tranquilVR (Relaxation & Sleep Aid)

Welcome!

This is the home page for your team space within Confluence. Team spaces are great for sharing knowledge and collaborating on projects, processes and procedures within your team.

Writing Assignment 1 Content

W4: Architecture and Design Materials Checkpoint

- W4: Design Materials
  - Non-Obvious Design Decisions
  - Classes, Functions, and Files
- W4: Architecture Materials
  - Architectural Style
  - High-Level Architecture
  - Logical Components

W6: Testing Materials Checkpoint

- Manual Testing Overview
  - Script 3 - Space Scene
  - Script 4 - Fairy Tale Scene
  - Script 5 - Underwater Scene
  - Script 1 - Main Menu
  - Script 2 - Scene Selection
- Automated Testing
- Traceability

W5: Architecture and Design Review - Student Life

W3: Getting Started Review

W1: Getting Started

W7: Getting Started Response

W8: Architecture and Design Revision and Response

W9: Project Wrap-Up

- Non Functional Requirements
- License
- High-Quality Commits
- Testing Quality

About Us

Alex Vilkomir

Lopez, Steven

Holguin Herrera, Bryan

Rada, Tyler

Vose, Avery

El-Halabi, Lilah

Verceles-Zara, Jan Andre

Blog stream

Blog stream

Create a blog post to share news and announcements with your team and company.

Recently updated

W9: Project Wrap-Up

Create blog post

about 3 hours ago • updated by El-Halabi, Lilah • view change

Testing Quality
about 3 hours ago • created by El-Halabi, Lilah

High-Quality Commits
about 3 hours ago • created by El-Halabi, Lilah

License
about 3 hours ago • created by El-Halabi, Lilah

Non Functional Requirements
about 3 hours ago • created by El-Halabi, Lilah

W8: Architecture and Design Revision and Response
yesterday at 7:35 PM • commented by Alex Vilkomir

W8: Architecture and Design Revision and Response
Apr 20, 2022 • updated by El-Halabi, Lilah • view change

W8: Architecture and Design Revision and Response
Apr 19, 2022 • updated by Rada, Tyler • view change

W7: Getting Started Response
Apr 13, 2022 • updated by El-Halabi, Lilah • view change

W7: Getting Started Response
Apr 13, 2022 • updated by Vose, Avery • view change

W1: Getting Started

Personas

Persona: Jeffrey Bombardo (22M)

Persona Details

Name: Jeffrey Bombardo

Jeffrey has been feeling anxious all day. For the past two days, he has been waiting for a call back from a local start-up. He recognizes this feeling as the precursor to a panic attack so in order to prevent it he starts up *tranquilVR*. Jeffrey selects his favorite environment and puts on his Vive, then starts the environment. Jeffrey spends around 30 minutes meditating until he feels calm.

Persona: Quarl Tuffin (32M)

Persona Details

Name: Quarl Tuffin (32M)

Works graveyard shifts at a local grocery store Mall-Wart. Quarl only has high school education. Has trouble sleeping due to tinnitus he acquired while in active duty.

Scenarios

Scenario: Jeffrey Bombardo

Scenario Details

Name: Jeffrey Bombardo

Overall Objective: Finding a way to calm his panic attacks

Personas Involved: Jeffrey Bombardo

What's Involved: Whenever they feel like a panic attack is coming on Jeffery puts on his Vive and starts an environment

Problems That Can't Be Addressed: We can't fix any motion sickness issues.

Possible Ways To Tackle These: Emphasizing a focus point and establishing stability.

Narrative

Jeffrey has been feeling anxious all day. For the past two days, he has been waiting for a call back from a local start-up. He recognizes this feeling as the p recursor to a panic attack so in order to prevent it he starts up tranquilVR. Jeffrey selects his favorite environment and puts on his Vive, then starts the envir onment. Jeffrey spends around 30 minutes meditating until he feels calm.

Scenario: Quarl Tuffin

Scenario Details

Name: Quarl Tuffin

Overall Objective: Sleep aid

Personas Involved: Quarl Tuffin

What's Involved: They are getting ready to sleep, so they open up their desktop and start up an environment in desktop mode

Problems That Can't Be Addressed: Building up a tolerance with the screensaver and not being able to actually relax

Possible Ways To Tackle These: Using a randomized element in each environment that spawns sporadically

Narrative

Quarl Tuffin, a recently discharged soldier, has started an overnight stocking job at Mall-Wart. Though he signed up for this job he has realized that he must alter his sleep schedule in a drastic manner. He must shift it so that he has the energy to work 11 PM to 6 AM shifts with a reasonable amount of energy and gusto. Though this should be relatively simple to adjust, for Quarl this is not so. When he was deployed in Afghanistan, he dealt with constant bombings and gunfights. These firefights and episodes happened so frequently that he developed tinnitus, a constant ringing of the ears typically seen in much older people. Due to this, he must listen to a different sound as to give his mind a break from the singular-noted ringing.

As a remedy to this, he uses the simplified desktop functionality within tranquilVR to play different sounds and display images that employ low light effects to allow for the atmosphere to be optimal for rest. As he rests, the app is sent to sleep after a specified timeframe so that the app is not constantly running all night.

