Florida International University
School of Computing and Information Sciences

Software Engineering Focus

Final Deliverable

Project Title: AR-VR-VE for Computer Science Education 1.0

Team Members: Lukas Borges, Cristian Cabrera, Hamilton Chevez, Nicolette Celli, Francisco Lozada, Filip Klepsa, Kevin Delamo

Product Owner(s): Francisco Ortega

Mentor(s): Francisco Ortega

Instructor: Francisco Ortega & Masoud Sadjadi

Abstract

This document presents the information necessary to gain a good understanding of "AR-VR-VE for Computer Science Education 1.0" a research and implementation based project that is overseen by Dr. Francisco Ortega, and all its implemented user stories, necessary software and hardware resources for implementation, workflow, team organization, system design, design patterns, and validation. This document presents the information for the first release of the project. There were five separate teams in the design, implementation, and research process. CodeVR, ARCSE, CodeAventures, CircGR, and LEAP-Trainer. Each team has its own section in this document.

Table of Contents

INTRODUCTION

This section describes the status of the system for the AR-VR-VE project.

Current System

ARCSE is on its first iteration of development. Its starting point came from part from the limitation of no user interaction in the WEB-VR 1.0 project and previous research on knowledge retention as a result of user interactivity.

CodeAdventures is a continuation of a project from a previous semester. Our team started with a game with one level, and functioning forward and back modules. Many of the models in our current version are the same models from the previous version.

The CircGR old system (as of the beginning of this semester/iteration of the project) consisted of a C# implementation demonstrating the linearithmic algorithm for multi-touch detection with a high recognition accuracy (ACC = 99% and MCC = 0.95). The demonstrated early-detection of gestures (without the need for gesture completion) with ACC=99% and MCC = 86% for the first window (64 points) and demonstrated the need for only one user or developer-provided training sample per template. The new (current) system is the same as the old system except that it is written in C++.

The LEAP-Trainer is a new addition to the AR-VR-VE project, thus it relied on the use of research papers in obtaining a concrete starting point.

Designed in Unreal Engine 4, CodeVR only supports one language at the moment (Python), with the possibility to add more languages in the future. At the moment, CodeVR is capable of getting any valid python file (.py) and render python assignments, function definitions and function calls in VR, all of them with minimum text equivalence. CodeVR also has a physics system that allows users to pick up and maneuver these blocks in VR.

Purpose of New System

ARCSE aims at providing a set of learning activities in augmented and mixed reality. The primary objective of the project is to create user interaction via the features of the Vuforia API and the 3D models rendered with Unity. Vuforia allows the user to interact via virtual buttons, predefined areas on the surface area of an image target, which users can touch to create events. Through this feature, the project aims at creating activities that explore algorithms, logic gates & boolean algebra, data structures, and OS scheduling algorithms for the objective of facilitating richer learning experiences for students.

CodeAdventures is aimed at students who are intimidated by computer science concepts and deem the major too difficult. This project strives to attract new students to the computer science field by creating a simple and enjoyable introduction to programming. As it stood last semester, the game was too confusing and difficult to play, so our team has improved various aspects to make the game more accessible.

CircGR was created in order to provide gesture designers with a more cross platform API that has a higher performance than the old system. In addition the new system was intended to be more well-structured and maintainable. Although it would essentially be the same representation of the algorithm implemented in the old system it was meant to be an overall better transformation of the code in terms of language used and the implementing the best practice approaches that can be derived from translating a code base of over 3,000 of code written in C# to C++. This way the API would not be dependent of the .NET framework that targets the Windows operating system.

The LEAP-Trainer utilizes the technology of the Leap Motion Controller (LMC). A commercialized, palm-sized, infrared sensor designed to make capturing hand and finger movements in 3D space affordable. Applying machine learning methods, this sub-project hopes to extend the usefulness to, or re-validate use of the LMC device for single-frame gesture recording and recognition. As the backbone of the American deaf culture, gesture recognition on American Sign Language (ASL) could potentially become an educational tool for children and adults.

The purpose of CodeVR is to help students learn computer science by programming python language in VR. To do that, in future iterations CodeVR will not only render graphics from .py files, but also interpret graphics in the environment and generate or edit .py files accordingly. More languages should be added in the future.

USER STORIES

The following section provides the detailed user stories that were implemented in this first iteration of the AR-VR-VE project. These user stories served as the basis for the implementation of the project's features. This section also shows the user stories that are to be considered for future development.

Implemented User Stories

ARCSE TEAM:

Sprint 1

#685 Learn Vuforia API
Description:

- As a developer, I would like to explore and learn the Vuforia api so that I can learn what features of the api will be useful for the solution of the project's problem.

Acceptance Criteria:

1. Understand how to implement the Virtual Button interface.

#686 Augmented Reality Template Scene
Description:

- As a developer, I would like to design an augmented reality scene with user input so I have a template to work off for the other scenes.

Acceptance Criteria:

1. There exist user interactivity.
2. There exist multiple markers being detected.
3. The scene is being rendered in stereoscopic view.

Sprint 2

#667 Logical OR Gate in AR

Description:

- As a user I would like to view a logical OR gate in augmented reality so I can see its output based on different binary inputs.

Acceptance Criteria:

1. The user can change the input to 00, 01, 10, and 11.
2. The model notifies or displays the output based on the input.

#667 Logical AND Gate in AR

Description:

- As a student I would like to view a logical AND gate in augmented reality so I can see its output value based on different binary inputs.

Acceptance Criteria:

1. The user can change the input to 00, 01, 10, and 11.
2. The model notifies or displays the output.

Sprint 3

#673 Logic XOR Gate in AR

Description:

- As a user I would like to view a logical XOR gate in AR so I can see its output value based on different binary inputs.

Acceptance Criteria

1. The user can change the inputs to 00, 01, 10, 11
2. The model notifies or displays the output to the user.

#720 Logical NOT Gate in AR
Description:

- As a user I would like to view a logical NOT gate in AR so I can see its output value based on different binary inputs.

Acceptance Criteria:

1. The input can be set to 1 or 0.
2. The scene displays boolean table.

Sprint 4

#674 Binary Conversion Activity
Description:

- As a user I would like to convert from binary to decimal by playing a quiz game in AR.

Acceptance Criteria:

1. The user can chose from multiple answers.

#681 Difficulty Selection for Binary Conversion Activity
Description:

- As a user I would like to select different difficulty levels for the Binary Conversion Activity so I can challenge myself with more difficult questions.

Acceptance Criteria:

1. The user can choose between: Easy, Medium, and Hard.
2. The user can only make one choice.

Sprint 5

#750 Conversion Activity Modes: Binary to Hex
Description:

- As a user I would like to be tested on converting binary values to their hexadecimal equivalent so I can master both number systems.

Acceptance Criteria:

1. The user can pick from four hexadecimal values as their answer.
2. The difficulty selection scene has a button for selecting conversion mode.

#751 Mobile Device: Activities Menu
Description:

- As a user I would like to use a menu on my mobile device to select which learning activity to do so I have preference and ease of learning experience.

Acceptance Criteria:

1. The menu displays first upon launching on mobile device.
2. The menu launches all AR scenes.

Sprint 6

#676 Decoder in Augmented Reality
Description:

- As a teacher I would like to view a Decoder implementation in AR so I can show students the results of various binary inputs into a single output.

Acceptance Criteria:

1. The user can set the value for every binary input of the decoder.
2. The user can clear all values in the inputs.
3. The user can clear a single value of the inputs.
4. The user can produce the result of the various inputs by interacting with a virtual button.

#684 Balancing Binary Search Tree in Augmented Reality
Description:

- As a student I would like to see how a binary search tree gets balanced when an element gets removed in AR so I can visually understand the algorithm.

Acceptance Criteria:

1. The user can remove an element.
2. The user can pause and continue the balancing.

CodeAdventures Team:

Sprint 1

#688 Learn Unity
Description:

- As a developer, I would like to learn to use the Unity engine, so I can continue the project.

Acceptance Criteria:

1. I must watch and read tutorials about Unity3D.

#689 Learn Blender
Description:

- As a developer, I would like to learn how to use Blender, so I can alter and create new 3D models.

Acceptance Criteria:

1. I must watch and read tutorials about Blender.

Sprint 2

#687 Add Keyboard/Mouse Controls
Description:

- As a developer, I would like to add keyboard/mouse controls to the game, so I can test simple features without a Vive.

Acceptance Criteria:

1. The game can be played with a keyboard and mouse.

#693 Add an Indicator
Description:

- As a user, I would like to see an indicator when picking up a box, so it is clear that I have performed this action.

Acceptance Criteria:

1. Box is red when it is being picked up.
2. Box reverts to its starting color when it is dropped.

Sprint 3

#724 Change the Key's Material
Description:

- As a user, I would like the key to be brighter and more visible, so I know what my objective is.

Acceptance Criteria:

1. The key is bright enough to be seen at the player's starting point.

#709 Option to Change Input Device
Description:

- As a user and developer, I'd like to have multiple options for interacting with the game so that I can (as user) interact in the method most comfortable for me, and (as a developer) increase the development cycle speed by cutting out the need to attach the Vive on each test session.

Acceptance Criteria:

1. Options Menu contains choice to switch between available input devices
2. When setting is changed to the appropriate input, that input is used for interaction.

#708 Options Menu to Change Game Settings
Description:

- As a user, I would like to have a menu for settings so that I may customize aspects of the game to suit my needs.

Acceptance Criteria:

1. Options menu can be brought up using a button in the game.
2. Pauses the game when brought up.

#731 Have Robot Communicate
Description:

- As a user, I'd like to have the Robot character communicate with me to facilitate my understanding of basic rules for the game so that I may clearly understand how to play.

Acceptance Criteria:

1. Text Box appears above the head of the robot.
2. Scrolls through various messages providing delays to allow for reading.
3. Text Box constantly faces player.
4. Provide details to the player essential for their understanding of how they could possibly complete the game. (Balance between giving hints and telling the player outright what to do.)

#732 Create Button for Completing Level

Description:

- As a user, I would like to have a button in the game as a goal so that my goal in the game is clearly visible. With this I can focus on how to reach the goal instead of trying to understand what the goal is.

Acceptance Criteria:

1. Button prompts victory when pressed by robot.
2. Button is clearly visible to the user so that they can plot how they can get to it.
3. Button also causes some visible changes in the game world as to fit within the context of how it is a goal

---

#733 Prevent Robot from Going Through Walls

Description:

- As a user, I would like the robot to stop moving when it bumps into a wall, so the game and its objective make sense.

Acceptance Criteria:

1. The robot stops moving when it comes into contact with a rigid body.

#734 Create Lightning Door

Description:

- As a user, I would like to see a door with lightning instead of metal bars, so it makes more sense when the robot passes through it.

Acceptance Criteria:

1. The door's bars are made of a non-rigid lightning material.
2. The lightning is animated.

Sprint 4

#722 Add a Main Menu
Description:

- As a user, I would like to see a main menu when starting the game, so I can know my options.

Acceptance Criteria:

1. A scene with three buttons (Play, Controls, Exit) appears when the game is run.

#723 Add Instructions for Controls
Description:

- As a user, I would like to see a layout of controls, so I can know how to play the game without someone else telling me.

Acceptance Criteria:

1. When the "Controls" button is clicked in the main menu, an image describing the controls appears.

#735 Create Second Level Model
Description:

- As a user, I would like to have a second level so that I may play more.

Acceptance Criteria:

1. Level builds upon concepts established in first level to provide a more complex problem to solve.
2. Includes multiple modules.
3. Requires the user to create module transitions.
4. Goal should be the same as prior level (press button with robot), but the path required to reach that goal differs.

#736 Create Impassable Areas
Description:

- As a developer, I would like to create impassable areas, so the player cannot completely freely navigate through the scene.

Acceptance Criteria:

1. The player cannot teleport outside of the cell.

#752 Add an In-game Menu

Description:

- As a user, I would like to see an in-game menu, so I can quit the game whenever I want to.

Acceptance Criteria:

1. After clicking the Esc key, a menu will pop up with the option “Quit”.

#753 Fix Ambient Lighting

Description:

- As a user, I would like to see better lighting in the level, so it is easier to notice and differentiate objects and objectives.

Acceptance Criteria:

1. The scene is more brightly lit.

Sprint 5

#742 Create Module Transitions

Description:

- As a user, I’d like to be able to create transitions between modules to have the robot run a sequence of actions.

Acceptance Criteria:

1. User can create transitions between modules.
2. Robot is able to transition from state to state with no issue.
3. Visual cues to aid users’ interaction with this utility.

#743 Add Left Turn Module

Description:

- As a user, I’d like the mobility granted by having the robot turn to the left.

Acceptance Criteria:

1. Robot turns 90 degrees to the left when the module activates.
2. Module links correctly with other modules.

#754 Add Right Turn Module

Description:

- As a user, I would like the game to have a right-turn module, so I can direct the robot to turn right.

Acceptance Criteria:

1. The robot moves to its right when the module is called until it collides with another object.
2. The module can be linked to other existing modules.

#757 Proceed to Next Level

Description:

- As a user, I would like to pass through a door after I complete the objective, so I can move on to the next level.

Acceptance Criteria:

1. Touching the exit door with the Vive controller will load the next level scene.

Sprint 6

#755 Create Third Level

Description:

- As a user, I would like a third level, so I can solve more difficult puzzles.

Acceptance Criteria:

1. A new scene appears after the player completes the previous level.
2. The objective is more difficult than the last objective.

#760 Highlight Modules During Transitions

Description:

- As a user, I would like to see an indicator when making transitions, so it is clear that I am doing so.

Acceptance Criteria:

1. When the down button is pressed on the Vive touchpad while in a module cube, the cube will change color.
2. When the button is released, the module will revert to its starting color.

