CMPE 492
Senior Project 2

Low Level Design Report

“VR PROJECT BLUE”

Project Supervisor

Dr. Özlem Albayrak

Project Team Members

Sancaralp Elmas

Hasan Yasin Yıldız

Uğur Evren Çamalan

Eren Çağlaroğlu

Project Jury Members

Dr. Ulaş Güleç

Dr. Elif Kurtaran Özbudak

Course Coordinator

Dr. Fırat Akba

Project Website

teduvrsp.github.io

Table of Content

1 Introduction

1.1 Object design trade-offs

1.1.1 User-Friendliness vs Game difficulty:

Since VR Project Blue is a serious game that aims to teach while entertaining, it does not contain any elements that will challenge the user. In addition, since any excessive difficulty in the game will alienate the player from the game, we will design the difficulty level to make the user feels as comfortable as possible. In addition, in a serious game, there will be no difficulty that will take the game away from reality, as excessive difficulty levels will take the game away from reality.

1.1.2 Visual and Audio Richness vs Performance:

VR Project Blue will have a variety of visual and audio effects. With this diversity, we aim for players to enjoy the game more. The objects or auditory elements we will use in the game will draw the player's sensory organs more into the game and increase the pleasure the players will get from the game. Additionally, we aim to make the game world look deeper thanks to the lighting and shading effects we will use in the game. However, these rich contents may negatively affect performance in some cases. That's why we make sure that the effects we use do not have any negative effects on the performance of the game. For example, we are careful not to use any visual or audio effects that will seriously reduce the Frame Rate in the game.

1.1.3 Guidance and Tips vs Free Exploration:

Since VR project blue aims to educate its users and raise awareness in the field of cyber security, there will be plenty of guidance and tips for the game. The game will include free exploration, but the player will be able to progress through the game with more clues. For example, the player will perform infiltration or password cracking operations in the game with the help of clues. Additionally, thanks to the hints, the player will enjoy the game more. Because he/she will make some kind of observation while trying to find these clues.

1.1.4 Easy Accessibility vs Game Mechanics Complexity:

VR Project Blue will be as mechanically less complex as possible to make the game quickly understandable and playable by any player. Because the complexity of the game mechanics makes it difficult to learn and can alienate players from the game. In addition, the mechanics we will use in the game will be designed at a level that can be easily done by every player. In this way, we aim to increase the ease of playability of the game by everyone. For example, the mechanics we will use in the game will be simple mechanics such as holding and dragging.

1.2 Interface documentation guidelines

For internal packages:

Class Name, implements I list
vis attribute: type
vis operation (arg list): return type

Where:

- vis = visibility (‘+’ for public, ‘#’ for protected, ‘-‘ for private, ‘,’ for separating get, set)
- attribute = data member aka field aka instance
- operation = method

We use this template for our interfaces, other than that Underlines means that attribute or operation is static. We use Pascal naming method for all our code. We explain functionalities of everything under the table title by title.

For external packages:

Description

URL: docs.unity3d.com/…

URL is the url to the documentation of that specific class.

We use this template for external packages since there are too complex to add by all their attributes and operations.

1.3 Engineering Standards

I'm writing to offer a set of suggested guidelines for creating cybersecurity games for virtual reality (VR). It is important to adhere to these standards and norms to improve the gaming experience, teach cybersecurity skills, and guarantee game security. By incorporating these principles, cybersecurity games developed in a virtual reality setting will be developed to a better standard of quality and believability.

1.3.1 VR Development Standards:

Robots and robotic devices - Safety criteria for personal care robots as outlined in ISO 13482:2014: Guidelines for the safe design and use of robotic devices in personal care applications are provided by this standard. Although it has nothing to do with cybersecurity games specifically, it does provide insight into important safety aspects for virtual reality environments.

1.3.2 Standards for Game Design and Development:

Information technology - Security methods - Information security management systems - ISO/IEC 27001:2013 - Requirements: This standard, which is very pertinent to the creation of cybersecurity games, lays out the conditions for setting up, putting into practice, maintaining, and continuously enhancing an information security management system.

1.3.3 Standards Particular to VR:

IEEE Recommended Practice for Virtual Reality (VR) - Quality Assurance, IEEE 3079-2020: To guarantee a high degree of quality and performance in your cybersecurity game, this article offers guidelines on quality assurance and testing techniques for VR apps.

