANUARY 2016, 23:59

The goal here, is to have a game that contains ALL features with simply 'minor' bugs. 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"! Please hand it in using the GitHub release system. Send your TA, tudelft@orbitgames.nl, and r.bidarra@tudelft.nl a link to the said release page.

FINAL RELEASE – TBA

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'.

The final version of your game is required to run both OFFLINE and ONLINE. If for some reason you think this will be difficult and deserves extra stars for your hard work, please indicate clearly WHY, and attach a fixed CPD accordingly. The simple reason for this requirement is that your servers will probably go offline after some time, but you obviously don't want your game to crash whenever it is launched in 2 years.

Please hand it in using the GitHub release system. Send your TA, tudelft@orbitgames.nl, and r.bidarra@tudelft.nl a link to the said release page.

In order for us to easily check if your server is working properly, we ask you to include a link in the email that refers to a main page on your server.

FINAL REPORT – TBA

You are expected to hand in a well formed report, one that contains all required components, has a clear no-nonsense layout, is to the point, and is approximately mistake free with regards to spelling and grammar. You may decide whether to write it in Dutch or English.

The following sections and elements should be included:

- Cover with title , using a screenshot or a piece of art from the game
- Team picture (or collage) with names indicated
- Names, study and faculty, emails, student numbers
- Introduction
- Target Audience
- Platform & Controls
- Story and setting of the game
- Technical components, briefly explain:
    - Each component and who worked on it
    - How it was solved
    - What (major) unexpected problems were tackled or disregarded along the way (if any)
- Code quality
    - How did you make your code readable and clean?
    - Did you work together with each other on the same code? If so, did it work?
- Art (Design & style, both in-game and UI)
    - What did you make, and who made it
    - What was downloaded/generated externally
    - Screenshots
- Process
    - How did you work with SCRUM? Did it work well or did it take more time than it saved? If, for some reason, you did not use SCRUM, how did you track todo's & tasks?
    - How did it go with the roles within your team?
- Conclusion
    - Is product as expected?
    - What went well?
    - What would you do better next time?

The report should be 1000-2500 words.

Presentations

Each presentation will be 15 minutes and will consist of 3 parts. You are expected to:

a. Give a presentation where you introduce the game, and talk about what technically challenging components your game contains and how you solved them (5 minutes),
b. Let a student from another group play your game under your guidance (5 minutes),
c. 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.

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

The exact date, time and location of the presentation are yet to be determined.

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 Gameanalytics.com you find great free game analytics tools for your game
There is an official unity tutorial page. For instance UI tutorials!

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 GITHUB

In order to properly work together you will use GitHub, however it won't always work out-of-the-box. There are 2 options for a proper setup of your GitHub repository.

1) When creating your repository, initialize your .gitignore with the Unity preset. See image.
2) Or, if you did not do this step, ensure you ignore Unity's temp, Library, and obj folders, as those will be generated and will conflict when you push them to GitHub. You don't need to touch those folders, so ignoring them is fine. In addition, you should ignore Visual Studio project files: *.csproj, *.sln, *.suo. You *should not* ignore *.meta files, as they are used for references within the scene.

   However, there are still a few other things that you might want to ignore. The full GitHub preset for Unity is:

```
[Ll]ibrary/
[Tt]emp/
[Oo]bj/
[Bb]uild/

# Autogenerated VS/MD solution and project files
*.csproj
*.unityproj
*.sln
*.suo
*.tmp
*.user
*.userprefs
*.pidb
*.booproj

# Unity3D generated meta files
*.pidb.meta

# Unity3D Generated File On Crash Reports
sysinfo.txt
```

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)

Another way to avoid conflicts is by separating parts of your game in different scenes. You could put the main menu in a different scene than you game, and even your game scene could *add* different parts of the game using separate scenes, in an additive manner using *Application.LoadLevelAdditive*.

If you want objects to stay alive while switching scenes you can call the *DontDestroyOnLoad* on that object. Additionally, data will stay alive when used as static variables.

THE NEW UNITY UI TOOLS

Firstly, the new UI tools of Unity offer really useful and easy to use components, such as buttons, sliders, text fields, input fields, scrollable panels etc. Adding functionality and style to these components is easy. It is not necessary to program your own, unless you need some specific functionality that Unity can't possibly provide. Also, avoid the use of the old "OnGui()" method and prevent the use of 3D planes that display text using textures.

Secondly, some people had problems with the buttons that were being navigated using the arrow buttons on the keyboard, and pressed using the space/enter/escape keys. You can avoid this by opening the properties of the EventSystem game object and setting:

- Horizontal Axis: Mouse X
- Vertical Axis: Mouse Y
- Submit Button: Fire1
- Cancel Button: Cancel

However, keep in mind that you then destroy the possibility for external controllers, like the XBOX 360 controller, to work on your game. The great benefit of using Input.GetAxis("horizontal") is that you automatically support other means of controlling your game.

Finally, you may use the old Unity UI system, but you would be developing a skill that is going to expire. The new easy to use UI tools are the future of interfaces for Unity! It really is simple.

Q: How do I know I am using the **old** UI system?
A: If you have to write code that includes the "OnGUI()" method.

Q: How do I know I am using the **new** UI system?
A: The actual UI elements are displayed in your scene hierarchy. In your scene there is a Canvas object in which UI elements are possibly included. All UI elements have RectTransform components, instead of regular Transform components. However, if you are writing code that generates UI elements, then the above doesn't apply. When your code doesn't use the "OnGUI()" method and you are using UnityEngine.UI package, then you are most certainly using the new system!

Hint: have a look at the "GameObject > UI" dropdown menu, here you can find some useful UI components for you game.

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.

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 advise you to have a look at this video!

Furthermore, there is a 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 controlling of certain aspects of your game. More information can be found here.

TIME MANAGEMENT

Whenever you think something takes an x amount of time, assume it actually will be at least $2x$.