#761 Destroy Transitions

Description:

- As a user, I'd like to be able to destroy transitions I no longer want to keep.

Acceptance Criteria:

1. User pushes up on the trackpad while inside a transition to destroy it.
2. Visual cue shows transition deleting.

#763 Add Congratulations Message

Description:

- As a user, I would like to see a congratulations message at the end of the last level, so I know I have completed the game.

Acceptance Criteria:

1. A text box pops up displaying a message when the player touches the exit door in the last level.
2. The game application will close when the player clicks a button while the message is active.

#764 Create Tutorial

Description:

- As a user, I'd like to have a guided experience that facilitates my learning of how the game operates.

Acceptance Criteria:

1. Tutorial is event based, causing the user to accomplish small goals at the request of the robot.
2. Provides the user with basic skills to interact and make informed choices in the game world without solving the level for them.

#765 Adapt Main Menu to VR

Description:

- As a user, I'd like to access the main menu while in VR.

Acceptance Criteria:

1. Main menu should appear at the start of the game with the VR headset.

CircGR Team:

Sprint 1

#690 Learn C# Syntax and Semantics

Description:

- As a developer I want to learn the C# programming language's syntax and semantics to reduce the time it takes for me to translate the MTGRLibrary to a C++ API

Acceptance Criteria:

1. I must watch videos lectures on Pluralsight about C#

#692 Learn Qt Framework for C++

Description:

- As a developer I want to learn the Qt framework to develop a C++ GUI for testing the CircGR API

Acceptance Criteria:

1. I must watch the video lectures on Pluralsight for integrating the Qt cross platform framework with C++

#694 Learn C++ API Design Best Practices

Description:

- As a developer I want to learn about API design best practices in C++ to create a well-written, robust, and scalable C++ API of the MTGRLibrary currently written in C#

Acceptance Criteria:

1. Begin reading "API Design for C++" by Martin Reddy

Sprint 2

#711 Translate the Point Class to C++

Description:

- As a developer I want to learn the Qt framework to develop a C++ GUI for testing the CircGR API

Acceptance Criteria:

1. I must watch the video lectures on Pluralsight for integrating the Qt cross platform framework with C++

#712 Translate the PointMap Class to C++

Description:

- As a developer I want to translate the PointMap Class in the MTGRLibrary so that it can be used by the Gesture Class.

Acceptance Criteria:

1. Must have C++ API best design practices implemented
2. Must have the required functions, member variables, and properties that provide the exact translate of the class written in C# to C++ code

---

#714 Translate the Geometry Class to C++

Description:

- As a developer I want to translate the Geometry class from the MTGRLibrary so that it can be used by the Gesture and CircGesture class.

Acceptance Criteria:

1. Must have C++ API best design practices implemented
2. Must have the required functions, member variables, and properties that provide the exact translate of the class written in C# to C++ code

---

#715 Translate the Gesture Class to C++

Description:

- As a developer I want to translate the Gesture Class from the MTGRLibrary to C++ so that it can be used by the Recognizer class, CircGesture class, CircGR class, and the CircClassifier class.

Acceptance Criteria:

1. Must have C++ API best design practices implemented
2. Must have the required functions, member variables, and properties that provide the exact translate of the class written in C# to C++ code

---

Sprint 3

#716 Translate the Recognizer Class to C++

Description:

- As a developer I want to translate the Recognizer class from the MTGRLibrary to C++ so that it can be used by the CircGR class.

Acceptance Criteria:

1. Must have C++ API best design practices implemented
2. Must have the required functions, member variables, and properties that provide the exact translate of the class written in C# to C++ code

---

#717 Translate the CircGesture Class to C++

Description:

- As a developer I want to translate the CircGesture class from the MTGRLibrary to C++ so that it can be used by the CircGR and CircClassifier classes.

Acceptance Criteria:

1. Must have C++ API best design practices implemented
2. Must have the required functions, member variables, and properties that provide the exact translate of the class written in C# to C++ code

---

#730 Translate the DirectionalEvents Class to C++

Description:

- As a developer I want to translate the DirectionalEvents class from the MTGRLibrary to C++ so that it can be used by the CircGesture, CircGR and CircClassifier classes.

Acceptance Criteria:

1. Must have C++ API best design practices implemented
2. Must have the required functions, member variables, and properties that provide the exact translate of the class written in C# to C++ code

---

Sprint 4

#718 Translate the CircGR Class to C++

Description:

- As a developer I want to translate the CircGR class from the MTGLibrary so that it can be used by the one dollar ($) gesture recognizer and the P dollar ($P) gesture recognizer

Acceptance Criteria:

1. Must have C++ API best design practices implemented
2. Must have the required functions, member variables, and properties that provide the exact translate of the class written in C# to C++ code

---

#719 Translate the CircClassifier Class to C++

Description:

- As a developer I want to translate the CircClassifier class so that it can be used by the one dollar ($) gesture recognizer and P dollar ($P) gesture recognizer.

Acceptance Criteria:

1. Must have C++ API best design practices implemented
2. Must have the required functions, member variables, and properties that provide the exact translate of the class written in C# to C++ code

---

Sprint 5

#691 Create GUI to Test CircGR-API

Description:

- As a developer I would like to create a Graphical User Interface to begin translating the MTGRLibrary to C++ and have a means of testing the CircGR-API

Acceptance Criteria:

1. The touchpad should interact with the MTGRLibrary
2. The touchpad can be in one of two main modes: Template or Candidate

3. The menu in the touch pad should also contain “Gesture”, “Test”, “Training”, and “Experiment” options

---

#746 Test & Debug MTCircGR API

Description:

- As a developer I want to test the translated MTCircGR API so that I can confirm that all classes perform their intended tasks and interconnects well with a relative test GUI.

Acceptance Criteria:

1. Must recognize the multi touch gesture recognitions
2. Must interconnect well with a test Graphical User Interface
3. API must be stable

Sprint 6

#774 Implement Gesture Parser

Description:

- As a developer I want to create a gesture parser so that I can obtain Point data about the candidate gestures from previous semester’s experiments.

Acceptance Criteria:

1. It must successfully deserialize gesture xml data
2. It should work with GUI in order to test CircGR API

#775 Implement Gesture Serializer*

Description:

- As a developer I want to create a multi-touch gesture serializer so that I can obtain formatted input information about the candidate gestures from a touch screen device.

Acceptance Criteria:

1. It must successfully serialize test gesture into an xml structured file
2. It should work with GUI in order to test CircGR API

LEAP-Trainer Team:

Sprint 1

#670 Learn Leap API

Description:

- As a developer, I want to read the Leap Motion Controller documentation, so that I can begin development on the project.

Acceptance Criteria:

2. Select a programming language.

#671 Learn Real Sense API

Description:

- As a developer, I want to read the Real Sense Camera documentation, so that I can begin development on the project.

Acceptance Criteria:

1. Select a programming language.

#672 Build a mini Leap enabled program

Description:

- As a developer, I want to build a miniature test program that is Leap enabled, so that I can begin exploring what is and is not possible with the API.

Acceptance Criteria:

1. Successfully compile a Leap enabled program.

Sprint 2

#710 Design – Gesture Process

Description:

- As a user, I would like to capture a gesture, save that gesture, replay that gesture, and be able to delete that gesture if needed, so that I can use those gestures to create a library of American Sign Language gestures.

Acceptance Criteria:

1. Implement a way to pull hand vector data from the mini application and save it to a file.
2. Design the gesture process for the Leap Motion Controller.
3. Design a mock interface for the user.

#713 Research – Template matching
Description:

- As a developer, I want to perform research on template matching so that I might be able to implement a working algorithm that can recognize a gesture.

Acceptance Criteria:

1. Find a suitable and logical way to implement template matching.

Sprint 3

#726 Build – Database of ASL Alphabet
Description:

- As a developer, I need to build a database of gestures so that the application can be trained to recognize those gestures in real-time when performed.

Acceptance Criteria:

1. Capture vector data for as many ASL letters that can be recognized by the Leap Motion Controller.
2. Aim for a sample size of approx. 50 recordings per gesture.

#725 Improve – Output of JSON file
Description:

- As a developer, I want to improve the quality of the output as it is presently too raw, so that the JSON file can be used to create a database.

Acceptance Criteria:

1. Limit output to a single frame or the mean of vector data collected in the set of frames.
2. Format the JSON from raw data to well-formed.

Sprint 4

#727 Evaluate – Machine Learning Algorithms using Weka & Constructed database
Description:

- As a developer, I want to evaluate different machine learning algorithms so that I can find the best or most practical solution to recognize gestures in real-time.

Acceptance Criteria:

1. Evaluate as many MLA's as reasonably possible and select the one with the highest hit-rate.

#739 Begin To Implement – A Machine Learning Algorithm from Resulting Data
Description:

- As a developer, I want to use the data collected from user story #727 to select the appropriate machine learning algorithm so that I can begin to implement the real-time gesture recognition feature.

Acceptance Criteria:

1. Begin to design the structure of the MLA selected.
2. Design the architecture that will pass recorded vector data to the MLA subsystem for processing.
3. Determine if recorded data set is sufficient.

Sprint 5

#741 Continue To Implement - A Machine Learning Algorithm from Resulting Data:

- As a developer, I want to continue to work on the Nearest Neighbor approach MLA selected from the data from user story #727 to continue to implement the real-time gesture recognition feature.

Acceptance Criteria:

1. Design the architecture that will pass recorded vector data to the MLA subsystem for processing.
2. Get the program to perform some interpretation in real-time.

Sprint 6

#756 Finish Implementing - A Machine Learning Algorithm:

- As a developer, I want to finish my work on the Nearest Neighbor approach MLA selected from the data from user story #727 to improve the current implementation of the real-time gesture recognition feature.

Acceptance Criteria:

1. Get the program to perform gesture recognition in real-time, with an acceptable hit-rate of at least 50%.

Pending User Stories

CODEVR TEAM:

-

ARCSE TEAM:

- #677 Multiplexer in Augmented Reality
- #678 Half-Adder Implementation in Augmented Reality
- #679 Full-Adder Implementation in Augmented Reality
- #680 R-S Latch Implementation in Augmented Reality
- #682 Quick Sort Algorithm in Augmented Reality
- #683 Binary Search Tree Creation in Augmented Reality

CodeAdventures Team: None

CircGR Team: None

LEAP-Trainer Team: None

Project Plan

This section describes the planning that went into the realization of this project. This project incorporated the agile development techniques and as such required the sprints to be planned. These sprint plannings are detailed in the section. This section also describes the components, both software and hardware, chosen for this project.

Hardware and Software Resources

The following is a list of all hardware and software resources that were used in this project:

CodeVR Team:

- **Software Resources**
  - C++
  - Unreal Engine 4.6
  - Visual Studio 2017
  - Steam VR
- **Hardware Resources**
  - HTC Vive
  - Dell Alienware Laptop
    - Intel Core i7-6700HQ quad-core processor
    - 16GB RAM
    - NVIDIA GeForce GTX 970M graphics card
  -

ARCSE Team:

- **Software Resources**
  - C#
  - Unity 5.6.4
  - Visual Studio Code 1.18

    - Vuforia 6.2.2
    - Blender 2.79
    - Lucidchart
- **Hardware Resources**
    - Macbook Pro (15 inch, 2017)
        - Intel Core i7 2.9 Ghz 7th Generation
        - 16GB Ram
        - 512 GB SSD
    - Logitech HD Pro Webcam C920
    - Nexus 6P

CodeAdventures Team:

- **Software Resources**
    - Unity 5.6.1
    - SteamVR plugin for Unity
    - Visual Studio Community 2017
    - Blender 2.79
- **Hardware Resources**
    - HTC Vive
    - Dell Alienware Laptop
        - Intel Core i7-6700HQ quad-core processor
        - 16GB RAM
        - NVIDIA GeForce GTX 970M graphics card

CircGR Team:

- **Software:**
    - Visual Studio 2017 Enterprise Edition
    - Visual C++
    - Visual C#
    - TinyXml Library
    - Github
    - Google Drive
- **Hardware:**

- ASUS VT207 Touch Display
- Dell XPS 13 Laptop

LEAP-Trainer Team:

- **Software Resources:**
  - C#
  - Microsoft .NET 4.0 Framework
  - Newtonsoft JSON Framework for .NET
  - Leap-Motion SDK
  - Visual Studio 2017
  - Windows 10
- **Hardware Resources:**
  - Leap Motion Controller from Leap-Motion
  - Microsoft Surface Book
    - Intel i7-6600U
    - 16GB RAM

Sprints Plan

Sprint Planning Meeting Minutes: Sprint 1

Attendees: Hamilton Chevez, Nicolette Celli, Filip Klepsa, Francisco Lozada, Lukas Borges, Cristian Cabrera, Kevin Delamo
Start time: 10:05 PM
End time: 10:20 PM

After discussion, the velocity of the team was estimated to be 120.

The product owner chose the following user stories to be done during the next sprint. They are ordered based on their priority.

- #670 Learn Leap API
- #671 Real Sense API
- #672 Build a mini Leap enabled program
- #685 Learn Vuforia Api
- #686 Augmented Reality Template Scene
- #687 Add Keyboard/Mouse Control
- #688 Learn Unity
- #689 Learn Blender
- #692 Learn Qt Framework for C++
- #693 Learn C# Syntax and Semantics
- #710 Design - Gesture Process

The team members indicated their willingness to work on the following user stories.

