Computer Science 3rd Year Games Design (3020H)

1) Introduction

This year the practicals will consist of a mix of Unity3D and OpenGL in order to give you a good understanding in both. You are required to use C# in the Unity practicals (no Javascript or Boo). The two in OpenGL will be done in Linux. You will be using OpenGL 3.2 which boasts a more modern work flow and uses a newer set of utility libraries. The Senior Lab on 2nd floor mezzanine level of the Computer Science building will have all of the tools required for you to complete these practicals.

The final project for 3020H will act as your 3rd year capstone project when specializing in Games Development. In this project you will have the opportunity to collaborate with artists from both the South African College of Music (SACM) and City Varsity.

2) Resources

There will be a collection of Unity sample files that will be uploaded to Vula that you may use. These samples demonstrate various techniques and should be used as a guideline and not ‘copy-paste’. They were all obtained from the Unity website, found at http://unity3d.com/learn/tutorials/modules. Unity3D is rather popular with the online community and as a result it is rather easy to obtain help online. Important samples for each practical will be listed for your convenience.

a) a Unity Tutorials

These are tutorials by Unity that teach you the basics of becoming a Unity developer. The resources can be found at http://unity3d.com/learn/tutorials/modules and assets can be downloaded via the Unity Asset Store.

Tutorials specific to 2D Unity Game Development can be found:

- http://blogs.unity3d.com/2013/11/12/unity-4-3-2d-game-development-overview/
- http://pixelnest.io/tutorials/2d-game-unity/

b) Theory and Practice

These websites contain tips for working with Unity.

- http://devmag.org.za/2012/07/12/50-tips-for-working-with-unity-best-practices/

c) Documentation

Unity3D is well documented. The documentation can be found http://unity3d.com/learn/documentation and is split up into a User Manual, Component Reference and a Scripting Reference.

3) Marking of Practicals

Practicals will be marked by the tutor according to the requirements. Then they will be compared with all other students' submissions to check for plagiarism and that the work is their own. Copying and pasting of the provided samples **is** plagiarism. Some of the later practicals will be marked by demoing to the tutor, for these appropriate demo times will be arranged closer to the hand in time. Marks are also awarded for coding style according to the following criteria.

a) Consistent layout

This applies to indentation, use of brackets, use of whitespace and the appropriate style of identifiers.

b) Appropriate user defined identifiers

Names that make the code easier to read and understand

c) Comments

Comments are vital during the marking process to explain complex pieces of code that cannot be understood at a quick glance. Also do not comment for the sake of commenting as this makes it difficult to read.

d) Object Orientated Structure

By now you should know to design programs that make use of object orientation to use classes. Do not put everything into one file.

4) Submitting Practicals

All submissions are to be done on Vula before the due date, remember that these will usually be quite large files and take time to be uploaded so don't submit to close to the deadline. If for some reason the upload size is too large (> 200Mb) please contact the tutor to arrange alternative hand in.

5) Compiling of practicals and appropriate use of version control

The Department requires you to use version control (using Git) for your capstone project (as is expected from you when working on bigger projects in industry). We will provide instructions on how to use basic version control mechanisms with Unity in section 6 of this introduction. You will also have access to resources where you can host your work.

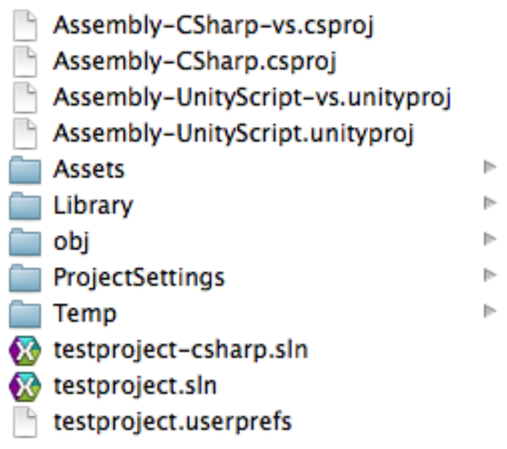
All the practicals are required to compile successfully on the Senior Lab computers, failing to do so successfully will result in a 50% penalty. Failure to use local version control in your practicals (excluding capstone project) will encounter a 10% penalty. Failure to use version control for collaboration and/or a working build of the final hand-in will result in a 20% penalty. Penalties will not exceed 50% in total of the student's final mark.