Initial Features

VR Integration (Vive Headset)

- One randomized moving element per environment
- Optional VR vs. Manual view

Menu Screen

- Volume control
- Scene selection
- Sleep timer (menu 15 – 60 min)
- Default music vs white noise (white + pink + brown)
- White noise changes environmental lighting.
- Device Compatibility

Minimum of 3 Different Environments

- **Underwater Ocean setting**
    - Schools of fish swimming (Boids)
    - Coral “sculptures”
    - Whale noises
    - Dolphins
- **Space (free falling stars, lights, planets, nebula gases)**
    - Twinkling stars
    - Spinning planets
    - Meteors passing
- **Fairy Tale Forest**
    - Falling leaves
    - Bright colors (possibly rainbow near the waterfall)
    - Waterfall
    - Small creatures (fairy, squirrels, rabbits, etc.)

User Stories

Scenario 1 Story:

Jeffery opens the application and decides on which environment he wants to watch to calm him down. His choices are underwater, fairytale, or a space environment. He then connects his Vive headset to ensure it's compatible.

Task: Implement environments and device compatibility feature

Scenario 2 Story:

Quarl opens the application in the desktop view and selects an environment that will help him fall asleep. He selects a 30-minute timer to ensure the program doesn't run longer than necessary.

Task: Implement environments and sleep timer

Configuration Management

How will GitLab be used for group projects? What are the rules around commits and branches?

GitLab will be used as the main repository. We will use a single main branch to house the code and assets.

What is expected of commit messages, and how will this be enforced (if it is)?

A brief description must be provided to let the other group members know what was added/changed.

What type of workflow will be used (feature branches? GitFlow? No branches?)

Within GitLab each scene will have its own branch to help keep track of the specific changes. There will also be a "commonly used" code branch that must be used in each scene.

Code Rules

What is the technology stack?

Our stack encompasses the interaction of the following frameworks and technologies:

- Unity 2020.3.26f1
- Unity Collaborate
- Unity DOTS
- Visual Studio Code
- C# Scripting
- HTC Vive VR Headset

How can new team members get set up (onboarding)?

New team members will need to download the Unity version listed in the technology stack and be given a seat on the Unity website. They will then will be able to download the project files from the unity hub and begin working.

What linters and analysis tools will be used?

Visual studio code has a built-in linter that will be used to maintain stylistic code choices.

Are there other ideas for tools that need to be explored?

Unity is moving away from Collab to Plastic SCM which allows for branches other than main we may have to transition to this system eventually. We have now transitioned into using GitHub desktop to manage code so our main repo is on GitLab.

What technologies do some team members need to learn? How has this been factored into the project plan?

The team will need to review their knowledge of both Unity and C# to be effective team members. This has been accounted for by allocating some time for people to get up to speed before major changes or pushes are made.

Testing Rules

You need to automate at least part of this semester, so how can you automate?

No automatic testing is available for Unity, so manual testing and internal class level checks will be used to test the software. An example of these internal class level tests is the “GetComponent” function that will need to be checked to prevent null references which are a common source of errors.

How does testing interact with commits (always test before commit? all tests pass? all tests pass before the merge?)

All tests must pass before commit. All code will be merged into main so it is important that it does not break other code.

W3: Getting Started Review

Reviewing **Student Life** writing assignment I:

1. Is the team charter complete? Are responses to all items clear?

Yes, all of the information is included and accurately label.

2. For each persona, are the details about the persona stated clearly? Is all the standard information for a persona provided? As a reminder, this should include personal information about the individual; details of the individual's job (if relevant, or if helpful in understanding the persona); details of the individual's education and experience; and details of the individual's interest in the product. Also, are the personas distinct? Having two personas that are essentially identical does not help in figuring out the product.

Persona 1: All the details are clearly stated and all the standard information is provided.

Persona 2: All the details are clearly stated and all the standard information is provided.

Both of the personas are distinct and they are pretty well defined and include different details to differentiate between them.

3. For each scenario, are the details about the scenario stated clearly? The scenario should include a scenario name; the personas involved; an overall objective (what are the personas trying to accomplish?); steps needed to reach this objective; limitations of existing systems that the product would address; and possible ways the product could help reach the objective. Scenarios should be tasks that the personas are trying to accomplish, and should generally not include the persona buying your product, but should describe how they would actually use it. Beyond this, are the scenarios distinct?

Since the scenarios are used to discover features for the product, scenarios with significant overlap do not help with this.

Yes, the scenarios are distinct, and they concisely solve the issue. Scenario one solves the end-user who uses Student Life as a study tool and Scenario Two solves the problem from the teacher's perspective as a learning tool. These two scenarios are distinct.

4. Are the features stated clearly? A feature should be a statement of a piece of functionality (users can create accounts, users can geotag photos taken in the application, users can check their itinerary, users can sort their messages). Are enough features provided? Remember, 3 is a minimum, you most likely will have more (we just need to start somewhere).

Yes there are three features, but they are not clear in what they do or how they work. They seem to be generic ideas of what they want the product to do but they don't specifically describe a way it will be accomplished.

5. User stories and tasks may or may not be provided on Confluence, so you do not need to evaluate these.

User stories and tasks are provided on Confluence with descriptions of the scenarios/features.

6. Are the configuration management rules given clearly? Has a specific workflow been selected? Have rules around commits and commit messages been provided?

The configuration management rules are given, the workflow they are using is based on features branches. However, the rules around commits and commit messages are vague. I believe there needs to be more elaboration on how the sprints will effect how often the team will review each feature before commits. Also elaborate on commit messaging conventions.

7. Is the technology stack clearly defined and explained?

Yes. the team says what IDE is being used (Android Studio), what language is being used (Kotlin), the database they are using (SQLite/Room), and how they plan on testing (Firebase).