- Hamilton Chevez
    - #685 Learn Vuforia Api
    - #686 Augmented Reality Template Scene
- Filip Klepsa
    - #670 Learn Leap API
    - #671 Real Sense API

    - #672 Build a mini Leap enabled program
- Kevin Delamo
    - #707 Learn Unity
    - #708 Options Menu to Change Games Setting
    - #709 Options to Change Input Device
- Francisco Lozada
    - #692 Learn Qt Framework for C++
    - #693 Learn C# Syntax and Semantics
- Nicolette Celli
    - #687 Add Keyboard/Mouse Control
    - #688 Learn Unity
    - #689 Learn Blender

For CodeVR, the priority is to finish moving current working features to Github, so we can start working on the Codebase:

    - #695 Move project to Github (latest features and portability)
    - #696 Call blueprint functions in the correct orders
    - #697 Improve the AST parser
    - #698 Learn jsoncpp
    - #704 Create a console overlay widget to see the python code
    - #705 Create a HUD for the game
    - #706 Move the game to VR form
    - #702 Improve the geometries being spawned (size, textures, shapes)
    - #699 Learn UE4
    - #700 Learn the project's codebase
    - #701 Integrate mesh spawn feature
    - #703 Add python support for UE4
- Cristian Cabrera
    - #698 Learn jsoncpp
    - #699 Learn UE4
    - #700 Learn the project's codebase

    - #697 Improve the AST parser
- Lukas Borges
    - #695 Move project to Github (latest features and portability)
    - #696 Call blueprint functions in the correct orders
    - #701 Integrate mesh spawn feature

Sprint Planning Meeting Minutes: Sprint 2

Attendees: Cristian Cabrera, Hamilton Chevez, Nicolette Celli, Filip Klepsa, Francisco Lozada, Lukas Borges, Kevin Delamo
Start time: 10:05 PM
End time: 10:20

After discussion, the velocity of the team was estimated to be 184.

The product owner chose the following user stories to be done during the next sprint. They are ordered based on their priority.

- #666 Logical OR Gate in AR
- #667 Logical AND Gate in AR
- #710 Design - Gesture Process
- #713 Research - Template Matchng
- #711 Translate Point Class to C++
- #712 Translate PointMap Class to C++
- #714 Translate Geometry Class to C++
- #715 Translate Gesture Class to C++
- #687 Add Keyboard/Mouse Control
- #693 Add an Indicator
- #698 Learn jsoncpp

The team members indicated their willingness to work on the following user stories.

- Hamilton Chevez
    - #666 Logical OR Gate in AR
    - #667 Logical AND Gate in AR
- Filip Klepsa
    - #710 Design - Gesture Process
    - #713 Research - Template Matching
- Kevin Delamo
    - #708 Options Menu to Change Game Settings
    - #709 Option to Change Input Device
- Francisco Lozada
    - #711 Translate Point Class to C++
    - #712 Translate PointMap Class to C++
    - #714 Translate Geometry Class to C++
    - #715 Translate Gesture Class to C++
- Nicolette Celli
    - #687 Add Keyboard/Mouse Control
    - #693 Add an Indicator
- Cristian Cabrera
    - #698 Learn jsoncpp
    - #700 Learn the project's codebase
    - #697 Improve the AST parser
- Lukas Borges
    - #695 Move project to Github (latest features and portability)
    - #696 Call blueprint functions in the correct orders
    - #701 Integrate mesh spawn feature

Sprint Planning Meeting Minutes: Sprint 3

Attendees: Cristian Cabrera, Hamilton Chevez, Nicolette Celli, Filip Klepsa, Francisco Lozada, Lukas Borges, Kevin Delamo
Start time: 5:45 PM
End time: 6:15 PM

After discussion, the velocity of the team was estimated to be X.

The product owner chose the following user stories to be done during the next sprint. They are ordered based on their priority.

- #673 Logic XOR Gate in AR
- #720 Logical NOT Gate in AR
- #716 Translate the Recognizer class to C++
- #717 Translate the CIrcGesture class to C++
- #697 Improve the AST parser
- #704 Create an in-game console widget.
- #708 Option Menu to Change Game Settings
- #709 Option to Change Input Device
- #725 Improve - Output of JSON file
- #726 Build - Database of ASL Alphabet
- #740 Pivot to a python json parser

The team members indicated their willingness to work on the following user stories.

- Hamilton Chevez
    - #673 Logic XOR Gate in AR
    - #720 Logical NOT Gate in AR
- Filip Klepsa
    - #725 Improve - Output of JSON file
    - #726 Build - Database of ASL Alphabet
- Kevin Delamo
    - #708 Option Menu to Change Game Settings
    - #709 Option to Change Input Device
- Francisco Lozada
    - #716 Translate the Recognizer class to C++

    - #717 Translate the CircGesture class to C++
    - #730 Translate the DirectionalEvents class to C++
- Nicolette Celli
    - #722 Add a Main Menu
    - #723 Add Instructions for Controls
    - #724 Change the Key's Material
- Cristian Cabrera
    - #697 Improve the AST parser
    - #728 Create new labels and materials for Items
    - #729 Integrate python server to UE4
- Lukas Borges
    - #704 Create an in-game console widget.
    - #721 Improve the AST parser
    - #729 Integrate python server to UE4

Sprint Planning Meeting Minutes: Sprint 4

Attendees: Cristian Cabrera, Hamilton Chevez, Nicolette Celli, Filip Klepsa, Francisco Lozada, Lukas Borges, Kevin Delamo
Start time: 5:45 PM
End time: 6:15 PM

After discussion, the velocity of the team was estimated to be X.

The product owner chose the following user stories to be done during the next sprint. They are ordered based on their priority.

- #674 Binary Conversion Activity

- #681 Queue in Augmented Reality
- #718 Translate CircGR Class to C++
- #719 Translate CircClassifier Class to C++
- #691 Create GUI to Test CircGR-API
- #727 Evaluate - Machine Learning Algorithms using Weka & Constructed Database
- #739 Begin To Implement – A Machine Learning Algorithm from Resulting Data
- #722 Add a Main Menu
- #723 Add Instructions for Controls
- #736 Create Impassable Areas
- #735 Create Second Level Model
- #721 Improve the AST parser
- #738 Implement classes for the parser
- #737 Move parser to C+

The team members indicated their willingness to work on the following user stories.

- Hamilton Chevez
  - #674 Binary Conversion Activity
  - #681 Queue in Augmented Reality
- Filip Klepsa
  - #727 Evaluate - Machine Learning Algorithms using Weka & Constructed Database
  - #739 Begin To Implement – A Machine Learning Algorithm from Resulting Data
- Kevin Delamo
  - #735 Create Second Level Model
- Francisco Lozada
  - #691 Create GUI to Test CircGR-API
  - #718 Translate CircGR Class to C++
  - #719 Translate CircClassifier Class to C++
- Nicolette Celli
  - #722 Add a Main Menu
  - #723 Add Instructions for Controls
  - #736 Create Impassable Areas

- Cristian Cabrera
    - #721 Improve the AST parser
    - #728 Create new labels and materials for Items
- Lukas Borges
    - #721 Improve the AST parser
    - #738 Implement classes for the parser
    - #737 Move parser to C++

Sprint Planning Meeting Minutes: Sprint 5

Attendees: Cristian Cabrera, Hamilton Chevez, Nicolette Celli, Filip Klepsa, Francisco Lozada, Lukas Borges, Kevin Delamo
Start time: 5:45 PM
End time: 6:15 PM

After discussion, the velocity of the team was estimated to be X.

The product owner chose the following user stories to be done during the next sprint. They are ordered based on their priority.

- #744 Recreate the parser using JSON for modern C++ Library
- #747 Parse instances of Assign nodes
- #748 Parse instances of FuncDef nodes
- #749 Parse instances of Expression nodes
- #745 Improve the geometries with data fed from the parser
- #750 Conversion Activity Modes: Binary to Hex
- #751 Mobile Devices: Activities Menu
- #742 Create Looping Transitions
- #743 Add Left Module
- #691 Create GUI to Test CircGR-API

- #746 Test & Debug MTCircGR API
- #741 Continue to Implement - A Machine Learning Algorithm from Resulting Data
- #754 Add Right Turn Module
- #755 Create Third Level
- #757 Proceed to Next Level
- #766 Expand Parser for all possibilities
- #767 Parse Strings
- #768 Parse all Numbers
- #769 Parse booleans
- #770 Parse for loops
- #771 Parse while loops
- #772 Parse if and else

The team members indicated their willingness to work on the following user stories.

- Hamilton Chevez
  - #750 Conversion Activity Modes: Binary to Hex
  - #751 Mobile Devices: Activities Menu
- Filip Klepsa
  - #741 Continue to Implement - A Machine Learning Algorithm from Resulting Data
- Kevin Delamo
  - #742 Create Looping Transitions
  - #743 Add Left Module
- Francisco Lozada
  - #691 Create GUI to Test CircGR-API
  - #746 Test & Debug MTCircGR API
- Nicolette Celli
  - #754 Add Right Turn Module
  - #755 Create Third Level
  - #757 Proceed to Next Level
- Cristian Cabrera
  - #766 Expand Parser for all possibilities
  - #767 Parse Strings

    - #768 Parse All Numbers
    - #769 Parse booleans
    - #770 Parse for loops
    - #771 Parse while loops
    - #772 Parse if and else
- Lukas Borges
    - #744 Recreate the parser using JSON for modern C++ Library
    - #747 Parse instances of Assign nodes
    - #748 Parse instances of FuncDef nodes
    - #749 Parse instances of Expression nodes
    - #745 Improve the geometries with data fed from the parser

Sprint Planning Meeting Minutes: Sprint 6

Attendees: Cristian Cabrera, Hamilton Chevez, Nicolette Celli, Filip Klepsa, Francisco Lozada, Lukas Borges, Kevin Delamo
Start time: 5:45 PM
End time: 6:15 PM

After discussion, the velocity of the team was estimated to be X.

The product owner chose the following user stories to be done during the next sprint. They are ordered based on their priority.

- #756 Finish Implementing - A Machine Learning Algorithm
- #676 Decoder In Augmented Reality
- #684 Balancing Binary Search Tree in Augmented Reality
- #755 Create Third Level
- #760 Highlight Modules During Transitions
- #763 Add Congratulations Message

    - #758 Draw geometries from data fed from parser.
    - #759 Use splines to connect pertinent geometries (nodes).
    - #762 Create a GUI for file selection
    - #773 Menu and Demoable GUI

The team members indicated their willingness to work on the following user stories.

- Hamilton Chevez
    - #676 Decoder In Augmented Reality
    - #684 Balancing Binary Search Tree in Augmented Reality
- Filip Klepsa
    - #756 Finish Implementing - A Machine Learning Algorithm
- Kevin Delamo
    - #761 Destroy Transitions
    - #764 Create Tutorial
    - #765 Adapt Main Menu to VR
- Francisco Lozada
    - #774 Implement Gesture Parser
    - #775 Implement Gesture Serializer
- Nicolette Celli
    - #755 Create Third Level
    - #760 Highlight Modules During Transitions
    - #763 Add Congratulations Message
- Cristian Cabrera
    - #766 Expand Parser for all possibilities
    - #772 Parse if and else
    - #773 Menu and Demoable GUI
- Lukas Borges
    - #758 Draw geometries from data fed from parser.
    - #759 Use splines to connect pertinent geometries (nodes).
    - #762 Create a GUI for file selection

System Design

This section contains information on the design decisions that went into this project. The architecture patterns are outlined and explained. The entire system is shown in a package diagram and the subsystems are explained. Finally, the design patterns used in the project are discussed.

Architectural Patterns

ARCSE Team:

- The ARCSE used the Model-View-Controller architecture for the Binary Conversion and Logic Reference Activities. The View consisted of a class that implemented Vuforia's IVirtualButtonEventHandler interface for binding the events of user interaction to calls in the controller for each activity. The models of the binary conversion activity maintained state of what mode of conversion was being performed, the answer data pool, and the choice the user picks. For the Logic Reference activity, state was kept for the current models being displayed and user's choice.

CodeAdventures Team:

- The system is structured using a Model-View-Controller architecture. We treated the Unity scene in which the game is run as a View, the game objects as Models, and the C# scripts as Controllers. This structure is standard for Unity projects and will allow us to separate the application's data, logic, and user interface, making it easier for future collaborators to continue our project.

LEAP-Trainer Team:

- The LEAP-Trainer's framework is a bit untraditional in the sense that the usual Model-View-Controller architecture representation needed to be modified to fit this application. This system connects to an external device through the USB in which all data

is collected from, adding an additional layer. To compensate, sequence diagrams were drawn to show the controller in use, the connection to the Leap Motion Device, File Helper, Training Module, or a combination. However, the overall structure of the project remains modular, partitioned by data gathering, training/interpretation logic, and the user interface.

