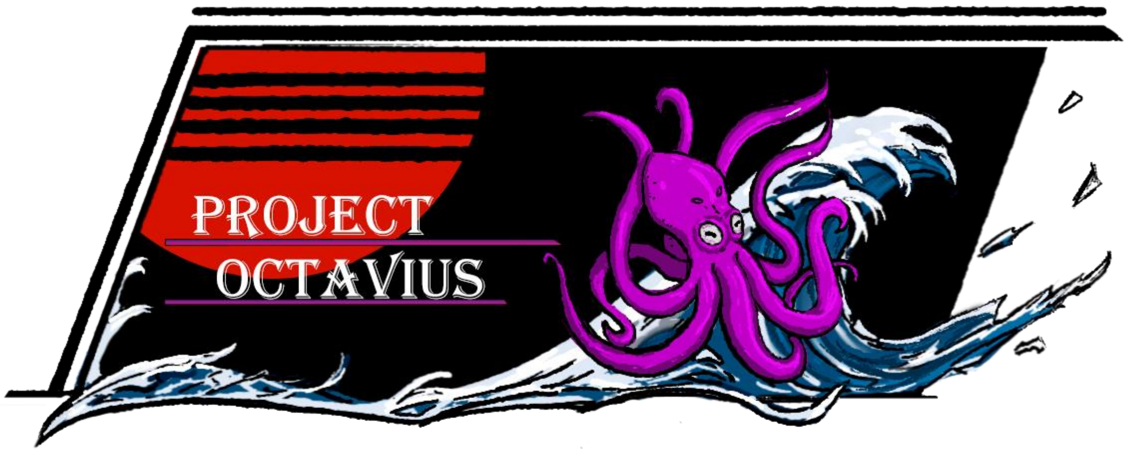


CYBER ARENA TV
PRODUCTION PLANNING
TECHNICAL DESIGN DOCUMENT

Team Name: Project Octavius

SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

Contents

About

Cyber Arena TV is a reality TV show set in the future where contestants have multiple mini games for players to play in. It is a futuristic game show where regular people enter the digital arena and duke-it-out against other Gladiators in a multitude of different last-person-standing challenges and one-on-one combat. These challenges each test different skills whilst also being easy to understand and learn.

The purpose of this Technical Design Document is to aid in the critical analysis of the problems we may face ahead and give a proposed solution, while also communicating priority and effort with the other team members. It will also provide essential information about our game Cyber Arena TV to our client. This information also allows all programming members and other members of the team to achieve their goals.

Change Log

Version	Author	Date of change	Description
0.0.0	AIE	22/09/2022	Initial Template created
0.0.1	Paul	12/10/2022	Template's Title changed to Cyber Arena Tv TDD. Filled out the following sections: About, Team Members, Software Table, Accounts Table and Version Control Team Member list.
0.0.2	Paul	13/10/2022	Filled out the Accounts table and the Windows / PC target platform and halfway finished the WebGL target platform.
0.0.3	Matt	13/10/2022	Added player control schemes and game data management to the custom game systems.
0.1.0	Paul and Matt	13/10/2022	Filled out 1 / 3 of the TDD.
0.1.1	Paul	14/10/2022	Filled out the Commit message format section and cleaned up the TDD.
0.2.0	Paul	14/10/2022	Filled out 2 / 3 of the TDD.
0.2.1	Paul	14/10/2022	Filled out the WebGL Target Platform Section
0.2.2	Matt	14/10/2022	Filled out the level generation, environmental hazards and rewrote player control scheme in the custom game systems section.
0.2.3	Paul	14/10/2022	Filled out the Mobile Target Platform section and completed the Target Platform section.
0.2.4	Matt	17/10/2022	Completed the Coding Standards section
0.2.5	Paul	18/10/2022	Fixed TDD by pasting text in from version 0.2.3
0.2.6	Matt	19/10/2022	Completed Technical Goals and Challenges + finished content for Custom Game Systems
0.2.7	Matt	20/10/2022	UML Diagrams to finalise the Custom Game Systems section
1.0.0	Paul And Matt	20/10/2022	TDD Completed

Team Members

Name	Role
Elliot Hayward	**Game Designer**
Jack Linton	**Game Designer**
Annita Te	**Artist**
Ash Purnell	**Artist**
Crystal Zhang	**Artist**
Matthew Carver	**Programmer**
Nicolien Caerteling	**Programmer**
Paul Ellul	**Programmer**

Development Environment

This section outlines the required software and systems required for development of Cyber Arena TV.