You are reminded to include a Make build system for the OpenGL practicals which will compile these practicals and link against the necessary OpenGL and utility libraries.

6) External version control with Unity3D

By now you've already had a brief recap on version control in CSC3022H (if not go work though the first consolidation session or the "using departmental GitLab server" walk through). We need to use a .gitignore file to exclude binary data from the local repository (and remote repository for the capstone project). Using git in Windows is basically the same as in Linux. You can grab just plain old git (here) which comes pre-bundled with a Bash shell with all your favorite *nix commands or if you so desire you may install a GUI tool like SourceTree for Windows (http://sourcetreeapp.com/). This is up to you. Feel free to explore: there is a host of possibilities here. Moving on to business...

Create a new Unity project and go look at the folder structure.


Assembly-CSharp-vs.csproj
Assembly-CSharp.csproj
Assembly-UnityScript-vs.unityproj
Assembly-UnityScript.unityproj
Assets
Library
obj
ProjectSettings
Temp
testproject-csharp.sln
testproject.sln
testproject.userprefs

All the information required by Unity lives in *Assets* and *ProjectSettings*. Everything else is generated from there. These two folders have to go into version control, but Unity has to be told to switch to version controlled mode:

1. Switch to Visible Meta Files in Editor → Project Settings → Editor → Version Control Mode

2. Switch to Force Text in Editor → Project Settings → Editor → Asset Serialization Mode
3. Save scene and project from File menu

Initialize an empty repository and add the following .gitignore file into the project directory (one level lower than Assets and Project Settings). Tell git to track the file and both directories. Now commit to your local repository.

Useful resources

- http://docs.unity3d.com/Manual/ExternalVersionControlSystemSupport.html
- Demo project used in lectures is under version control and can be cloned from the following BitBucket url: https://bitbucket.org/blwabona/survival. This can be done using the following command:

```
git clone https://bitbucket.org/blwabona/survival
```

Overview of the capstone project

OR "WHAT TO EXPECT FROM THE BEST LAID PLANS OF MICE AND MEN"

This presents an overview for the final game project for the year. This project will form the capstone project for third year computer science and replaces the project required for CSC3003S. More details on the project requirements and new software engineering deliverables will be released on the CSC3003S Vula tab closer to the time.

A side note about deadlines, the deadlines set in this document are fixed and will not be moved except in extreme circumstances. Enough time will be given for the completion of the practicals.

Project teams will consist of 2-3 people (excluding artist allocations). You will need to choose project partners and indicate this on Vula once the project entry opens up – you will be told when this will happen. This is to ensure that the software engineering requirements of the capstone project is satisfied.

You are expected to put together a ***short*** pitch (slides with pictures and basic game play idea) to the artists at City Varsity and SACM in order to gauge interest and find suitable collaborators for your art and audio game assets. We will tell you closer to the time when meeting times with these external partners can be established.

Once a collaboration is established you will agree on exactly what assets you require and their properties (subject to artist discretion on feasibility). As an example, if your game requires narration you are required to specify dialog cue sheets to the musicians. You will also need to request effects, sampling rate, bit depth, length, along with other specific properties for any ambiances, narrated dialog, music, attractors and reward music. The musicians will be able to assist you in this regard.

Visual assets, including models and artwork, requires exact specifications. These include properties such as aspect ratio, resolution and model mesh triangle count. This has to be done in consultation with the modelers. In the case of rigged visual 3D models it is worthwhile pointing out that you should ask your artists to send you some basic rigged models so that you can play around with this in your game as soon as possible. Rigging has been one of the biggest causes of visual problems in the past.

There is a possibility that ideas for game concepts may be proposed from these external sources as well. You may be able to find common ground between the expectations of the artists and the software engineering and feature requirements of this course.

We ask that you actively collaborate with your colleagues at the SACM and City Varsity, and reach compromises where the need arises. Since this is a Computer Science course you are not expected to create game assets yourself and may download free assets where the need arises (or use placeholders if all else fails).