8. Has a linter been selected? Is it clear which one is being used? Is it clear how it has been configured?

Yes, they are using the standard Lint tool that is automatically provided with Android Studio.

9. Are the onboarding documents clear? Do you believe you could set up your own machine for the development of this project, based on the provided instructions?

The document only mentions to clone the repository so there are not sufficient steps to properly onboard a person who may have not the context of ideas regarding the project.

10. Is it clear which parts of the system will use automated testing, and which parts will use manual testing? Are the testing tools identified?

They have clearly defined their testing rules and tools, JUnit and Firebase. They have not specified what code will be automatically tested and which code will be manually tested.

11. Have the rules for how testing and commits interact been defined?
The rules for testing have been defined as pretesting code before merging to main branch.

W4: Architecture and Design Materials Checkpoint

Architecture

High-level Architecture: https://confluence.cs.ecu.edu/x/lwAXAg

Architectural Style: https://confluence.cs.ecu.edu/x/fgAXAg

Logical Components: https://confluence.cs.ecu.edu/x/gwAXAg

Design

Important classes, functions, and files: https://confluence.cs.ecu.edu/x/hgAXAg

Is your application using a database? **No**

Non-obvious design decisions: https://confluence.cs.ecu.edu/x/iQAXAg

W4: Architecture Materials

Architectural Style

Layered Architecture

Components within this layered architecture pattern are organized into horizontal layers, each layer performing a specific role within the applications.