Software Requirements:

Software	Version	License	Used By	Used For
Unity 3D	2020.3.5f1	Education	Programmers, Designers, Artists	Development of Game
JetBrains Rider	2022.2.3	Education	Programmers	Programming software for Cyber Arena TV
Maya	2022.2	Education	Artists	3D Modeling, Rigging and Animation
Photoshop	23.0.2	Education	Artists	Drawing & Texturing
Substance Painter	2022.0.1	Education	Artists	Texturing
ZBrush	2022.0.1	Education	Artists	Sculpting & Modeling
MarmoSet	7.4.0	Education	Artists	Baking Lighting
PureRef	2022	Education	Artists	Mood boards and references
Krita	4.4.5	Education	Programmers, Designers, Artists	Texturing, level Design
HacknPlan	2022	Education	Programmers, Designers, Artists	Project Organization
Microsoft Teams	2022	Education	Programmers, Designers, Artists	Communication with the team
Microsoft Office365	2022	Education	Programmers, Designers, Artists	Project Organization and documentation
GitHub	2022	Education	Programmers, Designers, Artists	Repository of the main project Cyber Arena TV
Magic Voxel	2022	Education	Artists	Used to make voxel art

Accounts

The below table outlines any accounts that may or will be needed for the development of the Cyber Arena TV.

Account/Service	License	Used By	Used For	Owner
GitHub	Free	Programmers, Artists, Designers	Contributing to projects hosted on GitHub	Individual
Hack'n'Plan	Free	Programmers, Designers, Artists	Used for project planning	Individual
Microsoft Office 365	Education	Programmers, Designers, Artists	used for communication and documentation	Individual
Maya	Education	Artists	3D Modeling, Rigging and Animation	Individual
Photoshop	Education	Artists	Drawing & Texturing	Individual
Substance Painter	Education	Artists	Texturing	Individual
ZBrush	Education	Artists	Sculpting & Modeling	Individual
Marmoset	Education	Artists	Baking Lighting	Individual
JetBrains Rider	Education	Programmers	Programming software for Cyber Arena TV	Individual
Unity 3D	Free	Programmers, Designers, Artists	Development of Game	Individual
SoundSnap	Education	Designers	Making Sound Effects	Individual

Third Party Libraries

Asset/Library/Package name	License	Used For
Cinemachine	Unity package manager	Additional camera settings and functionality
Universal Render Pipeline	Unity package manager	Shaders and optimized graphics
Text Mesh Pro	Unity package manager	High quality text with extra formatting options
SoundSnap	Education	Sound Effects

Version Control

Repository

https://github.com/AIESydYr1Production2022/production-project-octavius.git

Contributors

- Paul Ellul - @BudgieBoi
- Matthew Carver - @MatthewLCarver
- Nicolien Caerteling - @ceresVI
- Elliot Hayward - @Ehayward
- Jack Linton - @LavaLP
- Annita Te - @Annitalikeice
- Ash Purnell - @AshPurnell
- Crystal Zhang - @ohmymisaki

Commit Message Format:

Our GIT message format will contain the following information in each commit we make to the repository.

- **Type**: Represents the type of change, often the “Type” can be inferred based on the associated ticket in your project management tool, which may include:
    - **Feature:** This commit implements a new feature, makes progress, or improves a feature.
    - **Fix:** A bug has been identified; this commit relates to changes that resolve the issue
    - **Refactor:** The code, folder structure, or other parts of the project needs some adjustments to better support maintainability and addition of new features.
    - **Performance**: This commit has changes that improve the projects performance
    - **Docs:** This commit has made changes to documentation
    - **Asset:** This commit has made changes or added a asset to the project

- **Scope:** Refers to the area of the project being changed, could refer to things like (menu) (inventory) (save system) (level) (controls) etc. Scope's may change throughout development but can broadly identified. Outline the scopes below that seem suitable for your project

- **Summary:** A short description of what has been changed.