CodeVR Team:

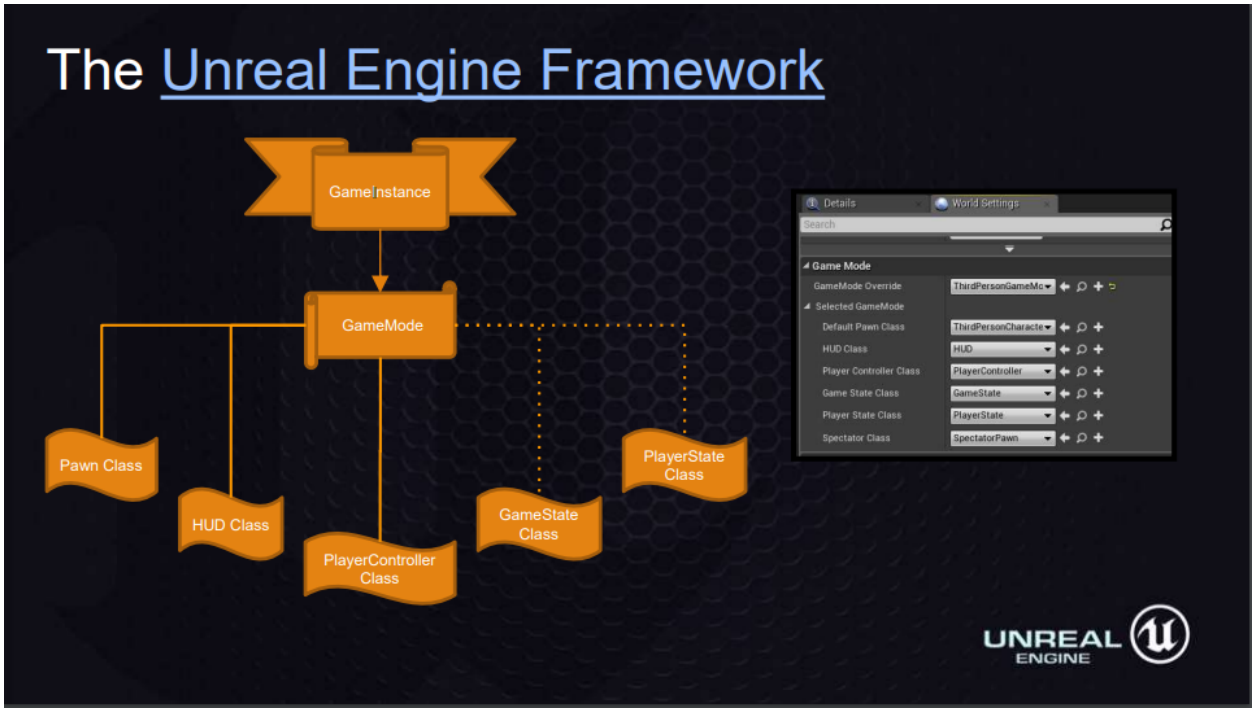
Game Engine:

Since the project relies on Virtual Reality, the decision was to make CodeVR a game application. So the first question to be answered is: How to make a game?
The modern answer to this question is “Design a Game Engine”. The Game Engine is the software layer that will solve the rendering (drawing) part of the game.
Early in the development of CodeVR, a Game Engine was in early development along with the application, but solving the rendering component is a slow, arduous, task.
Currently there are many Game Engine solutions that could solve this part of the problem, so it was decided that this project was going to be designed in the Unreal Engine 4 (UE4).

The application programming that was being developed in Rust, had to be ported into C++ so that it would run in the Unreal Engine, and that’s where the rest of the development took place.

Because the UE4 was chosen, most decisions at the Game Engine level were already made, and development takes place inside this framework.

Here's a quick overview of the UE4 framework:



- Game Instance: Wraps all others, is a class whose state persists switching on levels, game modes, pawns, etc. Where classes like GameMode or PlayerController are being reset and data stored in those classes is removed.
- Game Mode: Definition of the game.
    - Include things like the game rules and win conditions.
    - Holds important information about:
        - Pawn
        - PlayerController
        - GameState
        - PlayerState
- Pawn Class: Is the base class of all Actors that can be controlled by players or AI.

    - Pawn represents physical location, rotation, etc. of a player or entity within the game.
    - A character is a special type of pawn that has the ability to walk around
- HUD Class: The HUD is the base object for displaying elements overlaid on the screen. Every human-controlled player in the game has their own instance of the AHUD class which draws to their individual Viewport.
- PlayerController Class: Is the interface between the Pawn and the human player controlling it.
    - PlayerController decides what to do and then issues commands to the Pawn (e.g. “crouch”, “jump”).
    - Putting input handling or other functionality into the PlayerController is often necessary.
    - The PlayerController persists throughout the game, while the Pawn can be transient.
- GameState Class:
- PlayerState Class: Is the state of a participant in the game, such as a human player or a bot that is simulating a player. Non-player AI that exists as part of the game would not have a Player State.

So to sum it up:

- Object: Base building blocks in the Unreal Engine
    - Actor: Any object that can be placed into a level
        - Pawn: Subclass of Actor and serve as in-game avatar
            - Character: SubClass of a Pawn that is intended to be used as a player character.

System and Subsystem Decomposition

ARCSE-Team

- Models
    - InOutModel: A generic model for each Logic Gate model. Maintains state of whether the specific logic gate models are rendered or not.

    - DiffcButton: Used in the Binary Conversion Menu scene, maintains state of current difficulty selected.
    - ModeButton: Part of the Binary Conversion Activity, maintains state of the answer mode select: Binary or Hexadecimal.
    - AnswerButton: Part of the Binary Conversion Activity, maintains state of the answer choice the user selects.
    - OkayButton: Part of the Binary Conversion Activity, maintain state to determine if the next scene or event can be loaded.
- Controllers
    - ImageController: Controls the logic to render the Logic Gate models and to update the state of each logical InOutModel.
    - GameLogicController: Controls the logic for the Binary Conversion activity. Takes care of setting the instantiation of each model used in the conversion activity, the loading of the answer data pool, and the user's preferences for difficulty and mode of conversion.
- Views
    - VB_LogicTable: Provides the UI of virtual buttons to toggle and display the models of the logic gates and their respective reference table.
    - MobileMenuLauncher: Mobile only menu for launching activities with their associated scenes
    - BinaryConversionMenu: Receives user input through virtual button interaction and alerts users of choices made by toggling different color behaviour.
    - BinaryConversionActivity: Receives user input via virtual buttons for selecting answers. Renders updated button behaviour upon user interactivity.

CodeAdventures Team:

- Models
    - States: Represents a state from a state machine with two properties: Id that uniquely identifies the state, and Type which represents the function given to the robot to issue an action.
    - Transitions: Represents a transition from a state machine with three properties: Id that uniquely identifies the transition, From which is the state it is transitioning from, and To which is the state it is transitioning to.

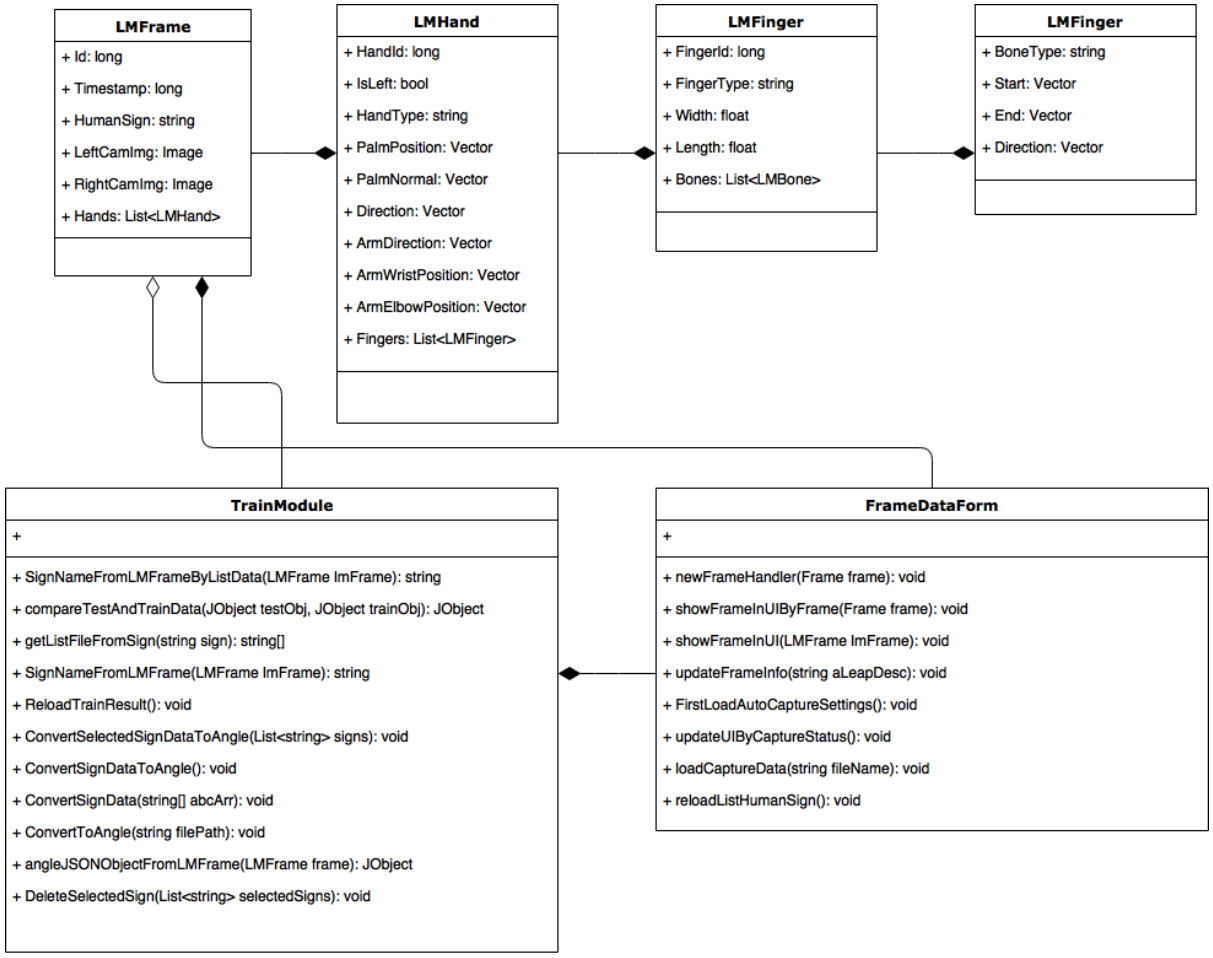
    - Modules: Represents the GameObject that holds the properties of the state model.
    - Robot: The main programmable robot that the player interacts with.
    - Board: Model where the state machine is built by placing and attaching modules together.
    - Tether: Represents a transition between two states.
- Controllers
    - Robot Controller: Controls the robot; has knowledge of the created state machine.
    - State Machine Controller: Builds the state machine based on the given modules.
    - Board Controller: Recognizes any module placed or removed from the board. Converts module data to states and sends it to the robot controller.
    - Vive Controller: Adds functionality for the Vive controller. Allows for toggling of the board, running the state machine, restarting or quitting the level, and player interaction with GameObjects.
- Views
    - Unity Scene: The scene which the player is loaded into and each model gets instantiated in. Currently loads three levels.

LEAP-Trainer Team:

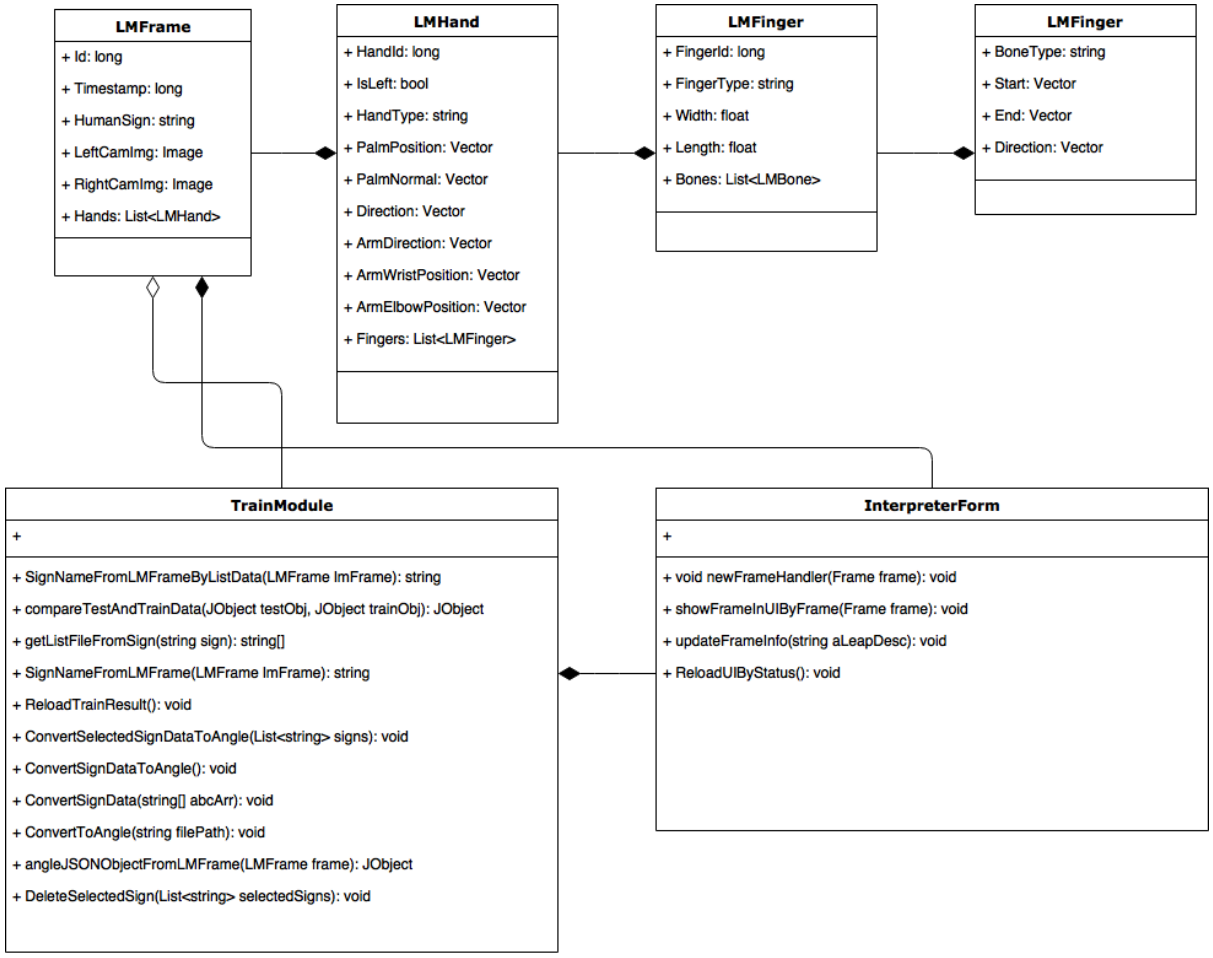
- **Models:**
    - LeapConstants: Contains constants used in angle conversion.
    - LMBone: Contains bone type, start, end, and direction.
    - LMFinger: Contains finger ID, type, width, length.
    - LMFrame: Contains frame ID, hand type, pitch/yaw/roll, wrist/elbow position, time stamp, hands, fingers.
    - LMHand: Contains palm position, palm direction, palm normal, arm direction, arm elbow, fingers.
    - LMSingleton: Contains the limits used in interpretation.
- **Controllers**
    - Capture: Handles capturing a frame.
    - Test: Handles testing a captured frame.
    - Interpret: Handles interpreting a captured frame.