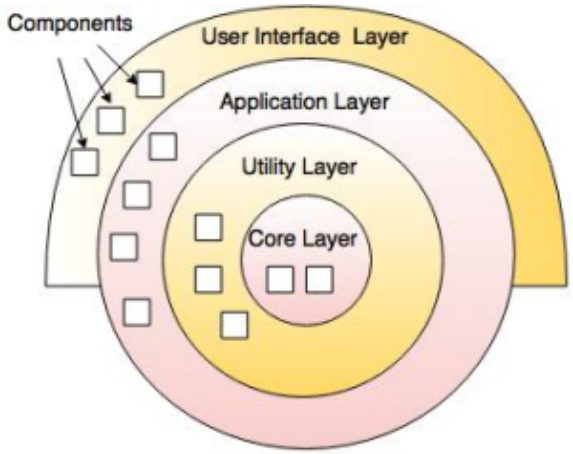


Fig.- Layered Architecture

Description

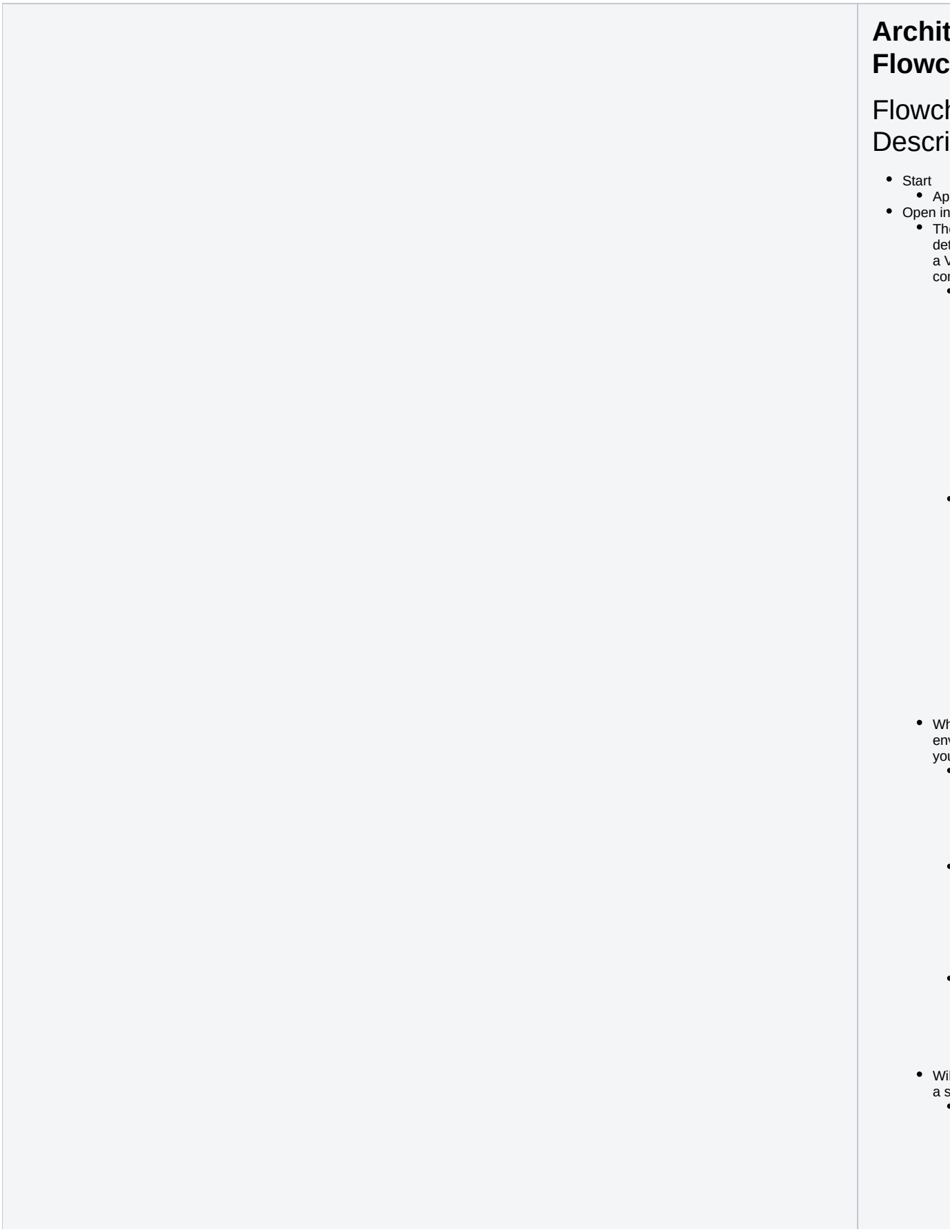
Core Layer: Each environment display

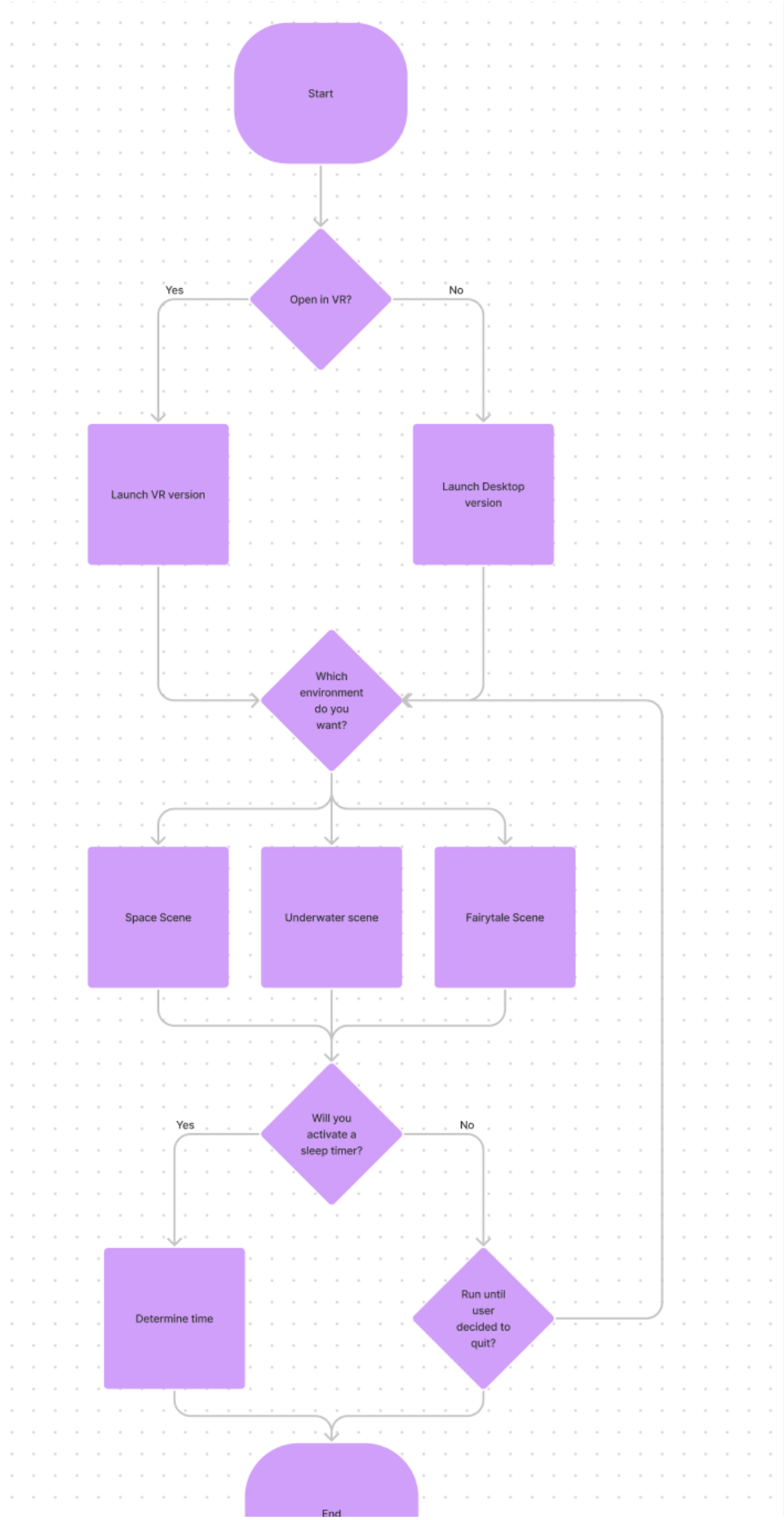
Utility Layer: VR implementation

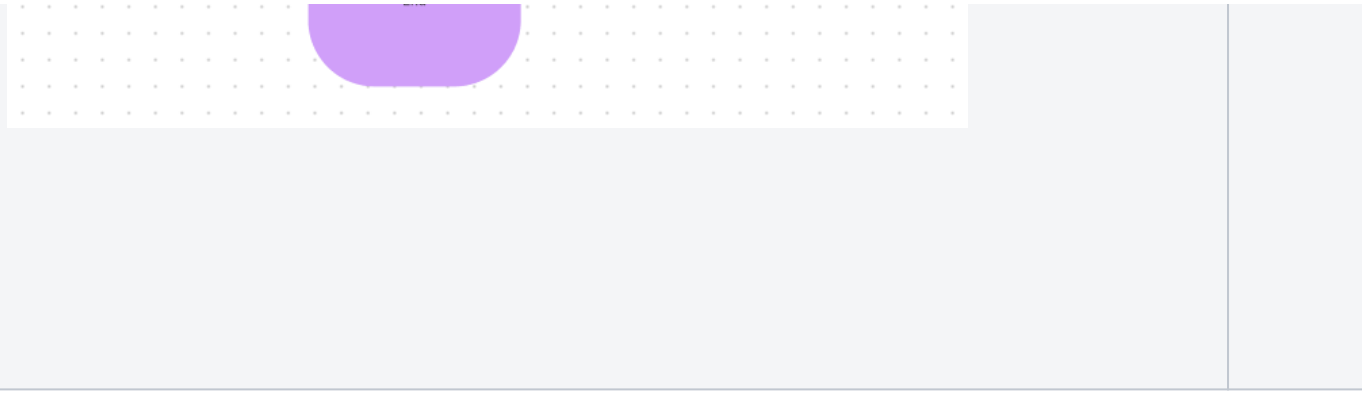
Application Layer: Main program that runs each of the environment

User Interface Layer: Screens, Menus

High-Level Architecture




Start
Open in VR?
Yes
No
Launch VR version
Launch Desktop version
Which environment do you want?
Space Scene
Underwater scene
Fairytale Scene
Will you activate a sleep timer?
Yes
No
Determine time
Run until user decided to quit?
End



Logical Components

FMOD Studio and Assets Package:

FMOD Studio Unity Integration exposes the full FMOD Core and Studio API's in C#. FMOD is a powerful audio system solution that can be integrated into Unity and the accompanying scenes. The default Unity audio system does not support adaptive audio level 2 solutions, which means dynamic sounds and real-time interaction cannot be implemented. Therefore, FMOD replaces the existing Unity audio system and comes packaged as an integration asset. However, this means FMOD assets need to be within each working scene and is unique to each scene. There will have to be 4 audio components (including the menu) that handle each scene's environment sounds effects and music in real-time. Also the settings menu will need to have access to the FMOD library for manipulating sounds, but it isn't an issue since FMOD is recognized as the default audio system.

W4: Design Materials

Classes, Functions, and Files

Base Unity systems:

Mono Behavior Class: These are functions that are handled by the Unity game engine. All other classes extend this class.

Update Function: This function is called every frame by Unity and much of the game functions run within this.

Awake Function: This function is called before all other methods when a scene is loaded, it is used to create object pools and other initialization tasks.

Start Function: This function is called at the start of each scene just after the awake method and performs a similar purpose to Awake.

Space Environment Functions and classes:

Arc Class: Handles calculating the distance moved between where a meteor spawns and its final position. Calculation runs inside a Unity coroutine to move only a small amount per frame for optimized performance.

Game Controller Class: Handles creating a pool of meteors to be used by the Arc class. Object pooling is used instead of run time instantiation to aid in performance.

Orbit Class: Controls the movement of the planets around the sun.

Rotate Function: Handles the position calculations of the Orbit Class.