It is important to note that although these guidelines do not pertain specifically to virtual reality cybersecurity games, they do include several topics that are pertinent to developing a safe, instructive, and entertaining gaming environment. Respecting these guidelines will improve the project's quality and encourage greater credibility and trust from your users.

1.4 Glossary; definitions, acronyms and abbreviations

VR : Virtual Reality

Player : User of video game software

vis : Visibility (‘+’ for public, ‘#’ for protected, ‘-‘ for private, ‘,’ for separating get, set)

Attribute : Data member aka field aka instance

Operation : Method on script

URL : Uniform Resource Locator. A URL is nothing more than the address of a given unique resource on the Web.

IEEE : Institute of Electrical and Electronics Engineers - Professional organizations company

ISO: International Organization for Standardization

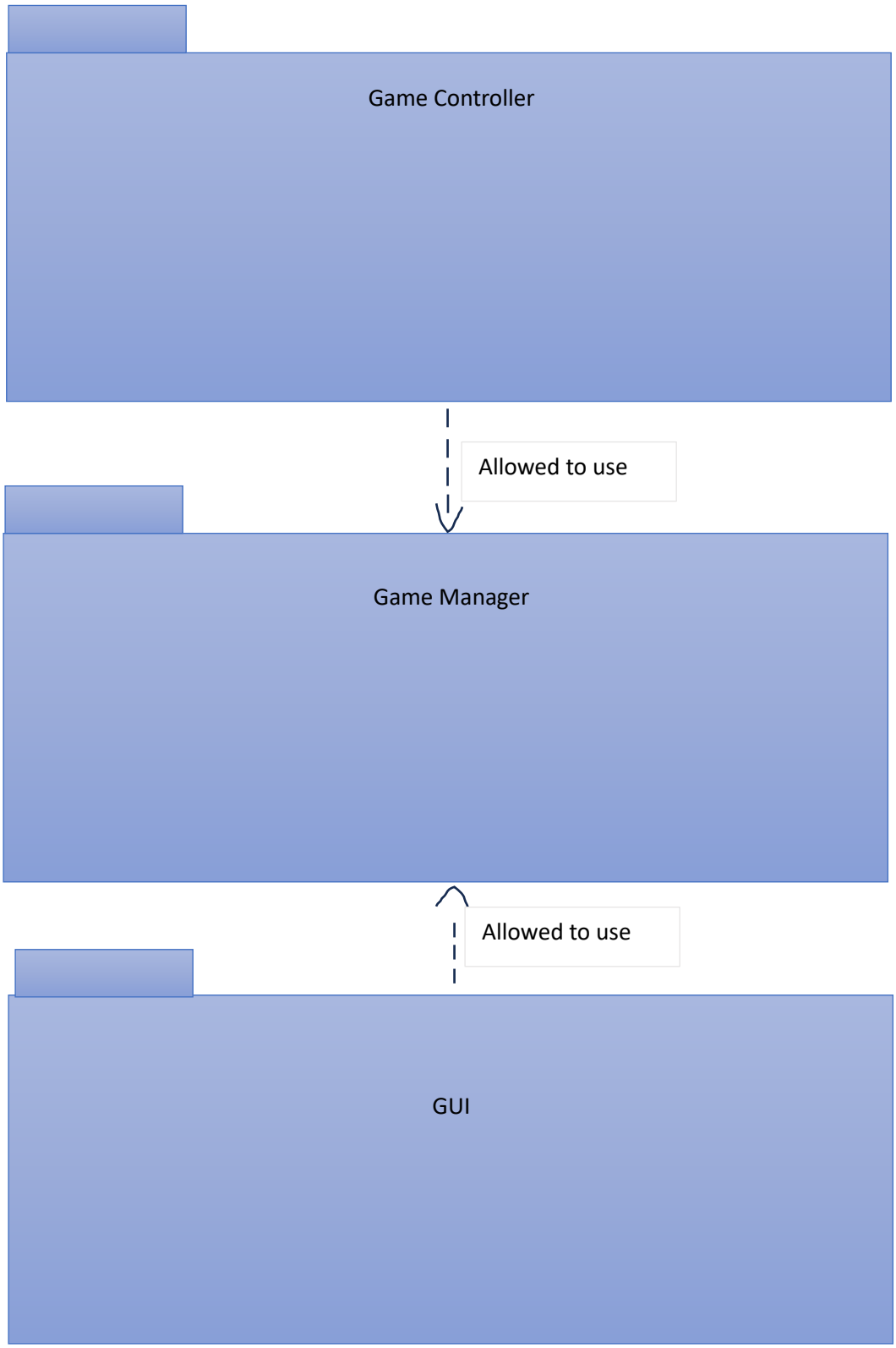
IEC : International Electrotechnical Commission

GUI : Graphical User Interface, layer that enables a person to communicate with a computer through the use of visuals.