- **Module**
  - TrainModule: Performs the train function which converts frame vector data into classifiable angles. Additionally, it contains the logic used to interpret trained gestures.
- **Views:**
  - Training Form: Main view of the LEAP-Trainer, used to capture and train gestures.
  - Testing Form: Sub view of the LEAP-Trainer, used to test a captured gesture.
  - Interpreter Form: Secondary view of the LEAP-Trainer, used to interpret trained gestures.

- **The Training:** This diagram represents the interactions with respect to the input device and the software when dealing with training gestures.



- **The Interpretation:** This diagram represents the interactions with respect to the input device and the software when dealing with interpreting gestures.

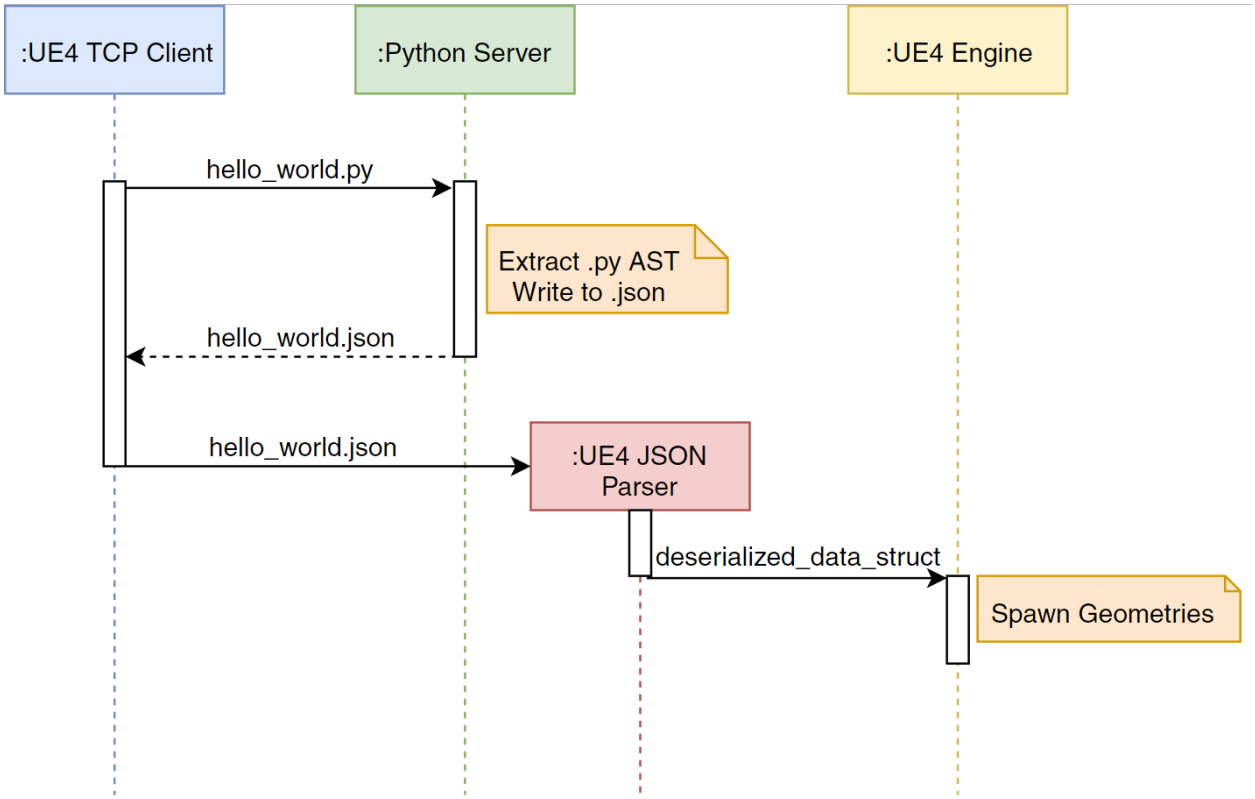


CodeVR Team

It was decided that the easiest way to go about creating CodeVR was to start from a valid (non-empty) python file program (.py). Extract the abstract syntax tree representation of this python code, and feed this data into Unreal.

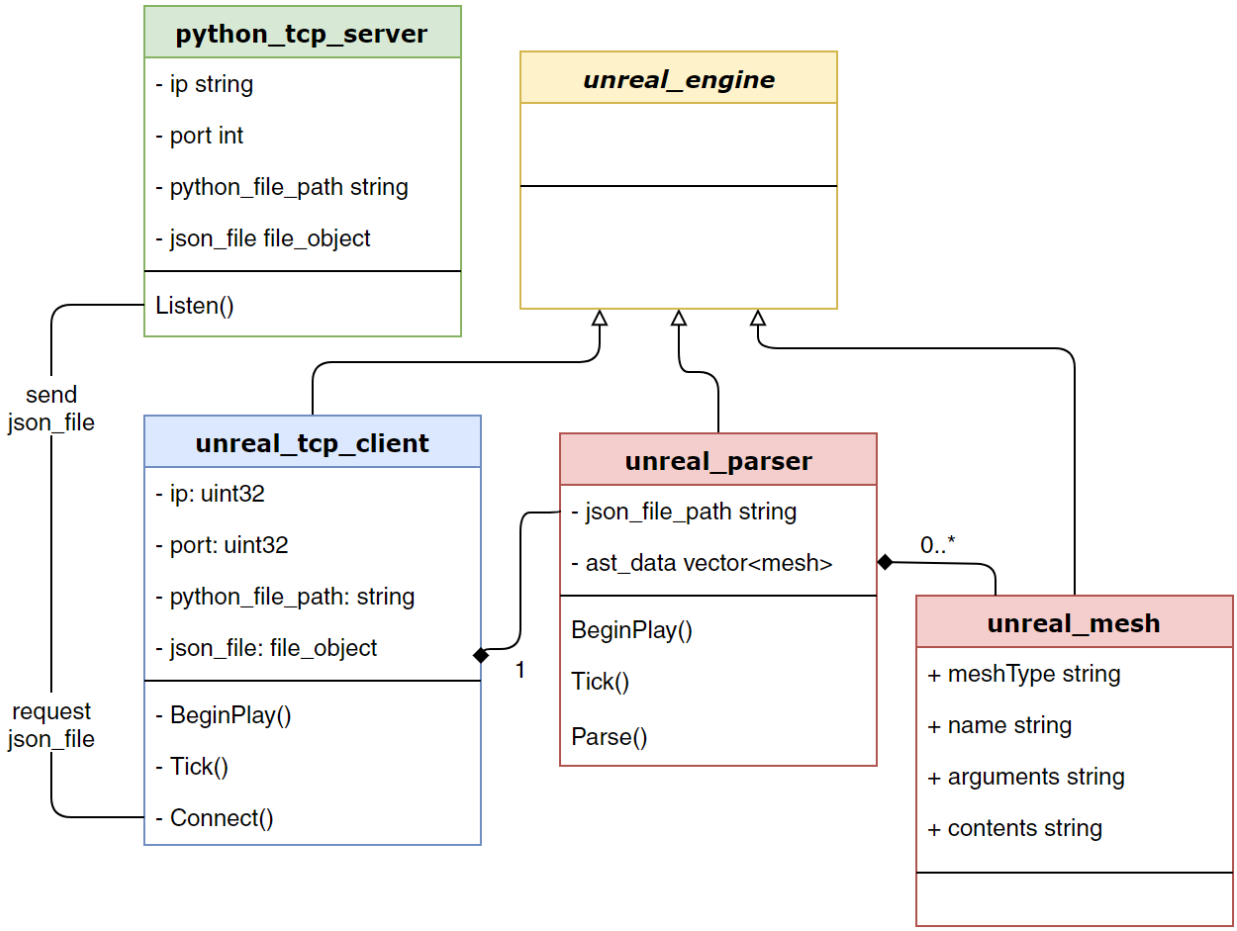
After this data loaded into Unreal's memory, Unreal is capable drawing 3D objects with their relative text information in the game world accordingly.
To better illustrate, here's a high level data flow:

In order for this data flow to take place, it is necessary to have an external python program that is able take any .py file as input and produce a .json file with Abstract Syntax Tree data correctly as output.
We used a library (python-astexport) to do this, and in order to better integrate this data flow with the Unreal Engine, TCP protocol was the communication protocol chosen.
The following system diagram should better illustrate the data communication part:



So what happens is:
Once the game starts, a python server is activated and listens (waits) on client connection.
A TCP client developed inside the Unreal Engine connects to the python server, and requests a specific .py file to be processed.
Python server finds the file in the file system, processes it, and returns the .json file to Unreal via TCP protocol.
Once Unreal has this .json file, it parses it, deserializes the data (loads it in memory) and spawn game geometries. This represents roughly is half of the functionality proposed by CodeVR.

The following diagram shows some of the inner workings of these features, and how they are all inherited (work inside) the Unreal Engine application.



Deployment Diagram

ARCSE-Team

CodeAdventures Team

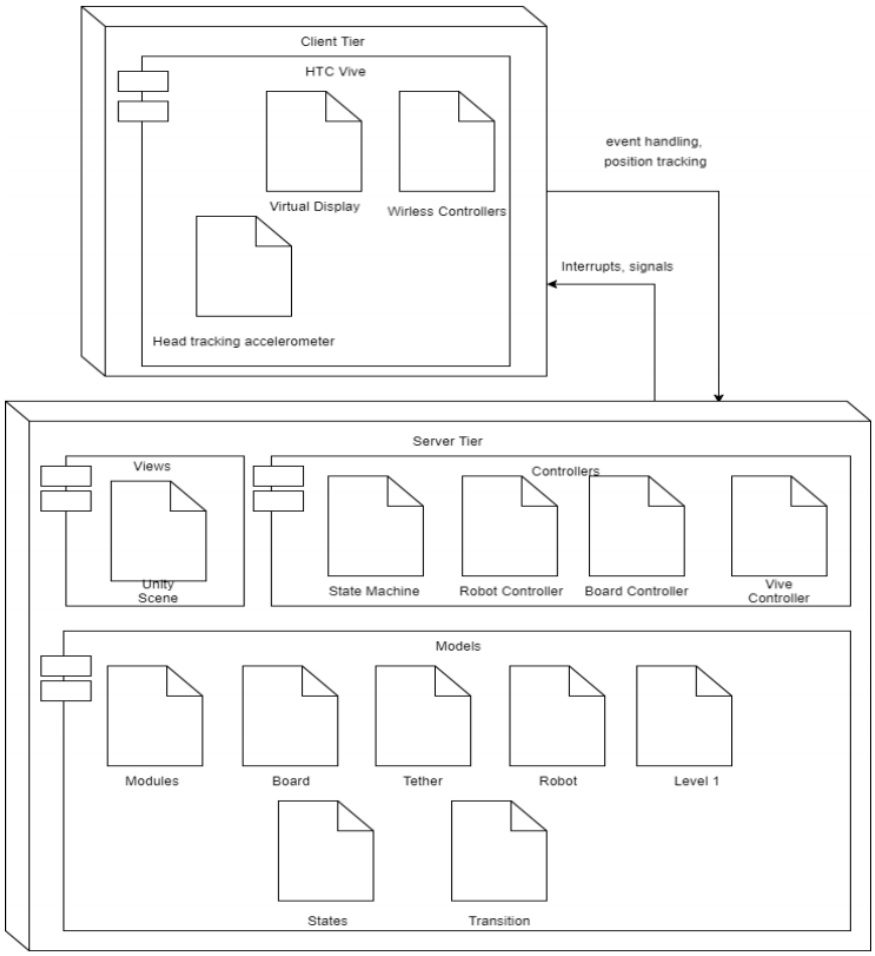


Fig 1. CodeAdventures Game Deployment Diagram

LEAP-Trainer Team

- The native application interface is provided through a dynamically loaded library. This library connects to the Leap Motion service and provides tracking data to the LEAP-Trainer application.