In conclusion: all these features and assets should come together to produce something that is fun to play. This does not necessarily mean massive amounts of content or over-engineering. You should focus on polishing game mechanics for one re- playable level, instead of having multiple levels with very little challenge to the player.

The battle plan	Hand-in	Start	End
Game Concept Pitch to City Varsity and SACM	No	X	X
Game Design Document	Yes	Wed, 6 April	Wed, 11 May
CSC3003S Prototype (Demo)	Demo	X	X
Final Project Demos	Hand-in & Demo	Mon, 15 August	Mon, 10 October (hard deadline)

Practical 1 – Unity Basics

OR "FLATLAND! A 2D GAME IN UNITY"

The first practical will introduce you to Unity. You are required to make a top down 2D game in a similar fashion to your 2nd year final games project. The theme for this practical will be the traditional shoot-em-up genre. There are many examples of top down shooter games and you are encouraged to be creative. Several playable versions of top down shooter games are available online, e.g. http://www.keepbusy.net/games.php?tag=Top-down+Shooter (Lost Outpost, Color Tanks, Robot Legions are cool examples).

The aim is to get you familiar with the workings of Unity to create games. As this will be your first practical, more marks are going to be awarded to the game framework and implementation rather than the quality of artwork and graphics.

You are required to create a two player top down shooter. However the theme is more a guide and if you wish to make a game that isn't similar to the examples you may do so, but please, discuss your idea with the TA before you start.

You will be required to demonstrate the following in your game:

- **Two player game play**

  The game needs to implement multiplayer game play on the same computer. This is not as difficult as you might think; simply create two player objects and assign them different keys. As this is a top down game, you would not require multiple screens. Your game can be either co-op or competitive.

- **Unity 2D Game Development**

  You will need to use the new 2D Game Development features in Unity. For working in 2D you should set the Editor behaviour mode to **2D** for various settings. This can be done through Edit>Project Settings>Editor and changing Default Behaviour Mode to 2D. This will mean that textures will be imported as sprites and the scene view will default to 2D.

  You need to learn how Unity works (Unity Methods, GameObjects, Components, Scripts, Prefabs) as well as get to grips with C#. Many resources exist online.

  Even though animation is not required, you need to support basic sprites for GameObjects. As mentioned before, marks will not be awarded for quality of the artwork.

  An example project can be found on Unity's website http://unity3d.com/learn/tutorials/modules/beginner/2d .
  You are encouraged to investigate and stick to Unity best practices.

- **Physics**

Unity simplifies collisions with the use of its physics system. You will need to implement basic movement and interaction between game objects (players, enemies, bullets etc.). Take a look at 2D Box Colliders (and polygon colliders for the brave) in order to achieve basic collisions. You will also need to make use of Rigid bodies.

- **Basic AI**

Basic AI is another important game feature. Such AI should control the monsters that run around, with some degree of randomness, randomly as they move towards the player. You would need to make use of colliders and the physics system for movement and interaction.

- **Scenes**

You will need to make use of multiple *Scenes* in Unity. This can either be having a menu as a different Scene or having multiple parts of a level in different Scenes. More marks will be awarded for creative uses.

- **Winning and losing conditions**

The game needs to have some winning objective, like destroying all the monsters and getting more points than the other player or a losing condition which occurs when the players die.

Marking

Marks are awarded for the overall design and feel of the game, a ‘coolness’ factor as well as for any unique additions you make to the game. The use of good visual effects is encouraged but you won’t get penalised for having basic graphics. Refer to the main marking heading at the top of this document for other criteria. Since this is a games course and this will be a complete playable game, marks are awarded for how enjoyable your game is to play.
Finally, focus on the core components discussed above. As mentioned earlier, there are many versions of this game with many features which may be above the scope of this first practical. We encourage students to put as much into the game as possible, but consult the marking guide to make sure you have implemented the most important features first. For example, things like character animations and destructible blocks can be left until last.

Feature	Marks
2-players	5%
Unity 2D	10%
Physics	15%
Basic AI	15%
Scenes	15%
Winning/Losing Conditions	10%
Enjoyment	15%
Extras	15%

Submission date

Hand in by 10:00am Monday, 4 April 2016.