Rotate Planet Class: Rotates a planet around its internal pole.

Fairy Forest Environment Functions:

Antialiasing Editor Class: Handles Customization of antialiasing settings of the environment.

Bloom And Flares Editor Class : Handles customization of Bloom settings of the environment.

Bloom Editor Class: Adds additional settings to the bloom Editor.

Camera Motion Blur Editor: Handles customization of camera bloom settings of the environment.

Color Correction Curves editor: Handles correcting the colors of curved textures.

Color Correction Lookup Editor: Applies settings from Color Correction curves editor.

Crease Shading Editor: Allows editing of the line shading in the environment.

Depth of Field Editor: Allows for the editing of depth of field of the camera.

Edge Detection Editor: Allows for more options for depth of field.

Noise and Grain Editor: Allows for more noise and grain option customization.

Sun Shafts Editor: Allows for more customization of sun shafts in the environment.

Tone mapping Editor: Allows for more customization of tone maps.

Vignette and Chromatic Editor: Allows for vignette customization.

Underwater Environment Functions:

Camera Move Class: Tester Class which allows the camera to move during runtime.

Fog Class: Creates the fog effect to be rendered over the camera.

Player Move Class: Allows player to move around environment using keyboard inputs.

UW Effect Class: Creates the underwater distortion effect to be rendered over the camera.

Main Menu Functions:

Button Handler Class: Handles the button inputs, settings and environment selection during runtime.

FMOD Audio API and defined namespaces:

The Studio API: used to playback and control events created in the FMOD Studio tool.

The Core API: for playing back sounds directly as well as advanced control of event playback.

The Runtime Manager API: provides helpers for the most common operations in writing Unity game components that use FMOD Studio Events.

Classes:

- Event Reference: Holds a reference to an FMOD Studio event.
- FMOD Platform: Platforms used by the FMOD Integration.
- Emitter Game Event: The Unity Events used by the Studio Event Emitter, Global Parameter Trigger and Parameter Trigger components.
- Loader Game Event: The Unity Events used by the Bank Loader component.
- Emitter Ref: Class used by the Studio Parameter Trigger for Studio Event Emitter and Parameter info.
- Param Ref: Class used to store important Parameter information.
- Import Type: Ways to import Banks into the project.
- Bank Load Type: Automatic Bank loading options used by the Runtime Manger.
- Meter Channel Ordering Type: Corresponds to the Metering Channel Order in FMOD Studio.