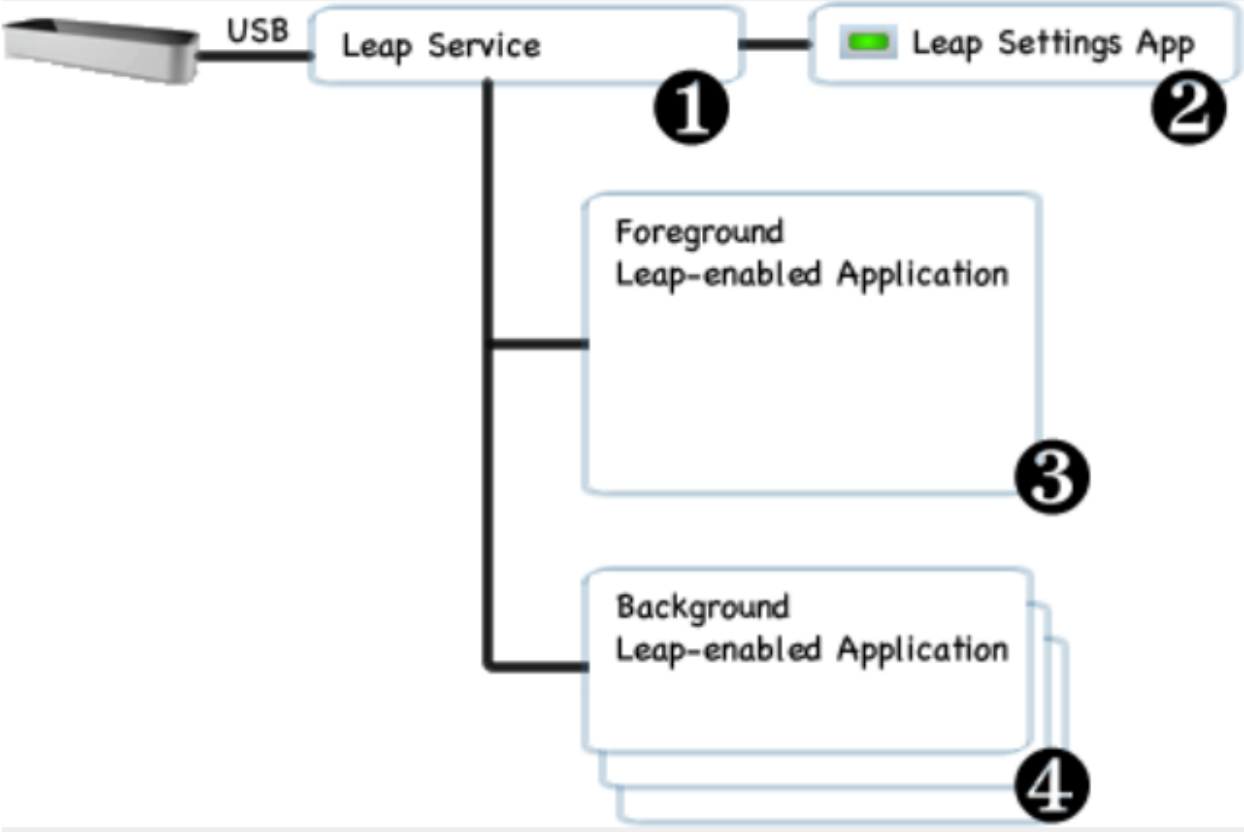


Fig 2. Leap Motion Controller Deployment Diagram

Design Patterns

System Validation

ARCSE Team:

Test Case	Description	Precondition	Expected Result	Actual Result	Status
LOG--001	Verify that setInputOne() is setting the correct value.	The user has interacted with the virtual button for input one, as a result setInputOne() has been called.	setIntputOne() returns the integer value 1	setInputOne() returns 1.	Pass
LOG-002	Verify that registerButton EventHandler assigns VButtonObjec t an event handler instance.	findGameObj ects() has been called and we have reference of the game objects in the scene.	registerButton EventHandler( ) returns True.	The method returned True.	Pass
Test Case ID	Description	Precondition	Expected Result	Actual Result	Status

LOG-003	Verify that updateModel() changes to correct 3d model in Unity.	updateModel() is called after user touches virtual button one to decimal value one.	updateModel() returns the InOutMOdel type as “Input One: One”	updateModel() returns the InOutModel type: “Input One: One”	Pass
LOG-004	Verify that UpdateInputTwo() changes the model and value of the input model.	InputTwo model is the binary value model of zero. The user touched the virtual button for input two.	The LogicalXOrTarget inputTwo field is set to 1.	UpdateInputTwo() returns the LogicalXORTarget inputTwo: 1.	Pass
BAS-001	SetDifficulty() saves the selected difficulty into the PlayerPrefs object.	PlayerPrefs.SetString(“Difficulty”, “Easy”) is called.	When PlayerPrefs.GetString(“Difficulty”) is called it will return “Easy”.	“Easy” is returned.	Pass
BAS-002	QualifyAnswer() returns false when the User picks the incorrect answer.	The correct answer for the current question is 49.	The user picks the answer 48.	QualifyAnswer() returns false.	Pass

Test Case ID	Description	Precondition	Expected Result	Actual Result	Status
BAS-003	toggleModeButton() when called, sets the PlayerPrefs “Mode” to Hex.	Current value for the PlayerPrefs “Mode” key is “Binary”	PlayerPrefs.GetString(“Mode”) returns “Hex”	The string “Hex” is returned when PlayerPrefs.GetString(“Mode”) is called.	Pass

CodeAdventures Team

Test Case ID	Description	Precondition	Expected Result	Actual Result	Status
NC-001	Check to see if box color changes when picked up.	The user is facing the box.	User clicks box and it changes color to red.	The box changes to red.	Pass
KD-001	Checks to see that settings in the entity change.	User has started the game and opened the options Menu.	User Changes a setting and setting in entity is changed.	The setting in the Options Entity changed to match	Pass
NC-002	Check to see if box color changes when	The user is facing the box.	The user releases the box and it reverts to its	The box reverts to its starting color.	Pass

	released.		starting color.		
KD-002	Robot touches Button	Player Ran Robot to touch button.	Robot runs and touches button causing the victory screen to appear.	Robot ran and touched button, prompting the victory screen to appear.	Pass
NC-003	Verify that mouse controls are functioning.	The game has loaded into the first scene and the user is moving the mouse.	The camera will follow the user's mouse.	The camera follows where the user is pointing the mouse.	Pass
KD-003	Player moves around the room while the textbox turns to face them.	Player has started the game.	The textbox will turn to face the user as they move around the room.	The textbox turned to face the player as they move around.	Pass
NC-004	Verify that keyboard controls are functioning.	The game has loaded into the first scene and the user is pressing the WASD keys.	The camera will move.	The camera moves.	Pass
NC-005	Check to see if robot goes through rigid	The player has entered and run a	The robot will stop at the bars.	The robot stopped when it hit the bars.	Pass

	bodies.	sequence in front of the metal bars.			
KD-005	Complete the first level	First level is completed	The second level will load	The second level loaded	Pass
NC-006	Check to see if the robot can pass through the lightning.	The robot is running on a player-made sequence.	The robot will pass through the lighting.	The robot passed through the lighting.	Pass
KD-006	User presses up on vivi trackpad while inside a forward module on the robot board and releases while on the right module.	Second level loaded.	A cylinder will be drawn between the modules.	Cylinder is drawn between the modules.	Pass
NC-007	Check to see if the player cannot pass through the lightning.	The user is moving towards the lightning door.	The user will not be able to go through the lightning.	The user passes through the lightning.	Fail
KD-007	User runs robot after a transition between	User draws transition between two modules	Both modules will execute and will cause the robot to touch	Both modules executed and caused the robot to touch	Pass

	forward and left modules is drawn.		the button in the level.	the button in the level.	
NC-008	Verify that the PLAY button is functioning.	The main menu scene is loaded and the user presses the PLAY button.	The first scene will be loaded.	The first scene loaded.	Pass
KD-008	Robot touches level in button.	User has run the robot using the left and forward modules.	Level ends and victory screen appears for level.	Level ends and victory screen appeared for level.	Pass
NC-009	Verify that the EXIT button is functioning.	The main menu scene is loaded and the user presses the EXIT button	The application will close.	The application closed.	Pass
KD-009	Verify the right turn module is functioning	The user has placed the right turn module on the board and pressed the run button.	The robot will move to its right until is collides with another object.	The robot moved to the right until it hit the wall.	Pass

NC-010	Verify that the CONTROLS button is functioning.	The main menu scene is loaded and the user presses the CONTROLS button.	An image on the control scheme will replace the current menu scene.	The image of the control scheme is displayed.	Pass
NC-011	Verify that the in-game menu opens.	The first level scene is loaded and the player presses the Esc key.	The in-game menu will appear, showing the QUIT button.	The in-game menu appears showing the QUIT button.	Pass
NC-012	Verify that the QUIT button is functioning.	The in-game menu is open and the player clicks the QUIT button.	The application will close.	The application closes.	Pass
NC-013	Verify that the Auto Fade script is functioning.	The in-game menu is open and the player swings his mouse quickly	The in-game menu will fade out until it disappears, then fade in when the camera is no longer moving quickly.	The menu fades out then fades back in when the camera stabilizes.	Pass
NC-014	Verify that loading the	The player is standing in	The next level will be loaded.	The next level was loaded.	Pass

	next level functions.	front of the exit door and has touched it with the Vive controller.			
NC-015	Verify that right turn module is functioning.	The user has placed the right turn module on the board and pressed the "Run" button.	The robot will move to its right until it collides with another object.	The robot moved to the right until it hit the wall.	Pass

CircGR Team:

Test case ID	Description	Precondition	Expected Result	Actual Result	Test result
Construct_Point	Test whether a Point object that stores relevant touch point info is properly created	Have the CircGR-API library (Point.cpp file in this case) and #include "Point.h" header within the code	Point object which contains X and Y coordinates, Stroke ID, and Timestamp information is created	same as expected results	Pass

Get_Difference_in_Timestamp	Tests whether the difference in creation time between two Point objects based on their Timestamp attribute is properly calculated	Have the CircGR-API library (Point.cpp file in this case) and #include “Point.h” header within the code	Difference in creation time between two points is returned	same as expected results	Pass
Get_Difference_in_Coordinates	Tests whether the spatial difference between two Point objects based on their respective X and Y coordinates is properly calculated	Have the CircGR-API library (Point.cpp file in this case) and #include “Point.h” header within the code	Spatial separation between two points is returned	same as expected results	Pass

Construct_Point_Map	Tests whether a map containing Stroke IDs as keys and list of Point objects as values is properly constructed	Have the CircGR-API library (PointMap.cpp file in this case) and #include "PointMap.h" header within the code as well as the proper stubs for the Point class	Map whose keys are integers and the values are lists of Point objects is created	same as expected results	Pass
Add_Point_to_Map	Tests whether a Point object is added to the correct value of a PointMap object	Have the CircGR-API library (PointMap.cpp file in this case) and #include "PointMap.h" header within the code as well as the proper stubs for the Point class	Point object is added to the corresponding list of Point objects that have the same Stroke ID	same as expected results	Pass

Get_Number_of_Points	Tests whether all Point objects generated are correctly counted and kept track of	Have the CircGR-API library (PointMap.cpp file in this case) and #include "PointMap.h" header within the code as well as the proper stubs for the Point class	The number of Point objects within a list of Point objects inside a Point Map is returned	same as expected results	Pass
Get_Number_of_Traces	Tests whether the number of distinct traces (or list of distinct Point objects) is correctly calculated	Have the CircGR-API library (PointMap.cpp file in this case) and #include "PointMap.h" header within the code as well as the proper stubs for the Point class	The number of distinct touch traces is returned	same as expected results	Pass

Get_List_of_Trace_IDs	Tests whether a list of the different trace IDs is correctly generated without any mistakes and duplications	Have the CircGR-API library (PointMap.cpp file in this case) and #include “PointMap.h” header within the code as well as the proper stubs for the Point class	The list of IDs for the different touch traces is returned	same as expected results	Pass
Construct_Generic_Circular_Gesture_Object	Test whether a Circular Gesture object that stores a resampled fixed number of specified points normalized with respect to scale and translated to origin as well as temporal and special angles is properly created	Have the CircGR-API library (all cpp and headers files inside namespace ‘GR’) and #include “CircGesture.h” header within the code	CircGesture object that contains all information relevant to the inputted multistroke circular gesture	same as expected results	Pass

Handle_Touch_Event_Directions	Tests whether a touch events direction information is properly calculated and handled within the DirectionalEvents class so that it can be used within the CircGesture class to calculate other relevant information like spatial angles	Have the CircGR-API library (all cpp and headers files inside namespace 'GR') and #include "DirectionalEvents.h" and "CircGesture.h" header within the code	Touch event directions correctly calculated	same as expected results	Pass
Verify_CircGR_Performs_Same_Function_As_CircClassifier_Class	Test whether a Circular Gesture object that contains circular measurements and labels pertaining to an inputted gesture is properly used by the CircGR to perform same function as the CircClassifier class. This function is to determine what template best matches the inputted gesture.	Have the CircGR-API library (all cpp and headers files inside namespace 'GR') and #include "CircGR.h" header within the code	Inputted circular gesture is accurately matched to the intended template	same as expected results	Pass

Classify_ Inputted_ Circular_ Gesture	Test whether a Circular Gesture object that contains circular measurements and labels pertaining to an inputted gesture is properly used by the CircClassifier class to determine what template best matches the inputted gesture.	Have the CircGR-API library (all cpp and headers files inside namespace 'GR') and #include "CircClassifier.h" header within the code	Inputted circular gesture is accurately matched to the intended template	same as expected results	Pass

LEAP-Trainer Team:

- **Training Validation:**

Test case ID	Testcase name	Description	Procedure	Expected Result	Test result
TC_01	Training Sign A	Check Training function for create gesture Sign A with time out = 500ms and file count = 40	1. Open App 2. Connect with device 3. Double click A 4. Set Auto capture time out = 500ms 5. Click auto capture 6. Perform gesture 7. Click TRAIN button on the lower left corner 8. Select A gesture 9. Click "OK"	1. Show capture and debug console on right screen 2. Sign A contains 80 files (40 files have the extension .txt and 40 files have _angle.text)	Pass

TC_02	Training Sign B	Check Training function for create gesture Sign B with time out = 500ms and file count = 40	1. Open App 2. Connect with device 3. Double click B 4. Set Auto capture time out = 500ms 5. Click auto capture 6. Perform gesture 7. Click TRAIN button on the lower left corner 8. Select B gesture 9. Click "OK"	1. Show capture and debug console on right screen 2. Sign B contains 80 files (40 files have the extension .txt and 40 files have _angle.text)	Pass
TC_03	Training Sign C	Check Training function for create gesture Sign C with time out = 500ms and file count = 40	1. Open App 2. Connect with device 3. Double click C 4. Set Auto capture time out = 500ms 5. Click auto capture 6. Perform gesture 7. Click TRAIN button on the lower left corner 8. Select C gesture 9. Click "OK"	1. Show capture and debug console on right screen 2. Sign C contains 80 files (40 files have the extension .txt and 40 files have _angle.text)	Pass