Useful Resources

- **Unity 2D Game Development:**
  - http://pixelnest.io/tutorials/2d-game-unity/
- **Unity Best Practices:**
  - http://devmag.org.za/2012/07/12/50-tips-for-working-with-unity-best-practices/
- **Unity Scripting**
  - https://unity3d.com/learn/tutorials/modules/beginner/scripting

Practical 2 – OpenGL: Basic Rendering and Transformations

OR "OpenGL Bootcamp"

This is your first practical in OpenGL which is designed to get you familiar with OpenGL before you delve into shaders. The aim is to teach you some basics like setting up a standard window and camera, and populating the scene with an object. After you have an object set up, you need to have it move about to demonstrate translations and rotations in OpenGL.

For the OpenGL practicals, you will make use of the SDL library to create a window and respond to input. SDL is a library that provides cross-platform access to functionality commonly required by real-time interactive applications (such as window creation, input handling, file I/O etc). SDL is popular for games development, having been used in FTL, World of Goo, Portal and Dota 2 to state a few examples.

Detailed instructions on how to setup SDL on your machine are available on their wiki (wiki.libsdl.org), however broadly you will need to install graphics drivers, and the libraries/headers for glew and SDL.

For this practical you will be **using OpenGL version 3.2 or higher**..

You are required to demonstrate the following:

1) Basic window setup

Your first task is to set up the scene by loading and displaying an OBJ model. This model should be centred on the origin. Setup code has been provided for you to manage object loading and window setup. You will need to edit this code to add in the required functionality. You will also need to set up the scene camera. The camera will remain fixed in world space; subsequent transformations will be applied to model(s) only. Place the camera on one of the world space coordinate axes, looking towards the world origin (0,0,0).

Open the template framework located under assignments, OpenGL Intro. Use the provided Makefile to build and run and ensure that a window opens and a white triangle is drawn.

2) Load up an object

You are provided with code to load OBJ files (as well as sample OBJ files to load). Use this code to load a model at startup, and have it draw to the screen.

3) Transformations and rotations

Enable the user to translate, scale and rotate the loaded object. Key-presses should be used to change between different modes (e.g., 'R' for to switch between rotation axes, 'S' for scale, etc) and the mouse to drive the actual changes.

4) Compound Transformations

For this task, your application must be able load a *second* model into the scene. Loading a second model should reset all transformations and return the initial model to its world-centered position and default orientation. This new model should be considered a “child” of the original model, and must be placed adjacent to the first loaded (“parent”) model, aligned according to the world space axes. You can do this by finding the bounding box for each model and using the dimensions of the boxes to place your models in world space. This new ‘compound model’ should respond to your rotation, translation and scaling operations as one unit. Conceptually, all transformations are applied to the parent, and then propagated to the attached child model. For example, if you apply a rotation operation, both models should rotate rigidly about the parent’s center of rotation. Similarly, scaling is about the parent’s centre. The models must not separate or display any transformation errors.

5) Colour alteration

The user should be able to change the colour of the model between five different pre-set options using key-presses ('1'-'5'). The choice of colours is up to you.

Plagiarism

Due to the wide availability of OpenGL code around the internet very strict penalties will apply for any code that is found to be not your own! We will be examining these practicals very carefully so please submit code that you have written.

Marking

This practical is designed to be run in Linux and thus failure to compile on the Senior Lab computers will result in a 50% penalty. The majority of the marks are set for the transformations and object loading section which are the primary focus of the practical; you will need to clearly demonstrate mastery of these in order to obtain full marks.

Basic Features	Marks
Load and display model	20%
Basic transformations	45%
Compound transformations	25%
Colour changes	10%

Submission Date
Hand in by 10:00am Monday, 25 April 2016.

Useful Resources
- learnopengl.com
- docs.gl
- opengl.org/wiki
- open-gl-tutorial.org
- open.gl
- wiki.libsdl.org
- GLM (glm.g-truct.net)

Practical 3 – Unity3D: Advanced

OR THE CREATION OF A 3D WORLD

This practical is intended to familiarise you with the 3D side of Unity. You will be setting up a 3D scene in Unity focused around a player model with a weapon or item (gun, flashlight?). You can use any model (a basic Cube would do, but search online for free models (or in the Unity Asset Store)).