The "fspro" files are native FMOD studio application project files for reading audio data into the Event Editor.

The ".bank" files contains all audio, scripts, and event data into a single encoded file that Unity can read via the FMOD integration package assets within each scene.

Non-Obvious Design Decisions

- **Performance Priority**
  - We have to put a massive importance on performance (best measured in framerate at the moment). The reason this is so important, and what is not obvious, is that since we are focusing on developing for VR Headsets, our final project will always be rendered twice. This essentially cuts down our framerate in half, since we are not testing with the Vive quite yet. In order to not have to do a huge amount of changes once we start testing our final product, we must keep performance in mind early in the project.

- **Two modes of interacting**
  - Since we are building a VR game, we need to remember that the UI will be controlled differently. Early on we decided that a "Look-based" system (where looking at a button for X amount of time will select said button) would be ideal for a relaxation app, as to not require handling controllers. We have to keep this interaction system in mind while designing UI elements, since we want both a mouse, and person (via looking only) to interact smoothly and accurately with our UI.

- **Audio splicing and configuration**
  - Audio, including music and environment sound effects, all have to be layered and spliced together in-order to present a professional, realistic, and surround VR experience. Essentially each soundtrack is a compilation of multiple audio slices of different sound effects that blend together to form an assemble of music. However, each individual sound can be controlled and dynamically morph into unique effects according to user interaction or choice of settings.
  - There will not be a conventional soundtrack for the VR scenes, so streaming audio/music files using the pooling system is currently being used.

W6: Testing Materials Checkpoint

Manual Testing

You need to provide the manual testing scripts that you are using for manual testing. For this report, at least five need to be provided. Each should include a link to the actual script. You should create as many as are actually needed to test your system.

The Project uses manual testing: **Yes**

Manual Testing Overview: https://confluence.cs.ecu.edu/x/DQ0XAg

Script 1: https://confluence.cs.ecu.edu/x/FA0XAg

Script 2: https://confluence.cs.ecu.edu/x/Gw0XAg

Script 3: https://confluence.cs.ecu.edu/x/Hg0XAg

Script 4: https://confluence.cs.ecu.edu/x/IA0XAg

Script 5: https://confluence.cs.ecu.edu/x/Ig0XAg

Automated Testing

You need to document your automated testing approach. This should be done on a single page which describes your automated testing approach. Details of what should be on this page are found in the assignment.

Automated Testing Overview: https://confluence.cs.ecu.edu/x/Dw0XAg

Traceability

You need to include traceability information. This should be given on a Confluence page as an overview, which can then be used to, e.g., link to specific tests or specific requirements (you can embed Jira information into Confluence if needed).

Traceability: https://confluence.cs.ecu.edu/x/EQ0XAg

Automated Testing

No automatic testing is available for Unity, so manual testing and internal class level checks will be used to test the software. An example of these internal class level tests is the “GetComponent” function that will need to be checked to prevent null references which are a common source of errors.

Manual Testing Overview

You need to provide the manual testing scripts that you are using for manual testing. For this report, at least five need to be provided. Each should include a link to the actual script. You should create as many as are actually needed to test your system.

The project uses manual testing: **Yes**

Script 1 - Main Menu

The manual testing for Main menu is making sure each button works and is linked to the proper scene once selected.

When testing the Quit button on the main menu, once the "Game" is fully built the application will exited.

For the time being, using Debug.log is being used to show that the Quit function is working.

Script 2 - Scene Selection

Manual Scene Selection is tested within the Main scene.

Using Unity's built in SceneManager, we are able to test each scene by clicking on the scene we want to load.

Once the button is clicked, the new scene is loaded, and the main scene is paused.

Script 3 - Space Scene

Space Scene Manual Testing Protocol:

1. Script based object references are done using TryGetComponent rather than GetComponent so that if A NULL reference occurs the error is handled and doesn't crash the whole program.
2. Environment testing is done by manually going through the environment wearing the VR headset and rotating your head to check all angles for environment clipping.
3. Ray casts are tested using Gizmo's, a Unity feature that allows for visualization of the ray cast.

Script 4 - Fairy Tale Scene

Fairy Tale Scene Manual Testing Protocol:

1. Script based object references are done using TryGetComponent rather than GetComponent so that if A NULL reference occurs the error is handled and doesn't crash the whole program.
2. Environment testing will be done by manually going through the environment wearing the VR headset and rotating your head to check all angles for environment clipping.
3. Ray casts are tested using Gizmo's, a Unity feature that allows for visualization of the ray cast.

Script 5 - Underwater Scene

Underwater Scene Manual Testing Protocol:

1. Script based object references are done using TryGetComponent rather than GetComponent so that if A NULL reference occurs the error is handled and doesn't crash the whole program.
2. Environment testing will be done by manually going through the environment wearing the VR headset and rotating your head to check all angles for environment clipping.
3. Ray casts are tested using Gizmo's, a Unity feature that allows for visualization of the ray cast.

Traceability

Manual testing traceability.

Start up TranquilVR executable.

Enter main menu.

Script 1: https://confluence.cs.ecu.edu/x/FA0XAg

Button I/O and selection are introduced. They accompany nested links for each 3D scene when selected.

Approach each option as a test case, whether there is any input lag or unintentional outcome (i.e. quit button does not exit game immediately).

Enter screen selection.

Script 2: https://confluence.cs.ecu.edu/x/Gw0XAg

Using Unity's built in SceneManager, test each scene by clicking on the visual representation of the scene (screen) to load into.