2. Packages

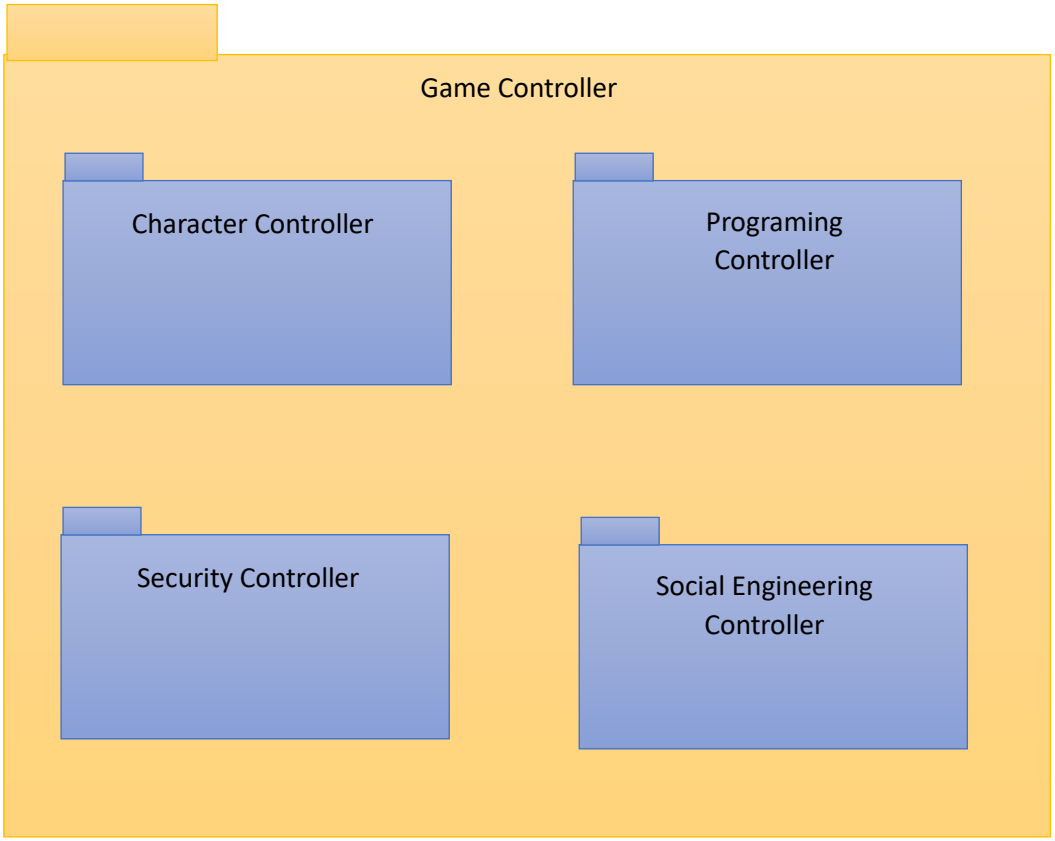
This section presents the packages of our project. Project Blue has three packages which are

Our project has 3 packages which are GUI, Game Controller and lastly Game Manager.



GUI and Game Controller are the top levels of our packages and these provides interaction between user and game. Game Controller's main goal is taking input from user and allow to control mechanics, essentials and progress of game according to input, preferences or progress. GUI's main goal is showing the desires of user like options, interactable objects and much more. Also GUI is a way to interact some mechanics with visuals.