You will be tested on your use of Unity's built in functionality for rendering various 3D aspects in a game. This practical will not test you on the ability to create advanced shaders but you will need to understand how to use the built in shaders in Unity.

This practical is designed to prepare you for your final gaming practical. We recommend that you design the practical in such a way that you can easily reuse it or pieces of it when it comes time for your final practical since this will save you some time.

You will need to achieve the following in this practical:

Player Model Setup & Movement

You will need to implement a model carrying a weapon, you will need to have controls set up that allow you to move the model around the world. In addition to basic movement, you will need to be able to move the weapon/item. Appropriate animation of the model will need to be displayed to represent these movements. There are samples that demonstrate all of this functionality, so please refer to the resource list at the end of this practical spec.

Cameras

Next you will need to implement a set of Unity cameras. You will have to add scripts to the cameras to define their behavior. The first camera will be a 3rd person chase camera that has initial acceleration and deceleration when following the model's movement; it's not a fixed camera but has a fluid motion. The second camera you will need to implement is an orbiting camera that will fly in a circle with the model always focused as its target. The last camera will be a first person camera that should have the weapon in view, it must also move up, down, left or right when the player moves the view. Both the 3rd person and orbiting cameras should be configurable in game by allowing the player to set the distance from the model (radius) as well as height easily.

Environment Setup & Physics

Now that the camera is set up you need to create the environment; you will need to place a ground object in the X-Z axis that represents the floor that the model drives on. Next you will need to place several randomly placed randomised objects; feel free to make use of your own. Remember to use Prefabs for multiple instances of an object.

These objects that you create will serve as collidable objects. By now you should be familiar with the Unity physics system. You now have to extend this by implementing 3D physics in your game world. The player model and world objects need to be able to collide. Remember to use Triggers and Colliders.

Ray casting

The Player Model needs to be able to shoot the weapon. In order to detect collisions you will have to implement ray casting. Appropriate objects that are hit by this ray need to be destructible.

Visual Effects

Now that most of the scene is set up you will need to add in some visual effects in the form of lights, texturing, text and particle effects. You will need to make use of different types of lights in your scene. You need to add in textures for each of the random objects that you place in the scene. This is done through Materials and Textures. To get you familiar with certain elements of Unity you will be required to render 3D text to the scene. The final piece of graphics functionality is the use of the built in particle effects engine and identifying a suitable effect that matches your scene well in terms of aesthetics. The Particle System can help make game effects look really professional. This could either be triggered when there are collisions or by player input.

Just a few notes about the randomness factor. In the practical you need to place a set of objects in the environment. These need to be of various model types (sphere, cube, etc) which are placed about the level randomly and also have a randomly assigned material type (texture). This will prevent hard coding of the various aspects in the game and will allow for some diversity and thus keep things interesting.

Sounds

You are required to implement audio for the scene. Examples of useful sounds to have include bullets firing or explosions when objects are destroyed.

Marking

Features	Marks
Player model setup, Movement & Animation	15%
Cameras	15%
Environment setup & Physics	15%
Ray casting	10%
Visual effects • Lights • Texturing • Particle effects • 3D text	30%
Sounds	5%
Randomness	10%

Submission Date

Hand in by 10:00am Monday, 25 July 2016.

Useful Resources:

- http://unity3d.com/learn/tutorials/projects/stealth/player-setup
- http://unity3d.com/learn/tutorials/projects/stealth/player-animator-controller
- http://unity3d.com/learn/tutorials/projects/stealth/enemy-animator-controller

Practical 4a – OpenGL: GLSL Shaders

This practical is designed to improve upon your OpenGL skills that were developed in the introduction practical. In this practical you build upon concepts learned in the first prac, which means you **should** reuse the original code. You are asked to demonstrate your competence with OpenGL shaders, lighting and texturing techniques.

What you need to do

Take the setup provided by the first practical: with the object located in the centre of the world. As before you should support translation (along each axis) of the model you have loaded - your previous solution will already contain this functionality. The camera must be constrained to look at the world centre but must also be able to orbit around this fixed point, based on key press input. Next you will need to add some additional steps for this practical which are outlined below.

1) Phong-shading

You are required to add light sources to the world, minimum of 2. Each of these needs to have different colour values to show up effectively on your object. Per-pixel lighting is required for this and the accumulated value of all the lights needs to be factored in for the final pixel value.