TC_04	Training Sign D	Check Training function for create gesture Sign D with time out = 500ms and file count = 40	1. Open App 2. Connect with device 3. Double click D 4. Set Auto capture time out = 500ms 5. Click auto capture 6. Perform gesture 7. Click TRAIN button on the lower left corner 8. Select D gesture 9. Click "OK"	1. Show capture and debug console on right screen 2. Sign D contains 80 files (40 files have the extension .txt and 40 files have _angle.text)	Pass
TC_05	Training Sign E	Check Training function for create gesture Sign E with time out = 500ms and file count = 40	1. Open App 2. Connect with device 3. Double click E 4. Set Auto capture time out = 500ms 5. Click auto capture 6. Perform gesture 7. Click TRAIN button on the lower left corner 8. Select E gesture 9. Click "OK"	1. Show capture and debug console on right screen 2. Sign E contains 80 files (40 files have the extension .txt and 40 files have _angle.text)	Pass

TC_06	Training Sign F	Check Training function for create gesture Sign F with time out = 500ms and file count = 40	1. Open App 2. Connect with device 3. Double click F 4. Set Auto capture time out = 500ms 5. Click auto capture 6. Perform gesture 7. Click TRAIN button on the lower left corner 8. Select F gesture 9. Click "OK"	1. Show capture and debug console on right screen 2. Sign F contains 80 files (40 files have the extension .txt and 40 files have _angle.text)	Pass
TC_07	Training Sign G	Check Training function for create gesture Sign G with time out = 500ms and file count = 40	1. Open App 2. Connect with device 3. Double click G 4. Set Auto capture time out = 500ms 5. Click auto capture 6. Perform gesture 7. Click TRAIN button on the lower left corner 8. Select G gesture 9. Click "OK"	1. Show capture and debug console on right screen 2. Sign G contains 80 files (40 files have the extension .txt and 40 files have _angle.text)	Pass

TC_08	Training Sign H	Check Training function for create gesture Sign H with time out = 500ms and file count = 40	1. Open App 2. Connect with device 3. Double click H 4. Set Auto capture time out = 500ms 5. Click auto capture 6. Perform gesture 7. Click TRAIN button on the lower left corner 8. Select H gesture 9. Click "OK"	1. Show capture and debug console on right screen 2. Sign H contains 80 files (40 files have the extension .txt and 40 files have _angle.text)	Pass
TC_09	Training Sign I	Check Training function for create gesture Sign I with time out = 500ms and file count = 40	1. Open App 2. Connect with device 3. Double click I 4. Set Auto capture time out = 500ms 5. Click auto capture 6. Perform gesture 7. Click TRAIN button on the lower left corner 8. Select I gesture 9. Click "OK"	1. Show capture and debug console on right screen 2. Sign I contains 80 files (40 files have the extension .txt and 40 files have _angle.text)	Pass

TC_10	Training Sign J	Check Training function for create gesture Sign J with time out = 500ms and file count = 40	1. Open App 2. Connect with device 3. Double click J 4. Set Auto capture time out = 500ms 5. Click auto capture 6. Perform gesture 7. Click TRAIN button on the lower left corner 8. Select J gesture 9. Click "OK"	1. Show capture and debug console on right screen 2. Sign J contains 80 files (40 files have the extension .txt and 40 files have _angle.text)	Pass
TC_11	Training Sign K	Check Training function for create gesture Sign K with time out = 500ms and file count = 40	1. Open App 2. Connect with device 3. Double click K 4. Set Auto capture time out = 500ms 5. Click auto capture 6. Perform gesture 7. Click TRAIN button on the lower left corner 8. Select K gesture 9. Click "OK"	1. Show capture and debug console on right screen 2. Sign K contains 80 files (40 files have the extension .txt and 40 files have _angle.text)	Pass

TC_12	Training Sign L	Check Training function for create gesture Sign L with time out = 500ms and file count = 40	1. Open App 2. Connect with device 3. Double click L 4. Set Auto capture time out = 500ms 5. Click auto capture 6. Perform gesture 7. Click TRAIN button on the lower left corner 8. Select L gesture 9. Click "OK"	1. Show capture and debug console on right screen 2. Sign L contains 80 files (40 files have the extension .txt and 40 files have _angle.text)	Pass
TC_13	Training Sign M	Check Training function for create gesture Sign M with time out = 500ms and file count = 40	1. Open App 2. Connect with device 3. Double click M 4. Set Auto capture time out = 500ms 5. Click auto capture 6. Perform gesture 7. Click TRAIN button on the lower left corner 8. Select M gesture 9. Click "OK"	1. Show capture and debug console on right screen 2. Sign M contains 80 files (40 files have the extension .txt and 40 files have _angle.text)	Pass

TC_14	Training Sign N	Check Training function for create gesture Sign N with time out = 500ms and file count = 40	1. Open App 2. Connect with device 3. Double click N 4. Set Auto capture time out = 500ms 5. Click auto capture 6. Perform gesture 7. Click TRAIN button on the lower left corner 8. Select N gesture 9. Click "OK"	1. Show capture and debug console on right screen 2. Sign N contains 80 files (40 files have the extension .txt and 40 files have _angle.text)	Pass
TC_15	Training Sign O	Check Training function for create gesture Sign O with time out = 500ms and file count = 40	1. Open App 2. Connect with device 3. Double click O 4. Set Auto capture time out = 500ms 5. Click auto capture 6. Perform gesture 7. Click TRAIN button on the lower left corner 8. Select O gesture 9. Click "OK"	1. Show capture and debug console on right screen 2. Sign O contains 80 files (40 files have the extension .txt and 40 files have _angle.text)	Pass

TC_16	Training Sign P	Check Training function for create gesture Sign P with time out = 500ms and file count = 40	1. Open App 2. Connect with device 3. Double click P 4. Set Auto capture time out = 500ms 5. Click auto capture 6. Perform gesture 7. Click TRAIN button on the lower left corner 8. Select P gesture 9. Click "OK"	1. Show capture and debug console on right screen 2. Sign P contains 80 files (40 files have the extension .txt and 40 files have _angle.text)	Pass
TC_17	Training Sign Q	Check Training function for create gesture Sign Q with time out = 500ms and file count = 40	1. Open App 2. Connect with device 3. Double click Q 4. Set Auto capture time out = 500ms 5. Click auto capture 6. Perform gesture 7. Click TRAIN button on the lower left corner 8. Select Q gesture 9. Click "OK"	1. Show capture and debug console on right screen 2. Sign Q contains 80 files (40 files have the extension .txt and 40 files have _angle.text)	Pass

TC_18	Training Sign R	Check Training function for create gesture Sign R with time out = 500ms and file count = 40	1. Open App 2. Connect with device 3. Double click R 4. Set Auto capture time out = 500ms 5. Click auto capture 6. Perform gesture 7. Click TRAIN button on the lower left corner 8. Select R gesture 9. Click "OK"	1. Show capture and debug console on right screen 2. Sign R contains 80 files (40 files have the extension .txt and 40 files have _angle.text)	Pass
TC_19	Training Sign S	Check Training function for create gesture Sign S with time out = 500ms and file count = 40	1. Open App 2. Connect with device 3. Double click S 4. Set Auto capture time out = 500ms 5. Click auto capture 6. Perform gesture 7. Click TRAIN button on the lower left corner 8. Select S gesture 9. Click "OK"	1. Show capture and debug console on right screen 2. Sign S contains 80 files (40 files have the extension .txt and 40 files have _angle.text)	Pass

TC_20	Training Sign T	Check Training function for create gesture Sign T with time out = 500ms and file count = 40	1. Open App 2. Connect with device 3. Double click T 4. Set Auto capture time out = 500ms 5. Click auto capture 6. Perform gesture 7. Click TRAIN button on the lower left corner 8. Select T gesture 9. Click "OK"	1. Show capture and debug console on right screen 2. Sign T contains 80 files (40 files have the extension .txt and 40 files have _angle.text)	Pass
TC_21	Training Sign U	Check Training function for create gesture Sign U with time out = 500ms and file count = 40	1. Open App 2. Connect with device 3. Double click U 4. Set Auto capture time out = 500ms 5. Click auto capture 6. Perform gesture 7. Click TRAIN button on the lower left corner 8. Select U gesture 9. Click "OK"	1. Show capture and debug console on right screen 2. Sign U contains 80 files (40 files have the extension .txt and 40 files have _angle.text)	Pass

TC_22	Training Sign V	Check Training function for create gesture Sign V with time out = 500ms and file count = 40	1. Open App 2. Connect with device 3. Double click U 4. Set Auto capture time out = 500ms 5. Click auto capture 6. Perform gesture 7. Click TRAIN button on the lower left corner 8. Select U gesture 9. Click "OK"	1. Show capture and debug console on right screen 2. Sign V contains 80 files (40 files have the extension .txt and 40 files have _angle.text)	Pass
TC_23	Training Sign W	Check Training function for create gesture Sign W with time out = 500ms and file count = 40	1. Open App 2. Connect with device 3. Double click W 4. Set Auto capture time out = 500ms 5. Click auto capture 6. Perform gesture 7. Click TRAIN button on the lower left corner 8. Select W gesture 9. Click "OK"	1. Show capture and debug console on right screen 2. Sign W contains 80 files (40 files have the extension .txt and 40 files have _angle.text)	Pass

TC_24	Training Sign X	Check Training function for create gesture Sign X with time out = 500ms and file count = 40	1. Open App 2. Connect with device 3. Double click X 4. Set Auto capture time out = 500ms 5. Click auto capture 6. Perform gesture 7. Click TRAIN button on the lower left corner 8. Select X gesture 9. Click "OK"	1. Show capture and debug console on right screen 2. Sign X contains 80 files (40 files have the extension .txt and 40 files have _angle.text)	Pass
TC_25	Training Sign Y	Check Training function for create gesture Sign Y with time out = 500ms and file count = 40	1. Open App 2. Connect with device 3. Double click Y 4. Set Auto capture time out = 500ms 5. Click auto capture 6. Perform gesture 7. Click TRAIN button on the lower left corner 8. Select Y gesture 9. Click "OK"	1. Show capture and debug console on right screen 2. Sign Y contains 80 files (40 files have the extension .txt and 40 files have _angle.text)	Pass

TC_26	Training Sign Z	Check Training function for create gesture Sign Z with time out = 500ms and file count = 40	1. Open App 2. Connect with device 3. Double click Z 4. Set Auto capture time out = 500ms 5. Click auto capture 6. Perform gesture 7. Click TRAIN button on the lower left corner 8. Select Z gesture 9. Click "OK"	1. Show capture and debug console on right screen 2. Sign Z contains 80 files (40 files have the extension .txt and 40 files have _angle.text)	Pass

- **Interpretation Validation:**

Test case ID	Testcase name	Description	Procedure	Expected Result	Test result
TC_01	Interpreter with Sign A	Check interpreter function for gesture Sign A	1. Open App 2. Connect with device 3. Create gesture Sign A and click TRAIN button 4. Click menu and choose Interpreter 5. Perform the gesture 6. Click Interpret button	1. Show capture and debug console on right screen 2. Show result (Detected Sign: A) and rate lowest fail counts in other signs (see the right corner screen)	Pass
TC_02	Interpreter with Sign B	Check interpreter function for gesture Sign B	1. Open App 2. Connect with device 3. Create gesture Sign B and click TRAIN button 4. Click menu and	1. Show capture and debug console on right screen 2. Show result (Detected Sign: B) and rate lowest fail	Pass

			choose Interpreter 5. Perform the gesture 6. Click Interpret button	counts in other signs (see the right corner screen)	
TC_03	Interpreter with Sign C	Check interpreter function for gesture Sign C	1. Open App 2. Connect with device 3. Create gesture Sign C and click TRAIN button 4. Click menu and choose Interpreter 5. Perform the gesture 6. Click Interpret button	1. Show capture and debug console on right screen 2. Show result (Detected Sign: C) and rate lowest fail counts in other signs (see the right corner screen)	Pass
TC_04	Interpreter with Sign D	Check interpreter function for gesture Sign D	1. Open App 2. Connect with device 3. Create gesture Sign D and click TRAIN button 4. Click menu and choose Interpreter 5. Perform the gesture 6. Click Interpret button	1. Show capture and debug console on right screen 2. Show result (Detected Sign: D) and rate lowest fail counts in other signs (see the right corner screen)	Pass
TC_05	Interpreter with Sign E	Check interpreter function for gesture Sign E	1. Open App 2. Connect with device 3. Create gesture Sign E and click TRAIN button 4. Click menu and choose Interpreter 5. Perform the gesture 6. Click Interpret	1. Show capture and debug console on right screen 2. Show result (Detected Sign: E) and rate lowest fail counts in other signs (see the right corner screen)	Pass

			button		
TC_06	Interpreter with Sign F	Check interpreter function for gesture Sign F	1. Open App 2. Connect with device 3. Create gesture Sign F and click TRAIN button 4. Click menu and choose Interpreter 5. Perform the gesture 6. Click Interpret button	1. Show capture and debug console on right screen 2. Show result (Detected Sign: F) and rate lowest fail counts in other signs (see the right corner screen)	Pass
TC_07	Interpreter with Sign G	Check interpreter function for gesture Sign G	1. Open App 2. Connect with device 3. Create gesture Sign G and click TRAIN button 4. Click menu and choose Interpreter 5. Perform the gesture 6. Click Interpret button	1. Show capture and debug console on right screen 2. Show result (Detected Sign: G) and rate lowest fail counts in other signs (see the right corner screen)	Pass
TC_08	Interpreter with Sign H	Check interpreter function for gesture Sign H	1. Open App 2. Connect with device 3. Create gesture Sign H and click TRAIN button 4. Click menu and choose Interpreter 5. Perform the gesture 6. Click Interpret button	1. Show capture and debug console on right screen 2. Show result (Detected Sign: H) and rate lowest fail counts in other signs (see the right corner screen)	Pass

