EWI3620TU - Minor project Software Ontwerpen en Toepassen 2014/2015 [Q2]

YOUR INDIE GAME

VERSION 2.1 (26/11/2014) - *Changes are indicated in a dark red color*

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.

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, as 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. Many of the indie games that are popular right now, were firstly created during a game jam! 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.

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 player data

In addition, you will also...

- ...gain experience in indie game development and game jamming
- ...gain experience with 3D game engines
- ...gain experience with the (indie) industry standard software
- ...gain experience in 3D modeling, texturing and shading
- ...gain experience in analyzing and testing your game

In short, you will experience game development and even become an indie game developer with enough experience to create your own video 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".

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.

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 4.6

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. The version you are expected to use is 4.6. Download it here.

YOUR OWN ASSETS

We will *not* allow the use of third party content/code/assets. This includes for instance 3d models, images, and particle systems imported from the Unity3D asset store or any other place. You have enough time to create your own assets!

This project's main goal is to make a technically challenging game. Gameplay and fun comes second! Therefore, you are not allowed to use any kind of externally acquired library or code (snippet). You can of course check them out for inspiration, but you have to implement it all yourself! This is going to challenge you guys, you will learn a lot from it! We stimulate you to be technically creative.

EXCEPTIONS

However, **music and sounds** may be imported. Just make sure that it is totally legal (take a look at the license!). Of course, also here, creating your own content will be a plus Furthermore, you are allowed to use third party software, as long as it is legal, to create your sounds, music, models, art, textures, etc. On the Ludum Dare Compo tools page, you will find an awesome list of free tools!

Furthermore, **textures** are often composed of other images and effects. You *are* allowed to use third party images as textures (as is), or you may use and/or combine them to make your own textures.. Again, as long as it is legal. On CGTextures you will find a huge amount of freely available textures for you to use in your textures or art. Additionally, **normal maps** (or bump maps) are quite difficult to make yourself. These should be baked from high-poly models onto low-poly models, or generated based on textures. So for normal maps you are allowed to use generator/baking tools.

Also, regarding **3D models and animations,** you are allowed to use two to three imported or generated ones, but you will not get any points for this. Unless generating the model was not trivial, and meant that you needed technical knowledge and skill. Discuss this with your student assistent first.

Finally, for the **web & databases** components, you are allowed to use third party frameworks, as long as there is some coding left to do, such as database queries. The main goal of this component is understanding ‘game analysis’ and to apply some of your knowledge regarding databases.

For every use (of course, legal) of textures, music or content from other people, make sure to always mention that explicitly in your final report, clearly indicating the respective source.

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.

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.

Finally, it is required to have *at least* the following amount of difficulty stars per component type, but more is encouraged/advised:

Computer Graphics (CG):	12 stars
Artificial Intelligence (AI):	8 stars
Web & Databases (WD):	8 stars
Programming (PR):	10 stars

The list of component ideas, along with an indication of their difficulty using stars (stars assignment may differ per application):

- Computer Graphics (CG)
    - 3D Models:
        - Procedural generated* meshes ★★★
        - Animated procedural meshes ★★★★
        - 3D models ★ ~ ★★★
        - 3D animated models ★★ ~ ★★★
        - ...
    - Textures:
        - Procedurally generated* textures ★★★★
        - Texture as input (e.g. for level/algorithm) ★★
        - Animated Textures (e.g. fire, water, magic, TV screens) ★★
        - ...
    - Special effects & Juiciness
        - Animations with eases: [In/Out][Cubic/Quad/Circular/Bounced] ★★
        - Sound effects ★
        - Camera shakes (e.g. on 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)
    - Pathfinding using own algorithm ★★★
    - Some "consciousness" in enemies or the level ★★★
    - Use genetic algorithms ★★★★
    - Use a neural network ★★★★
    - Enemies that learn ★★★
    - Enemy you always lose from ★★★
    - ...
- Web & Databases (WD), Game Analysis
    - Collect playthrough data ★★ required!
    - Store data on web server ★★ required!
    - Visualize data on web server ★★
    - 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 ★
        - Player can go back in time (like an "undo") ★★★★
        - Split screen multiplayer ★★
        - ...
    - Game loop
        - FPS independent (use Time.deltaTime) ★★
        - Game speed can be changed by player (e.g. fast forward) ★★
        - ...
    - Physics
        - Use Unity's triggers only for collision checks ★
        - 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. Thus, 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
- Surrealism is the mechanic
- 10 seconds
- Things you hate
- Procedurally generated

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" – "10 seconds after a nuclear explosion" –
"Things you hate: comic sans" – "Everything is procedurally generated"

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

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 you should 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
- Tom Viering, t.j.viering@student.tudelft.nl
- Bas Dado, b.dado@student.tudelft.nl

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 (10 nov – 16 nov)	*12 nov:* Read the assignment carefully. Think of base game design. Define core prototypes and the goals to learn from them. ***Deliverable: Core Project Document*** *Then:* Start prototyping and testing components.
Week 2 (17 nov – 23 nov)	Prototyping and testing components.
Week 3 (24 nov – 30 nov)	*26 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*** *28 nov:* ***Deliverables: Game Design Document***
Week 4 (01 dec – 07 dec)	Start making your indie game! You may use your prototypes.
Week 5 (08 dec – 14 dec)	Build your indie game. ***Deliverables: peer reviews (deadline TBA)***
Week 6 (15 dec – 21 dec)	Build your indie game. Elaborate on tests with your game, and adjust your documents accordingly. ***Deliverables: early access game (deadline TBA)***
	Happy new year!
Week 7 (05 jan – 11 jan)	Build your indie game.
Week 8 (12 jan – 18 jan)	Build your indie game. Elaborate on tests with your game, and adjust your documents accordingly. ***Deliverables: beta game (deadline TBA)***
Week 9 (19 jan – 25 jan)	Final bug fixes and tweaks. Prepare presentation! ***Deliverables: peer reviews, and your indie game!!! :D (deadline and presentation date TBA)***

Grading

Your grade will be determined based on the following points:

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

Deliverables

All deliverables are to be uploaded on the wiki of your GitHub account. Send a copy to your student assistent, and cc the following 2 addresses:

- games@wolfox.nl
- r.bidarra@tudelft.nl

CORE PROJECT DOCUMENT – 12TH OF NOVEMBER 2014, 18:00

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:
   - The difficulty stars you assign to it (1 to 5)
   - *Detailed* description of the component. E.g., how many? what for? why 4 stars?
   - 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

PROTOTYPING REPORT – 26TH OF NOVEMBER 2014, 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 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 – 28TH OF NOVEMBER 2014, 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

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 send a revised copy to the TA's if you update your document.

The game design document should be 1500-2500 words.

ATTACHMENT 1

Tips & Tricks

THE NEW UNITY UI TOOLS

Firstly, the new UI tools of Unity 4.6 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! Invest in your knowledge for your future work with Unity and learn to use the new UI system. 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: In your scene there is an EventSystem object, and a Canvas object in which UI elements are possibly included. Also, you can visually edit your UI components, which you couldn't really do with the old system.

Tip 1: have a look at the "GameObject > UI" dropdown menu, here you can find some useful UI components for you game.
Tip 2: have a look here and here for tutorials and an overview of the new UI system.
Tip 3: if you want to check the documentation of Unity 4.6, 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 does not fix a broken game, but it greatly improves the game feel, which basically means how smooth and pleasing your game is to play. Snappy or slow movements often block the fun (unless it is part of your gameplay of course). 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!

TIME MANAGEMENT

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