2) Shader files

You will need to create both a vertex and fragment shader for this purpose. This is where you will put the code necessary to have phong-shading in the scene.

3) Moving lights

The lights that are added to the scene need to be able to move: this can either be done by transforming them individually or even having them all orbit in either a clockwise or anti-clockwise direction around the scene. This provides another method to show how the light falls on the object's surface.

4) Texturing

You need to apply a texture to the surface. To ease the task of loading a texture image file, we suggest that you use the freely available C++ single header file **stb_image.h**. You can find this header file at https://github.com/nothings/stb. You can assume that the OBJ model you load contains texture coordinates – if it does not, you should display a message indicating this and simply ignore the texturing stage.

5) Bump Mapping

In order to provide some challenge to the more eager students there is a final 15% available for adding bump-mapping to the surface of the object. This is achieved through shader techniques that perturb the surface of the model. This requires not only the texture file but a secondary image that is a bump-map.

Marking

Features	Marks
Phong shading	15%
Shader files	15%
Moving lights	15%
Texturing	10%
Bump mapping	30%

Submission Date

Hand in by 10:00am Monday, 15 August 2016.

Useful Resources:

- OpenGL Tutorials (Setup & Phong-Shading) http://lighthouse3d.com/opengl/
- OpenGL Tutorials (Various shader techniques) http://nehe.gamedev.net/

Practical 5 – 3rd Year Games Project

OR "YOUR GAME TITLE HERE"

In this practical you will be working in groups to create a fully featured 3D game. There are various aspects your need to incorporate in your game, but apart from including those features you are free to make any kind of game you wish. The first phase of this project will be to create a detailed game specification document. This document must not exceed 3000 words and must detail all the aspects that you are going to be implementing in your final game. Although the only remaining hand-in is the final submission in October, there are set milestones which indicate what you should have accomplished at the given date. There will also be a prototype demo to the T.A. and tutor in mid-September. This prototype is scheduled after you should have a basic world environment and navigation system in place. The dates for the group demonstrations will be scheduled closer to the time. **The final presentations will be done during Wednesday 12th and Thursday 13th October during the lecture slots, the time and place will be confirmed closer to the time**. The features required in the game will be discussed below as well as an outline for the game design document.

Game Features

1) 3D environment

The game world needs to feature 3D assets (models) that are moved about in the world, however, you may choose to utilise a top-down or isometric viewport giving a 2.5D perspective. You will be penalised if you implement this **AND** use 2D sprites for the entire game, the use of 2D HUD elements are acceptable however. The recommended format is that of the 3rd Practical utilising 3D cameras.

2) 2-player gameplay

There needs to be support for two independent players. This can be implemented through the use of multiple viewports that show the view for the different players (Split-Screen). Use the Normalized Viewport Rectangle option in your camera.

3) Use of visual effects (Shaders) *(optional)*

You need to implement certain visual effects, the better the effects the better your marks will be. These can be done in the form of rendertextures (for mirrors or screens), explosions, particle effects, lighting, fog, etc. However, you could also implement Shaders using Unity. The OpenGL pracs would have given you an understanding of how shaders work. The links under resources contain tutorials on how to implement Shaders in Unity.

4) Use of agents

You will need to include computer controlled characters in the game; these may be in the form of friendly or enemy characters depending on your chosen game. The kind of agents you implement will be based on what you learn in the Agents module... More details will be available once you have completed the relevant part of the course. Implementing compelling agent behaviour can become very complex, so please discuss suitable schemes with the TA.

5) Consistent theme

You need to decide upon a theme for your game and stick to it throughout the implementation. Marks will be awarded for consistency.

6) Use of sound (Required)

To enhance the gaming experience, you have to include sound effects in the game. There will be a penalty if your game features no sound.

7) Any other features

Include any other features that I have left off and you think would enhance your game. The better the features you implement the higher your marks.

A note about game assets

You will have access to a group of 3D animators/artists from City Varsity/CFAD who will provide game art, 3D models, and where possible, animation for your 3D models. Please see the section on the Game Design Document below for more details. While you are required to use their services and to treat them as part of your team, you **can** also add your own game assets should you wish to.

