COMP392 – Advanced Graphics

Final Project

The JavaScript 3D Game – Value 30%

Part 1 (Game Concept) – **Value 5%**

- **First Draft of External Game Document, Menu Scene**

Due Week # 11 (Friday April 1), 2016 @ midnight

Part 2 (First Working Level) – **Value 5%**

- **First Working Level**, **Basic Game Mechanics, Scoring System**

Due Week # 12 (Friday April 8), 2016 @ midnight

Part 3 (Second and Third Level) – **Value 10%**

- **Second and Third Working Levels**

Due Week # 13 (Friday April 15), 2016 @ midnight

Part 4 (Finished Version) – Value 10%

- **Instruction and Game End Scenes**

Due Week #14 (Wednesday April 20, 2016) @ midnight.

The JavaScript 3D Game **Maximum Mark: 78**

Overview: Either alone or as a Team of up to 3, you will create an original 3D game. Puzzle games will not be accepted. The game must have a **Menu Scene**, **Instructions Scene**, at least **3 Game-Level Scenes**, and a **Game-Over Scene**. A **scoring system** must also be included. You must use your own graphic and sound assets. You may choose a WebGL API (such as ThreeJS) and a Physics engine (such as PhysiJS) to create your interface. **Note:** you may use the output from an assignment any team member as a starter level.

Instructions :

(26 Marks: GUI, 26 Marks: Functionality, 5 Marks: Internal Documentation, 13 Marks: External Documentation, 4 Marks: Version Control, 4 Marks: Cloud Deployment)

1. Your application will have the following characteristics **(25 Marks: GUI, 25 Marks Functionality)**

a. A **Menu Scene** – that will allow the user to get ready and display at least 3 options: **Play**, **Instructions** and **Exit** (2 Marks: GUI, 2 Marks: Functionality)
b. An **Instructions Scene** – will display rules and instructions on how to win the game (2 Marks: GUI, 2 Marks: Functionality)
c. **Gameplay Scenes** This is where the main game occurs. The game will have at least **3 Game Level Scenes.** Each Game Level must appear and function differently than other game levels. Each Game Level should have a unique goal. (6 Marks: GUI, 6 Marks: Functionality).
d. A **Game-Over Scene** – this will display the player's final score and give the player the option to **Play Again** or **Exit to Menu** (2 Marks: GUI, 2 Marks: Functionality)
e. Player control of an **Avatar** (a vehicle or character) – the main input may be a combination of mouse and keyboard clicks. The player's avatar may have **weapons** or other **devices** that he can use to defeat the computer controlled enemies (3 Marks: GUI, 3 Marks: Functionality).
f. Computer control (AI) of the **enemies**. The enemies should be abundant enough to challenge the player but not be impossible to beat. (3 Marks: GUI, 3 Marks: Functionality)
g. Random opportunities to generate points for the player aside from killing enemies or avoiding hazards (2 Marks: GUI, 2 Marks: Functionality)
h. A **Scoring system** – ensure that the player's score is accurately calculated and displayed somewhere on the **Gameplay Scene.** You may use a JavaScript Web API (such as **EaselJS**) to produce a scoreboard on an HTML Canvas (2 Mark: GUI, 2 Mark: Functionality).
i. The player must have a **life counter** or **health status** that decreases each time his **avatar** is "killed" (1 Mark: GUI, 1 Mark: Functionality)
j. Add **sound effects** for collisions with enemies, collecting points, shooting attacks, explosions, etc. You may use a Web API (such as **SoundJS**) to produce sound effects (2 Marks: GUI, 2 Mark: Functionality).
k. Add a **Game soundtrack** (1 Marks: GUI, 1 Mark: Functionality).

2. Include **Internal Documentation** for your program **(5 Marks: Internal Documentation):**
   a. Ensure you include a program header for each module of your game that indicates: the Source file name, Author's name, Last Modified by, Date last Modified, Program description, Revision History (2 Marks: Internal Documentation).
   b. Ensure you include a header for all of your functions and classes (1 Marks: Internal Documentation
   c. Ensure your program uses contextual variable names that help make the program human-readable (1 Marks: Internal Documentation).
   d. Ensure you include inline comments that describe elements of your GUI Design for your arcade game (1 Marks: Internal Documentation)
3. Include **External Documentation** for your program that includes **(12 Marks: External Documentation)**:
   a. **A company Logo** (0.5 Marks: External Documentation).
   b. **Table of Contents** (0.5 Marks: External Documentation).

c. **Version History** – ensure you include details for each version of your code (1 Mark: External Documentation).
d. **Detailed Game Description** – describing how your game works (1 Mark: External Documentation).
e. **Controls** (0.5 Mark: External Documentation).
f. **Interface Sketch (Wireframes)** – this section should include wireframes of each of your game screens with appropriate labels (2 Marks: External Documentation)
g. **Screen Descriptions (Screen Shots)** – Include at least 6 screen shots for your game: 1 for your Start Screen, 1 for your Gameplay Screen, 1 for your Game-End Screen and 1 for each level of difficulty (3 Marks: External Documentation).
h. **Game World** – Describe your game environment (0.5 Mark: External Documentation).
i. **Levels** – Describe each of your game levels or challenge levels (1 Mark: External Documentation).
j. **Characters / Vehicles** – Describe the character's Avatar (0.5 Mark: External Documentation).
k. **Enemies** – Describe the computer-controlled enemies and how they function (0.5 Mark: External Documentation).
l. **Weapons** – Describe any weapons available to the player (0.5 Mark: External Documentation).
m. **Scoring** – Describe how the player can score and how the score is calculated (0.5 Mark: External Documentation).
n. **Sound Index** – Include an index of all your sound clips (0.5 Mark: External Documentation).
o. **Art / Multimedia Index** – Include examples of your image assets. Each image should be displayed as a thumbnail (0.5 Mark: External Documentation).

4. Share your files on **GitHub** and deploy to a **Cloud Service** (Microsoft Azure, Heroku, etc.) to demonstrate Version Control Best Practices **(4 Marks: Version Control, 4 Marks: Cloud Deployment).**

a. Your repository must include **your code** and be well structured (2 Marks: Version Control).
b. Your repository must include **commits** that demonstrates the project being updated at different stages of development – each time a major change is implemented (2 Marks: Version Control).
c. Ensure your game is live and online. Deploy to a Cloud Service of your choice (4 Marks: Cloud Deployment).

Optional Game Features (i.e. Potential Bonus Marks).

A. Include a final "boss monster" to defeat.
B. Add power-ups for the player's **avatar** (e.g. extra speed, a shield) that he can add to his "inventory" and use whenever he chooses.
C. Add Cheat Codes.

D. Create a mini-game in a section of the main arcade game (e.g. disarm a bomb, pick a lock, etc.).
E. Empower the player to gain an NPC (non-player character) computer-controlled ally.
F. Make it possible for the player to save / load his game.

SUBMITTING YOUR WORK

Your submission should include:

1. An external document (MS Word or PDF).
2. A link to your project files on GitHub.
3. A link to your live site on a Cloud Service of your choice.
4. Your project files zipped and submitted to e-centennial

This assignment is weighted **30%** of your total mark for this course.

Late submissions:

- 10% deducted for each additional day.

External code (e.g. from the internet or other sources) can be used for student submissions within the following parameters:

1. The code source (i.e. where you got the code and who wrote it) must be cited in your internal documentation.
2. It encompasses a maximum of 10% of your code (any more will be considered cheating).
3. You must understand any code you use and include documentation (comments) around the code that explains its function.
4. You must get written approval from me via email.