2.1 Game Controller



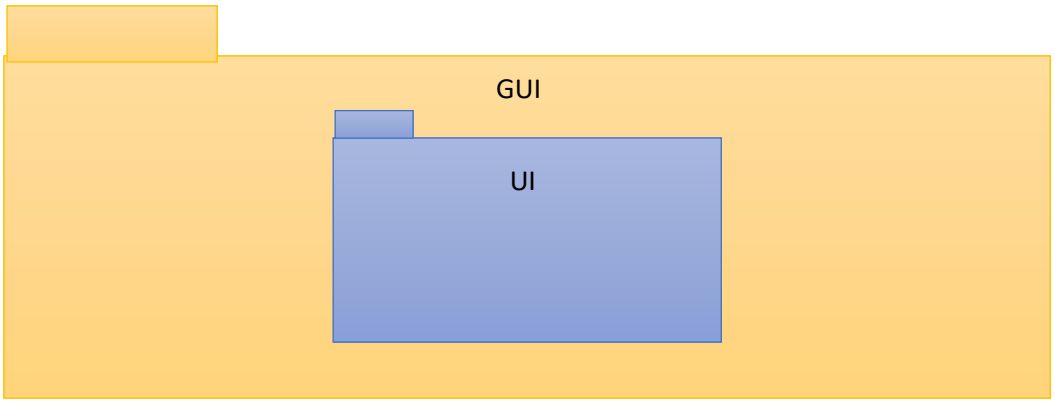
Character Controller is for control the behavoiur of character.

Security Controller is for the control NPC's in game.

Programing Controller is fort he control programing mechanics in game.

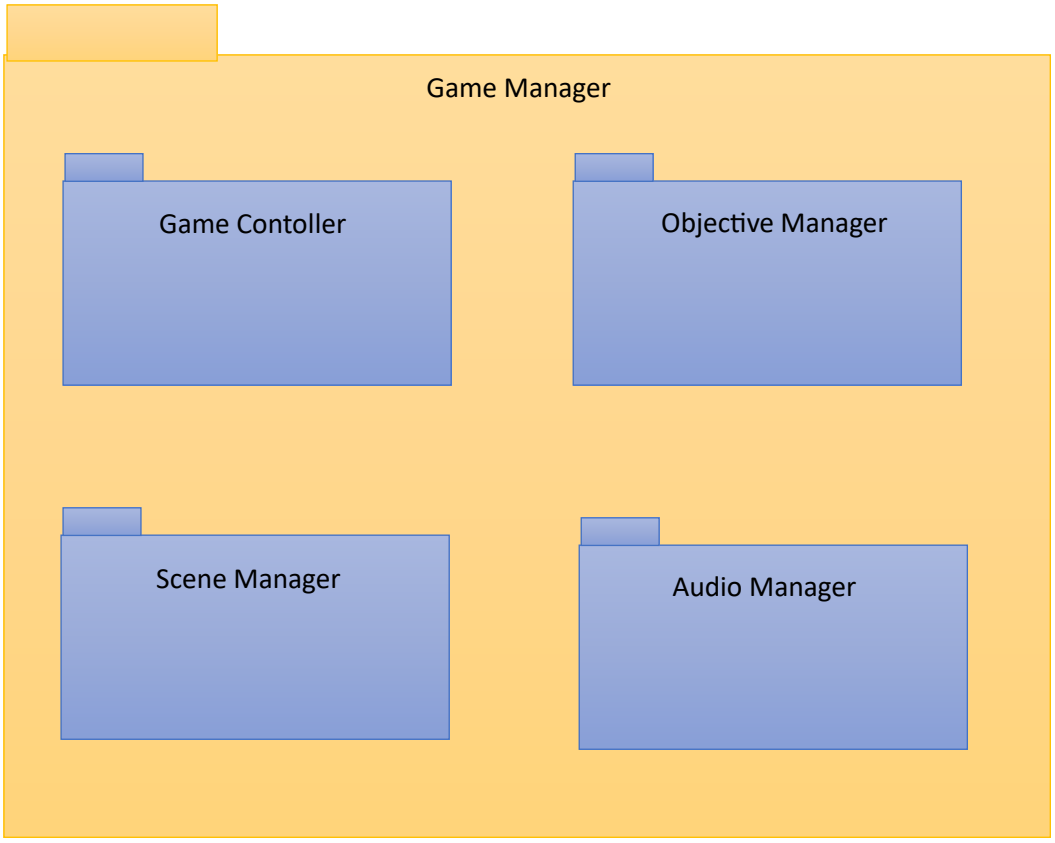
Social Engineering Controller is fort he control objectives and progress in social engineering gameplay.

2.2 GUI



UI is for the controle user interface, visuals and their access to the game manager.

2.3 Game Manager



Game Controller is the class for control progress in game.

Objective manager is for the control requirements of objectives and progress in missions.

Scene Manager is controlling class for scene which is currently playing.

Audio Manager is for control audios in game such as sound effects and musics.