Game Project Web page

Each project team is required to produce a web page for the CS game archive:
http://pubs.cs.uct.ac.za/gamesproj/

This is a requirement and is due **before** the final game hand-in. Failure to submit this will result in a **30% penalty on your final Game project mark**. The format for your web page submission is as follows (please see above link for examples):

- A single ZIP file containing the Game project web page and accompanying data
- The data that should be included in each ZIP file is:
    - a text summary of the game: including Game Title, names of developers and artists, genre, game style (e.g. multiplayer), distinguishing features, implementation platform etc.
    - a single PNG/JPG screenshot which you think shows off something cool about your game. This can be a collage of images, as shown in examples
    - a single web page (see below)
- The single web page should be called index.html
- Web page must be static: no PHP, ASP, server side JS etc.
- The page should be structured as shown in the examples: contain a text introduction, screenshots, notes on game play and preferably embedded YouTube or Vimeo clips showing how the game plays. You can include any other

relevant information.

- You could include a Web Build of your game (Unity makes this easy). However, be warned that certain components do not end up looking the same in the Web Player. Make sure to test this before hand-in.

Game Design Document Requirements

The game design document must not exceed **3000 words (min 2000)**. You will need to include a few key features in the design document, but keep in mind this is just to get you to think about your project and work out what you want to do. This will serve as a reference guide and used by the T.A. and tutor to determine the feasibility of your project. A reminder that this will be conducted in the groups you have chosen and must be submitted by only ONE of the group members.
Note that you will start work on the Game Specification/Design in April, after meeting with the artists and presenting your pitch. We will arrange for a meeting with City Varsity/CFAD early in the year since they will be handling most of the art asset generation, most notably 3D models and animation. At the meeting, the CS project partners will **pitch** their game ideas and meet with potential animators and identify someone to include in their team. The animator should be involved in the creative decisions around the look and feel of the game world and objects: they should not simply be viewed as work-horses to produce assets. The animators can guide you on what they can reasonably produce in terms of animations and other assets. This information will need to feature in your game Design Document. The Artist *must* agree to the final asset list. After this meeting, you will have some weeks to liaise further with the animators and to produce the final game design document. This document is due on **11th May**.

The document must include the following:

- Title
  This is the name of your game, or preliminary name.
- Executive Summary
  A brief summary of what the overall game will be. This must highlight the key game features you will be implementing with simple explanations of each. This should be short about half a page in total.
- Style & Theme
  A description of the style of game-play, this must include the technical aspects about the environment and chosen theme for the game.
- Story line (if applicable)
  A brief plot outline for your game, if this is applicable to you.
- Feature synopsis
  This is where you will provide detail about each of the features you are planning to implement. A separate section for each feature listed above is required. Particular attention should be paid to the additional features you plan to implement.
- Item, environment & level design
  Outline the type of weapons or items that will be implemented in the game.

Also include level design ideas and sketches to detail the kind of environment your game will be featuring.

- Asset List

  This must be a final list of all the art assets required in the game. The artist for your group must have agreed to this list.

- Gantt chart (timeline)

  You need not use a Gantt chart to illustrate our timeline; a simple table of dates would suffice. You need to include this to show that you have a sensible plan to get the work done on time.

- A.I. Implementation

  Since you will not know what type of A.I. you will be covering in your lectures, you will need to discuss how you are planning on implementing some form of A.I. in your game. This will include descriptions of the interaction ability of the A.I. and the player. The details on specific A.I. requirements will be released as soon as they are available.

Marking

There is no set marking guide for the final project that will be made available to you. The marking will be done based on the inclusion of the features listed above. Including better quality and more interesting features will give you higher marks. The projects will be evaluated based on all the different submissions from the class.
The weighting for the Game Design document is 10%, 5% for web page. The final demonstration and evaluation counts 85%. This forms the final mark for project.

Section	Marks
Game Design Document	10%
Game Web Page	5%
Final Game Evaluation	85%

Submission Dates

- Game Design Document – 10h00 Monday, 11 May 2016
- Prototype (Demo) – TBA
- Game Web page **-** TBA
- Project Completion – **10:00AM, Monday, 10 October 2016**
- Final Demo – Wednesday 12th & Thursday 13th October 2016