Commit Message Format: `Type (scope): Summary`
Examples:

Feature (menu): Added Exit button to main menu
Fix (menu): Updated button prefab with so that hover works on web builds
Feature (sandbox): Added rock asset to test scene, Created Rock prefab
Docs (readme): Updated project summary and title
Performance (level): Improved level generation code
Performance (player): reduced texture resolution and model vertex count for the player to ensure performance on web builds
Asset (Environment): added environmental buildings, trees, and benches to the project
Asset (Character): added the main character to the project

Target Platform

This project will be deployed to the following platforms:

- Windows / PC
- WebGL
- Mobile

Windows / PC

Windows / PC Limitations

- Performance constraints on the game would greatly impact the game Cyber Arena TV as the amount of assets in each minigame would greatly impact how smooth each minigame will play.

- The Available inputs for Windows / PC are keyboard, mouse and controller. The three main inputs used today.

- The Graphics capabilities that are going to be limited in each of the minigames are rendering graphics in the 3D top-down setting, reflections from reflective surfaces and high poly counts. Ultimately this comes down to the consumers PC specifications, but we aim to make Cyber Arena TV play smoothly on low System Specifications.

Minimum Windows / PC Specs

The minimum system requirements that are required to run Cyber Arena TV on Windows / PC are as follows.

- CPU: Intel(R) Core(TM) i7-9700F CPU @ 3.00GHz (8 CPUs), ~3.0GHz
- GPU: Any Graphics Card with 256 MB of VRAM or higher
- RAM: 8 GB of memory
- HDD: 15 GB of free space
- OS: Windows 10 Education 64-bit (10.0, Build 19044) (19041.vb_release.191206-1406)
- DirectX: DirectX 12

These specifications are the minimum requirements to play Cyber Arena TV on Windows / PC…

Release Build Instructions

The build release instructions are as follows:

1) Select the Windows / PC build option in the build settings tab (By default it is already selected).
2) add your scenes into the build view. Make sure the first scene you add is the main menu scene or any other scene you want the player to see first.
3) go into player settings at the bottom left of the screen and customize all the player settings to the groups liking e.g.: Icon, splash Screen, Company Name, product name etc.
4) 4.1) After all the steps above are completed click the Build button or if you want to run the game immediately after building click the Build and Run button.
   4.2) once you click an option it will ask where to put your build. You can either choose a folder or create one to place the release build of the game. (If you want to change the name of the executable e.g.: CyberArenaTV.exe. Go into player settings and change the Product Name to the desired name you want).

WebGL

WebGL Limitations

- WebGL is a low-level graphics API and the sheer size of models and accompanying memory limitations can make it difficult to run in the browser.
- WebGL presents latency issues which will deter the player who will expect fast load times in Cyber Arena TV. The reason behind this is because everything has to be transferred to the browser before it can run.
- WebGL is difficult, if not impossible, for deploying 3D assets to mobile web browsers. The problem is Unity currently doesn't support the use of WebGL for mobile web browsers. Despite occasional successful deployments on high-end devices, most devices don’t have the memory to support the builds.

Minimum WebGL Specs

The minimum system requirements that are required to run Cyber Arena TV on Windows / PC are as follows.

- The Required web browsers are Chrome, Firefox, Internet Explorer 11, Edge, and Safari 9 or later.
- 2 gigabytes of system memory
- A GPU with at least 512 MB of video memory or higher
- A GPU with a maximum of 1 GB of RAM
- Minimum GPUs are the following:
  AMD/ATI: Radeon 4xxx or higher
  NVIDIA: GeForce 2xx or higher, Quadro FX 3800 or higher
  Intel: HD 4000 or higher

Release Build Instructions

The build release instructions are as follows:

1) Select the WebGL build option in the build settings tab.
2) add your scenes into the build view. Make sure the first scene you add is the main menu scene or any other scene you want the player to see first.
3) go into player settings at the bottom left of the screen and customize all the player settings to the groups liking e.g.: Icon, splash Screen, Company Name, product name etc.
4) 4.1) After all the steps above are completed click the Build button or if you want to run the game immediately after building click the Build and Run button.
   4.2) once you click an option it will ask where to put your build. You can either choose a folder or create one to place the release build of the game.