3 Class Interfaces

3.1 Scenario Control System:

IStep
+ StartStep(): void + CompleteStep(): void # BindOnStepStart(UnityAction): void # BindOnStepComplete(UnityAction): void # Initialize(): void

IStep: This interface contains the common operations that must be implemented for any step in our scenario control system.

StartStep: This method must be called when the step is started.

CompleteStep: This method must be called when the step is completed.

BindOnStepStart: This method must used to bind the parameter actions to the implemented events.

BindOnStepComplete: This method must used to bind the parameter actions to the implemented events.

Initialize: This method must be used to set all the variables according to the step and call the bind functions.

StepBase, implements IStep
OnStepStart: UnityEvent # OnStepComplete: UnityEvent + stepIndex: int + isActive: bool - _isComplete: bool
Implemented from IStep

StepBase: This is the base step class where that all the generic steps will be derived from.

OnStepStart: This variable should keep the external events that shall be activated when the step is started.

OnStepComplete: This variable should keep the external events that shall be activated when the step is completed.

IsActive: this should return either the step is the active step or not.

IsComplete: this should return either the step is completed or not.

StepManager
+,- Instance : StepManager +,- int CurrentStepIndex: int +,- int TotalSteps: int
+NextStep(): void -Unity Event Functions

StepManager: This class contains the necessary functions to Initialize and Handle step changes

Instance: Since this is a manager it is built with singleton pattern and could be reached wherever it's necessary.

CurrentStepIndex: index of the active step

TotalSteps: count of the total steps in the current scenario

NextStep: Sets the current step to the next

3.2 XR Controller

Uses actions from the input system to interpret feature values on a tracked input controller device into XR Interaction states, such Select. It also applies to the transform of the GameObject the current Pose value of a tracked device.

There are analogous Select Action Value, Activate Action Value, and UI Press Action Value values for the Select Action, Activate Action, and UI Press Action properties. These activities have different action types: optional value type actions for the latter and button type actions for the former. In order to determine whether the select interaction state is active, the component reads whether the choose action is done per frame. It also records the float value from the select action value. The float value will be read from the Select Action in the event that the Select Action Value is not set. The Activate and UI Press actions go through this procedure once more.

For proporties and descriptions:

https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@2.0/manual/xr-controller-action-based.html

3.3 Input Action Manager

All of the inputs of type InputAction in a list of assets of type InputActionAsset are automatically enabled or disabled by this component.

For proporties and descriptions:

https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@2.0/manual/input-action-manager.html

3.4 XRInteractionManager

In a scenario, the Interaction Manager serves as a middleman between the interactables and the actors.

Several Interaction Managers are feasible, and each can have a legitimate collection of Interactors and Interactables.

Both interactables and interactactors register themselves with a legitimate interaction manager in the scene when they become awake (if that particular manager hasn't previously been allocated by the inspector). In order for Interactors and Interactables to communicate, each scene needs to have at least one Interaction Manager.

For proporties and descriptions:

https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@0.0/api/UnityEngine.XR.Interaction.Toolkit.XRInteractionManager.html

3.5 XR Origin

In an XR scenario, the XR Origin stands for the world's center. To transfer trackable features and objects to their final scale, orientation, and position inside the Unity scene is the aim of the XR Origin. It gives details on a Camera, a Floor Offset Object, and an Origin.

For proporties and descriptions:

https://docs.unity3d.com/Packages/com.unity.xr.core-utils@2.0/manual/xr-origin.html

3.6 XR Interactor Line Visual

A LineRenderer and an Interactor are aligned by an Interactor helper object.

For proporties and descriptions:

https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@2.0/manual/xr-interactor-line-visual.html

4 References

Unity. (n.d.). *XR controller (action-based): XR Interaction Toolkit: 2.0.4*. XR Interaction Toolkit | 2.0.4. https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@2.0/manual/xr-controller-action-based.html

Unity. (n.d.). *Input action manager: XR interaction toolkit: 2.0.4*. XR Interaction Toolkit | 2.0.4. https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@2.0/manual/input-action-manager.html

Unity. (n.d.). *Class XRInteractionManager: XR interaction toolkit: 0.0.6-preview*. XR Interaction Toolkit | 0.0.6-preview. https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@0.0/api/UnityEngine.XR.Interaction.Toolkit.XRInteractionManager.html

Unity. (n.d.). *XR Interactor Line Visual: XR interaction toolkit: 2.0.4*. XR Interaction Toolkit | 2.0.4. https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@2.0/manual/xr-interactor-line-visual.html