Make sure each screen correlates correctly to the proper scene and there is no unintentional outcomes (i.e. screen glitches or bug with scene loading).

Enter space scene.

Script 3: https://confluence.cs.ecu.edu/x/Hg0XAg

Script based object references are done using a particular error-handling function within Unity (i.e. TryGetComponent, Gizmo's).

Have this scene guarantee to fully load with minimal to no errors in object references, prefabs, plugins, and/or other Unity related files. Utilize the stack trace to find what would stop or crash the game.

Enter fairy tale scene.

Script 4: https://confluence.cs.ecu.edu/x/IA0XAg

Test case script based object references just like the previous scene.

Have this scene guarantee to fully load with minimal to no errors in object references, prefabs, plugins, and/or other Unity related files. Utilize the stack trace to find what would stop or crash the game.

Enter underwater scene.

Script 5: https://confluence.cs.ecu.edu/x/Ig0XAg

Test case script based object references just like the previous scene.

Have this scene guarantee to fully load with minimal to no errors in object references, prefabs, plugins, and/or other Unity related files. Utilize the stack trace to find what would stop or crash the game.

W5: Architecture and Design Review - Student Life

1. **Is the high-level architecture of the system clear and easily understood? Is a diagram provided? Is there an explanation included, or just the diagram?**
    a. The high-level architecture of this system was clear and easily understood but it didn't resemble architecture. The diagram resembles design more than an architectural style. There is an explanation included but it resembles what the diagram includes that doesn't relate. The information they provided explains how the system communicates locally and uses a log-in system to save the information but doesn't explain architecture.
2. **Are the architectural styles used by the system provided? Are there individual pages available providing more details for each listed style, along with a summary page? If a diagram is provided for a style (not required, but recommended if it helps with the explanation), is it useful? Is the textual description clear and easily understood?**
    a. Yes, the architectural styles were used by the system. There are no pages providing more details since there was only one architectural style stated as being used. There is a diagram added to help describe the architecture style, and the diagram is useful, especially as a reminder on how the Model, View, Controller architecture style works. The textual description appropriately relates the actual system used to the diagram provided, and it was easily understood.
3. **Are the logical components in the system clearly listed? Is it clear what each component is/does? (Note: Explanations were not mandated, but should be provided if it is not clear what a component is for – best practice is to provide an explanation regardless so later readers do not have to guess.)**
    a. The logical components are clearly listed on the page. Each description of their components is well documented and perhaps there can be room for more details if needed.
4. **Are the important classes, functions, and files in the code clearly listed? Are their relationships clear? If diagrams are provided, are these easy to understand and well explained? If design patterns are mentioned, is it clear which patterns are being used, and how?**
    a. Yes, the classes and functions are listed clearly with a small description on what each one is responsible for/what they do. There are no diagrams or design patterns related to classes, functions, or files.
5. **If a database is being used, is the layout of the database clear? For a relational database, this means that a clear schema has been provided. For a NoSQL database (e.g., Firebase, MongoDB), this means that a description of the stored data (e.g., stored JSON documents) is provided and clearly explained.**
    a. Their database is clearly explained and the proper diagram is used to show the relations between the data. The only concern is the primary key is not bolded for clarity.
6. **Are non-obvious design decisions explained clearly? Does the description help to explain what decisions were made, and why?**
    a. Since the team did not define any non-obvious design decisions, their explanation of why this is the case was transparent. They defined that the app links each user to their specified quiz and that there is a database that holds the questions that they will be presented. In addition to the database of questions being held, the questions answered will also be held so that the user may review what questions they had gotten wrong or topics they had troubles with.

W7: Getting Started Response

Question 1 Review: The team charter from Relaxation and Sleep Aid is complete. All of their responses to the items are clear.

> **Response:** No response; we appreciate them thoroughly looking into our responses.

Question 2 Review: The details listed for each of the personas are stated very clearly. All of the standard information is provided and this includes the persona's name, details of their jobs, details about their education/ experience, and the individual's interest in the product. The personas given are very distinct from one another

> **Response:** No response; we appreciate them thoroughly looking into our responses.

Question 3 Review: The details of the scenarios are stated clearly and are differentiated by the persona involved in the scenario. The overall objective of each scenario is clearly stated. One persona is attempting to alter their sleep schedule and the other persona is trying to prevent a panic attack. These two scenarios do not have any significant overlap.

> **Response:** No response; we appreciate them thoroughly looking into our responses.

Question 4 Review: The features are stated however are not stated in the format stated above. Features are organized as bullet points; however, there are more than 3 features listed.

> **Response:** No response; we appreciate them thoroughly looking into our responses and them stating that our features list was formated in bullets instead of prompted but all information was included otherwise.

Question 5 Review: Two user stories and tasks are provided that describe a problem and feature implementation to solve that problem.

> **Response:** No response; we appreciate them thoroughly looking into our responses.

Question 6 Review: Yes. The main branch contains shared resources and each scene is contained within its own branch. Before making a commit, the code must pass all tests and a brief description of changes made must be provided.

> **Response:** No response; we appreciate them thoroughly looking into our responses

Question 7 Review: Yes, they have a list of their technology stack. They are using Unity, Visual Studio, and HTC Vive VR Headset.

> **Response:** No response; we appreciate them thoroughly looking into our responses.

Question 8 Review: They are using a built-in linter that Visual Studios provides. They did not provide what it is specifically called, aka IntelliSense. Yes, it is clear how it is configured.