Mobile

Mobile Limitations

- A big difference when making games for mobile devices and even game consoles altogether is that your game must handle memory leaks way better than on PC Computers
- Working in closed environments. mobile devices have a fixed amount of RAM where you must assure you are handling memory in a proper way.
- Limited Input methods since mobile devices rely heavily on the touch input method and not something external and physical like a controller or keyboard.

Minimum Mobile Specs

The minimum system requirements that are required to run Cyber Arena TV on mobile are as follows.

- 16 GB of free space

- 2 GB of RAM
- CPU: Dual-core 1.84 GHz Twister
- GPU: PowerVR GT7600 (six-core graphics)

Release Build Instructions

The build release instructions are as follows:

1) Select the Android build option in the build settings tab (as we cannot use iOS build option as you need to pay for it).
2) add your scenes into the build view. Make sure the first scene you add is the main menu scene or any other scene you want the player to see first.
3) go into player settings at the bottom left of the screen and customize all the player settings to the groups liking e.g.: Icon, splash Screen, Company Name, product name etc.
4) 4.1) After all the steps above are completed click the Build button or if you want to run the game immediately after building click the Build and Run button.
   4.2) After clicking either option it will ask where to build the application and to name the application. Simply choose a folder or the dedicated builds folder and name the application e.g., CyberArenaTV.apk.

Deliverables

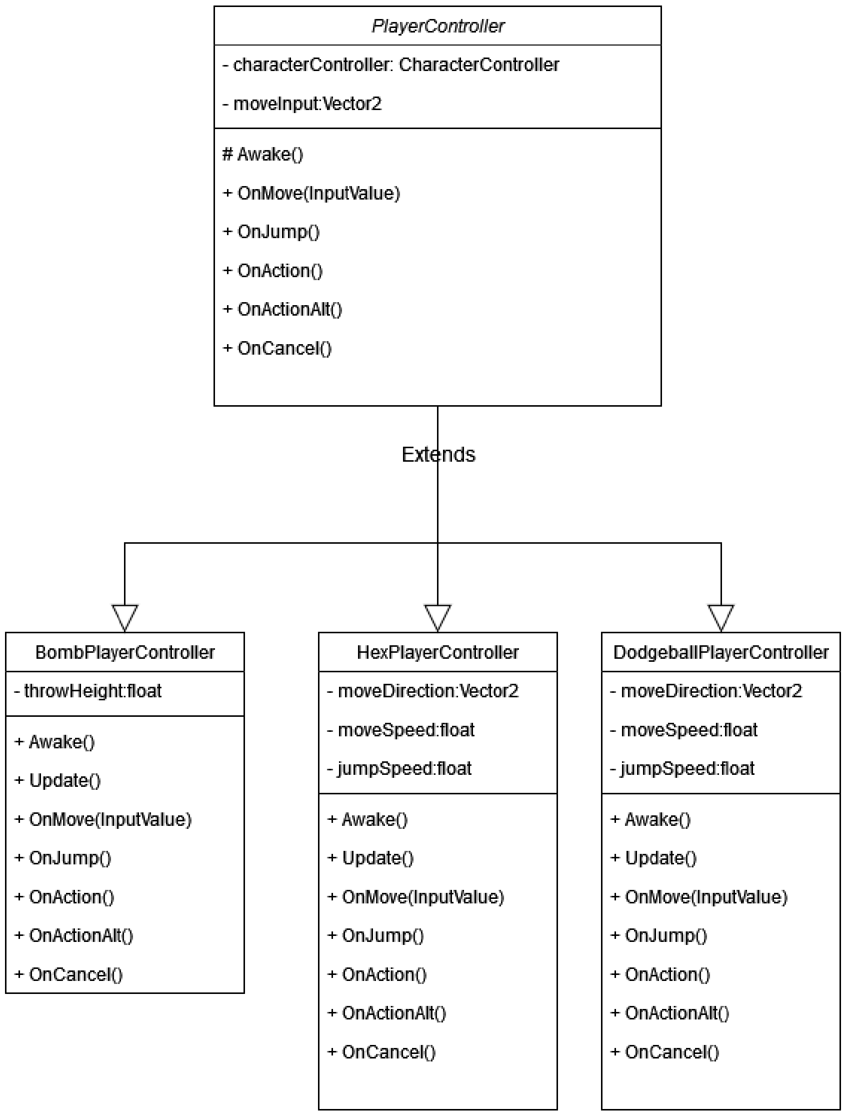
A Build of the project should be generated every 2-3 days and placed in the following location:

E:\CyberArenaTVBuildsFolder\Version#.#.#\CyberArenaTV Version #.#.#.exe
and
Teams Project Octavius Builds folder on Microsoft teams

For storing all future builds, we will be using an external HDD or SDD to store the builds separately as to not expand the size of our GitHub project folder so we can all work on the game without any storage issues. We will also be Uploading the builds to our Microsoft teams build folder in the files tab so everyone in our team can access it including the client.

Build ID:
For our build ID we will be using semantic versioning (also known as **Sem Ver**). This way when we update each build of the game, we can easily put in the semantic versioning what changes we have made, for example: Release. Milestone. Patch.

Custom Game Systems



- Player Control Scheme

  The new Unity Input System will be implemented for character controls.

  - Move - Value action type utilizing a Stick control type
    - Gamepad - Left Stick path
    - Keyboard & Mouse - Composite type of 2D Vector and Digital Normalized mode
      - WASD
  - Jump Button - Button action type
    - Gamepad - Button South path
    - Keyboard & Mouse - Space path
  - Action Button - Button action type
    - Gamepad – Right Trigger path
    - Keyboard & Mouse – Left Button Mouse path

- ActionAlt Button - Button action type
  - Gamepad – Right Trigger path – Hold Interaction
  - Keyboard & Mouse – Left Button Mouse path – Hold Interaction
- Cancel Button - Button action type
  - Gamepad - Button East path
  - Keyboard & Mouse - Control path

A Player Controller abstract class will have multiple children for each different control scheme that will implement the behaviour required for the specific mini game.

- Game Data Management

  Game and Player data management will be managed and accessed from a static class instance. The class will access 4x Player ScriptableObjects for the data saving and loading. The device ID will be captured on player join in the main menu and stored into the ScriptableObject along with their character selection.

  On load of a new scene when a player joins, an empty GameObject with a script will capture the device ID and search for their scriptable object to instantiate the Prefab along with other relevant data (score, wins etc.)

- Level Generation

  The Icebreaker mini game will require a grid generation algorithm to produce specific level shapes, heights, and complexity.

  An empty object with the script “HexPlatform” will contain an enum for the “*platformType”*, an enum for the “*platformHeightStyle”* and with those parameters generate a hexagon GameObject and position it on a grid.

- Environmental Hazards

  A Sphere appears in one or more mini games that will directly impact the player/s experience. The Sphere will move akin to the ball in brick-breaker or pong with no gravity and rebounding off the surface collided with.

  The Sphere will always be generated from within a random Y and Z range of an empty GameObject to look like it is appearing from a random place from the sides of the game screen. The Sphere will randomly select a player from the player list, loop through the HexTile list, using the HexPlatform object reference, to determine the HexTile the selected player is standing on and will move directly towards that HexTile world position.

  The Sphere will be destroyed when it has reflected offscreen and the empty GameObject/s will start a timer with a random time. The first to reach less than 0.0f will create a new Sphere and the cycle will continue to the conclusion of the mini game.

Coding Standards

Naming Conventions

Namespaces are all PascalCase

```
Avangarde.CurrentGame.UI.MainMenu
```

Classes & Methods are all PascalCase

```
public class MyClass
{
    public void SomeMethod{}
}
```

Static Fields should be PascalCase

```
public static int SomeStaticValue = 10;
```

Parameters should be camelCase

```
public void HighlightElement(bool someCondition)
```

Add Callback suffix to delegates

```
// Declare a delegate type for processing a user:
public delegate void ProcessUserCallback(User u);
```

Properties are PascalCase

```
private string name;
public string Name
{
    get
    {
        return name;
    }
    set
    {
        name = value;
    }
}
```

Add On prefix to events and actions

```
public UnityAction OnDeath;
```

Best Practices

Declarations

Always implicitly use the access modifier to remove ambiguous declarations

```
private int privateVariable;
```

Always use a single line for variable declarations to keep the code clear and readable

```
private int firstVariable;
private int secondVariable;
```

Always preface an interface with an **I**

```
IPointerDown
```

Public fields

Minimise public variables by declaring them as private with a public accessor property.

```
private string name;
public string Name
{
    get
    {
        return this.name;
    }
    set
    {
        this.name = value;
    }
}
```

If a private variable needs to be seen but not accessed within the Inspector, use a SerializeField attribute.