> **Response:** No response; we appreciate them thoroughly looking into our responses. Yes, we are using IntelliSense (a code completion tool that is built into Microsoft Visual Studio).

Question 9 Review: Yes their onboarding is clear. To set it up all you need to do is have unity hub/unity downloaded onto your machine and the organization leader will need to give you a seat and then you will have access to the unity files.

> **Response:** No response; we appreciate them thoroughly looking into our responses.

Question 10 Review: All of the testing mentioned does not seem to be true automated testing; however, the document does provide explanations as to why this is the case. The testing mentioned seems to be more so methods of debugging. No external testing tools are mentioned.

> **Response:** No response; we appreciate them thoroughly looking into our responses.

Question 11 Review: It is clear that all code should be tested manually before committing and pushing changes to the main branch to avoid large issues. The ways in which code will be tested is not made very clear in the previous section. What is defined as "working" code? Is working code simply code which doesn't produce compiler errors.

> **Response:** We defined working code as adding features without breaking or modifying other features of the code. For example one of the criteria that is checked is missing or broken references in the scene. Checking for compiler errors is also a prerequisite before committing to the git.

W8: Architecture and Design Revision and Response

1. **Is the high-level architecture of the system clear and easily understood? Is a diagram provided? Is there an explanation included, or just the diagram?**
    a. Review: There is a diagram provided which is easily understood and has a well-detailed explanation with it. The diagram however is more so the design part.
    b. Professor Review: This is a useful diagram. However, this is not really an architecture, and it is more a design part. It would be more appropriate for the architecture to describe different moving parts and their interaction.
    c. Our Response: Our group was a bit confused about this part of the assignment. Our project does not have much when it comes to "moving parts" since Unity takes care of most of the needed components for us. We also thought that the diagram showed the way that the whole project was "moving" not only through code but design as well.
2. **Are the architectural styles used by the system provided? Are there individual pages available providing more details for each listed style, along with a summary page? If a diagram is provided for a style (not required, but recommended if it helps with the explanation), is it useful? Is the textual description clear and easily understood?**
    a. Review: The architectural styles used by the system are provided. There is a page made for the one Architectural style they listed along with its summary. They included a diagram for the layered style which is useful for the reader and it has a clear description that is easily understood.
    b. Professor Review: The styles are good, and the diagram is nice.
    c. Our Response: Thank you for the diagram compliment.
3. **Are the logical components in the system clearly listed? Is it clear what each component is/does? (Note: Explanations were not mandated, but should be provided if it is not clear what a component is for – best practice is to provide an explanation regardless so later readers do not have to guess.)**
    a. Review: The logical components in the system are clearly listed and it is clear what each component does.
    b. Professor Review: You have a good description of the component. However, the logical component is more about how you are dividing your code into different components.
    c. Our Response: We appreciate the group comments on our logical components. Based on the information the professor has responded the logical components included in our development would be the separation of scripts for different elements in our environments like our boids, menu selection scenes, sound elements, partial systems, and VR integration software.
4. **Are the important classes, functions, and files in the code clearly listed? Are their relationships clear? If diagrams are provided, are these easy to understand and well explained? If design patterns are mentioned, is it clear which patterns are being used, and how?**
    a. Review: The important classes, functions, and files in the code are clearly listed and their relationships are clear. There are no diagrams listed. Design Patterns are not mentioned.
    b. Professor Review: Very detailed and good description.
    c. Our Response: Our group was not sure how to show a diagram for this section, so we decided to go into more detail on the descriptions.
5. **If a database is being used, is the layout of the database clear? For a relational database, this means that a clear schema has been provided. For a NoSQL database (e.g., Firebase, MongoDB), this means that a description of the stored data (e.g., stored JSON documents) is provided and clearly explained.**
    a. Review: Databases are not being used.
    b. Professor Review: N/A
    c. Our Response: Our project does not use any type of database.
6. **Are non-obvious design decisions explained clearly? Does the description help to explain what decisions were made, and why?**
    a. Response: The non-obvious design decisions are explained clearly, and the descriptions provided explain why the decisions were made and why.
    b. Professor: Good description
    c. Our Response: The non-obvious design decisions were fairly simple for us to figure out.

W9: Project Wrap-Up

Non-Functional Requirements

You should define the non-functional requirements for your system if you have not yet done so. These are aspects such as performance, reliability, legal constraints, or platform constraints that do not relate to specific functionality but instead relate to overall aspects of, or constraints on, the system. For measurable non-functional requirements, a metric should be included: performance should not be "fast", but "a page should load in under 100ms", for instance. These should be specified as a single page on Confluence.

Licensing

If you have not done so already, you should pick a license for your application. If you opt for an open-source license, you should add a LICENSE file to your repository. Guidelines for helping to select a license can be found here: https://choosealicense.com/. If you opt for a closed license, add a Confluence page describing how you would plan to license the software in the future.

High-Quality Commits

For each team member that is claiming to have made at least 5 high-quality commits, indicate, first, the name of the team member, then second, provide links to 5 high-quality commits by that member on GitLab. Note that this means different team members could get different scores for the configuration management portion of the project evaluation.

Testing Quality

You should identify 5 high-quality automated tests. These are well-written tests that focus on specific parts of the application. If you used manual testing, you should identify 5 high-quality manual test scripts. These are clear scripts that include the specific steps and data that the tester would need to ensure the test is repeatable and that it runs successfully.

High-Quality Commits

License

Non Functional Requirements

--

Testing Quality