Ensure that all functions are prefixed with a summary for readability.

Brace style

```
public void CreateSomething()
{
    //code
}
```

```
switch (someExpression)
{

    case 0:
        {
            DoSomething();
        }
        break;

    case 1:
        {
            DoSomethingElse();
        }
        break;

    case 2:
        {
            int n = 1;
            DoAnotherThing(n);
        }
        break;
}
```

Always use braces for conditional statements that the compiler could run without braces.

```
for(int i=0; i<10; i++)
{
    ExecuteSomething();
}
```

```
[SerializeField]
private bool hasHealthPotion = true;
```

If a magic number needs to be used, always create a named variable for it to avoid confusion.

```
float magicNumber = 3.5f;
float yHeight = transform.position.y + magicNumber;
```

Unity Best Practices

Define a class for an individual GameObject and avoid using public index-coupled arrays.

```
[Serializable]
public class Weapon
{
   public GameObject prefab;
   public ParticleSystem particles;
   public Bullet bullet;
}
```

If these need to be assigned in the inspector, create a separate class.

```
[Serializable]
public class Bullets
{
   public Bullet fireBullet;
   public Bullet iceBullet;
   public Bullet windBullet;
}
```

DO NOT USE (especially in the Update method):

```
GameObject.Find();
```

```
someObject.GetComponent()
```

Use Enums where possible instead of coupled arrays.

```
public void WindAttack()
{
   Fire(bullets[WeaponType.Wind]);
}
```

Use singletons for convenience (AudioManager, GameManager) but avoid using them for unique instances of a prefab (Player).

Define static properties and methods for public variables and methods that are used often from outside the class. Write GameManager.Player instead of GameManager.Instance.player.

Coding Standards Enforcement

Weekly Code Reviews

A weekly code review will be held immediately after the Wednesday Scrum meeting. The team will share screens on Microsoft Teams, quickly step through their code, and makes notes for edits to keep in line with the coding standards. This process is to ensure that DRY code is produced and will have the added benefits of keeping the team cognisant of the codebase and to motivate members to contribute more if they have the capacity.

Jetbrains Rider Settings

Appropriate settings will be created in Jetbrains Rider and exported for the team to use so there is consistency in the warnings that will prompt the team to keep their code consistent and reduce the time spent in the weekly code review sessions.

The addition of Linting tools has been assessed and deemed excessive for the scope of this project.

Technical Goals and Challenges

Technical Goals:

- Maintaining 60 FPS on the minimum specifications whilst maintaining the graphic fidelity of the scene, emissive lighting, dynamic camera and multiple players.
- Saving, loading and managing the Game Data for the play session across multiple game-types.
- Fast load times.

Technical Risks:

- Frame dips as the camera moves in and out according to the player/s controls that renders more objects in the scene.
- Data loss between scenes and/or incorrect data assignment due to PlayerInputManager load order.
- Slow loading due to unoptimized data management and/or high graphical load in the scene.

Risk Avoidance

In order to maintain a minimum FPS of 60, there are several approaches to alleviate the risks:

- Use Lightmapping to bake the lighting for the static objects in the scene. This should increase runtime for a factor of 2-3 times.
- Texture compression
- Occlusion Culling

To minimise the risk of incorrect an/or loss of data, the solutions are:

- Create a stream reader/writer class to data storage.
- Utilise a static class to store player, game and level data for accessibility across scenes.
- Leveraging the PlayerInputManager API for device registration and pairing to the player data.

In order to maintain fast loading times, the solutions are:

- Utilise binary conversion for string reading and/or stored data where possible.
- Conduct manual garbage collection to maximise speed.