TC_09	Interpreter with Sign I	Check interpreter function for gesture Sign I	1. Open App 2. Connect with device 3. Create gesture Sign I and click TRAIN button 4. Click menu and choose Interpreter 5. Perform the gesture 6. Click Interpret button	1. Show capture and debug console on right screen 2. Show result (Detected Sign: I) and rate lowest fail counts in other signs (see the right corner screen)	Pass
TC_10	Interpreter with Sign J	Check interpreter function for gesture Sign J	1. Open App 2. Connect with device 3. Create gesture Sign J and click TRAIN button 4. Click menu and choose Interpreter 5. Perform the gesture 6. Click Interpret button	1. Show capture and debug console on right screen 2. Show result (Detected Sign: J) and rate lowest fail counts in other signs (see the right corner screen)	Pass
TC_11	Interpreter with Sign K	Check interpreter function for gesture Sign K	1. Open App 2. Connect with device 3. Create gesture Sign K and click TRAIN button 4. Click menu and choose Interpreter 5. Perform the gesture 6. Click Interpret button	1. Show capture and debug console on right screen 2. Show result (Detected Sign: K) and rate lowest fail counts in other signs (see the right corner screen)	Pass

TC_12	Interpreter with Sign L	Check interpreter function for gesture Sign L	1. Open App 2. Connect with device 3. Create gesture Sign L and click TRAIN button 4. Click menu and choose Interpreter 5. Perform the gesture 6. Click Interpret button	1. Show capture and debug console on right screen 2. Show result (Detected Sign: L) and rate lowest fail counts in other signs (see the right corner screen)	Pass
TC_13	Interpreter with Sign M	Check interpreter function for gesture Sign M	1. Open App 2. Connect with device 3. Create gesture Sign M and click TRAIN button 4. Click menu and choose Interpreter 5. Perform the gesture 6. Click Interpret button	1. Show capture and debug console on right screen 2. Show result (Detected Sign: M) and rate lowest fail counts in other signs (see the right corner screen)	Pass
TC_14	Interpreter with Sign N	Check interpreter function for gesture Sign N	1. Open App 2. Connect with device 3. Create gesture Sign N and click TRAIN button 4. Click menu and choose Interpreter 5. Perform the gesture 6. Click Interpret button	1. Show capture and debug console on right screen 2. Show result (Detected Sign: N) and rate lowest fail counts in other signs (see the right corner screen)	Pass

TC_15	Interpreter with Sign O	Check interpreter function for gesture Sign O	1. Open App 2. Connect with device 3. Create gesture Sign O and click TRAIN button 4. Click menu and choose Interpreter 5. Perform the gesture 6. Click Interpret button	1. Show capture and debug console on right screen 2. Show result (Detected Sign: O) and rate lowest fail counts in other signs (see the right corner screen)	Pass
TC_16	Interpreter with Sign P	Check interpreter function for gesture Sign P	1. Open App 2. Connect with device 3. Create gesture Sign P and click TRAIN button 4. Click menu and choose Interpreter 5. Perform the gesture 6. Click Interpret button	1. Show capture and debug console on right screen 2. Show result (Detected Sign: P) and rate lowest fail counts in other signs (see the right corner screen)	Pass
TC_17	Interpreter with Sign Q	Check interpreter function for gesture Sign Q	1. Open App 2. Connect with device 3. Create gesture Sign Q and click TRAIN button 4. Click menu and choose Interpreter 5. Perform the gesture 6. Click Interpret button	1. Show capture and debug console on right screen 2. Show result (Detected Sign: Q) and rate lowest fail counts in other signs (see the right corner screen)	Pass

TC_18	Interpreter with Sign R	Check interpreter function for gesture Sign R	1. Open App 2. Connect with device 3. Create gesture Sign R and click TRAIN button 4. Click menu and choose Interpreter 5. Perform the gesture 6. Click Interpret button	1. Show capture and debug console on right screen 2. Show result (Detected Sign: R) and rate lowest fail counts in other signs (see the right corner screen)	Pass
TC_19	Interpreter with Sign S	Check interpreter function for gesture Sign S	1. Open App 2. Connect with device 3. Create gesture Sign S and click TRAIN button 4. Click menu and choose Interpreter 5. Perform the gesture 6. Click Interpret button	1. Show capture and debug console on right screen 2. Show result (Detected Sign: S) and rate lowest fail counts in other signs (see the right corner screen)	Pass
TC_20	Interpreter with Sign T	Check interpreter function for gesture Sign T	1. Open App 2. Connect with device 3. Create gesture Sign T and click TRAIN button 4. Click menu and choose Interpreter 5. Perform the gesture 6. Click Interpret button	1. Show capture and debug console on right screen 2. Show result (Detected Sign: T) and rate lowest fail counts in other signs (see the right corner screen)	Pass

TC_21	Interpreter with Sign U	Check interpreter function for gesture Sign U	1. Open App 2. Connect with device 3. Create gesture Sign U and click TRAIN button 4. Click menu and choose Interpreter 5. Perform the gesture 6. Click Interpret button	1. Show capture and debug console on right screen 2. Show result (Detected Sign: U) and rate lowest fail counts in other signs (see the right corner screen)	Pass
TC_22	Interpreter with Sign V	Check interpreter function for gesture Sign V	1. Open App 2. Connect with device 3. Create gesture Sign V and click TRAIN button 4. Click menu and choose Interpreter 5. Perform the gesture 6. Click Interpret button	1. Show capture and debug console on right screen 2. Show result (Detected Sign: V) and rate lowest fail counts in other signs (see the right corner screen)	Pass
TC_23	Interpreter with Sign W	Check interpreter function for gesture Sign W	1. Open App 2. Connect with device 3. Create gesture Sign W and click TRAIN button 4. Click menu and choose Interpreter 5. Perform the gesture 6. Click Interpret button	1. Show capture and debug console on right screen 2. Show result (Detected Sign: W) and rate lowest fail counts in other signs (see the right corner screen)	Pass

TC_24	Interpreter with Sign X	Check interpreter function for gesture Sign X	1. Open App 2. Connect with device 3. Create gesture Sign X and click TRAIN button 4. Click menu and choose Interpreter 5. Perform the gesture 6. Click Interpret button	1. Show capture and debug console on right screen 2. Show result (Detected Sign: X) and rate lowest fail counts in other signs (see the right corner screen)	Pass
TC_25	Interpreter with Sign Y	Check interpreter function for gesture Sign Y	1. Open App 2. Connect with device 3. Create gesture Sign Y and click TRAIN button 4. Click menu and choose Interpreter 5. Perform the gesture 6. Click Interpret button	1. Show capture and debug console on right screen 2. Show result (Detected Sign: Y) and rate lowest fail counts in other signs (see the right corner screen)	Pass
TC_26	Interpreter with Sign Z	Check interpreter function for gesture Sign Z	1. Open App 2. Connect with device 3. Create gesture Sign Z and click TRAIN button 4. Click menu and choose Interpreter 5. Perform the gesture 6. Click Interpret button	1. Show capture and debug console on right screen 2. Show result (Detected Sign: Z) and rate lowest fail counts in other signs (see the right corner screen)	Pass

CodeVR Team:

Test case ID	Testcase name	Description	Procedure	Expected Result	Test result
TC01	AST extract	Python program must extract AST data correctly	1. Run the python script 2. .py file as input 3. Correct json as output	All expressions, assignments and function definitions should match	Pass
TC02	Parse json data	C++ Parser should be able to read the json and deserialize the data accordingly	1. Run the parse function with selected file. 2. Load all data structures accordingly and correctly	All expressions, assignments and funcefinitions correctly deserialized in their data structures	Pass
TC03	Geometry Spawner	Traverses all data structures, draw them in game world accordingly	1. Run the server 2. Run the client 3. Send file 4. Get json 5. Run parser 6. Traverse data structures	It traversed all required data structures and drew geometries accordingly	Pass

CodeVR's objective is to improve computer science students' learning of python language and programming concepts. This claim will be tested for vsalidity. It is part of the lab's plan to run statistically sound experiments to solidify this claim once the application is ready.

GLOSSARY

ARCSE Team:

CodeAdventures Team:

CircGR Team:

- CircGR: Multi-touch Circular Gesture Recognizer
- Scratchpad: Program created to test user touch input

LEAP-Trainer Team:

- Leap Motion Controller: The handheld hardware device utilized by the software to record and interpret gestures.

CodeVR Team

APPENDIX

Appendix A - UML Diagrams

ARCSE Team:

Use Case Diagrams:

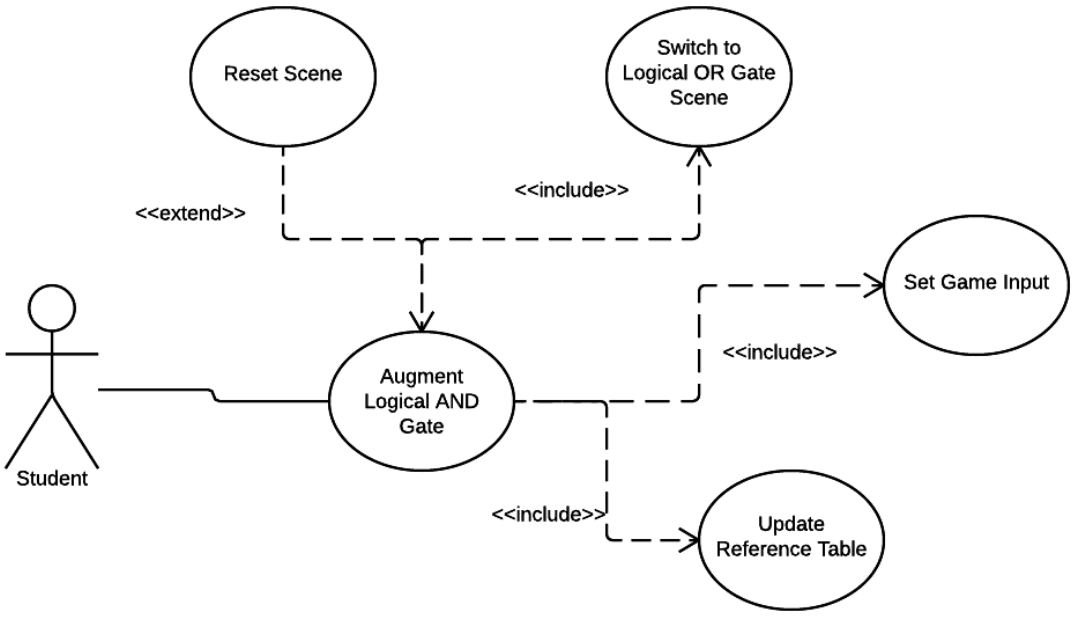


Figure 1. Logical AND Use Case Diagram

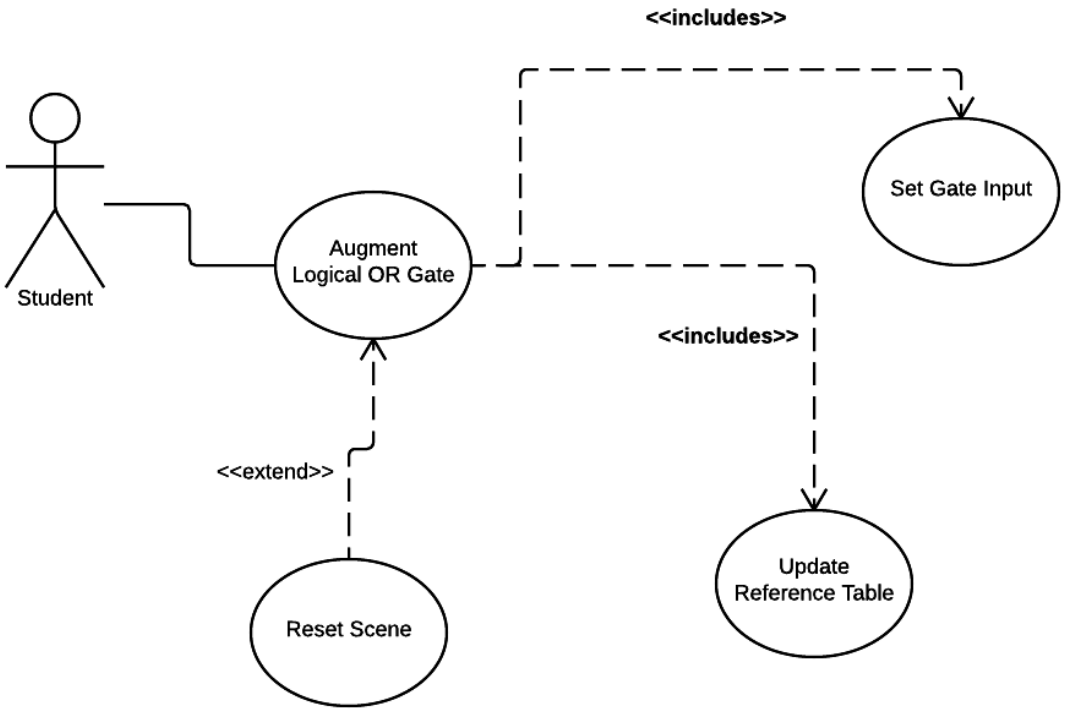


Fig. 2 Logical OR Use Case Diagram

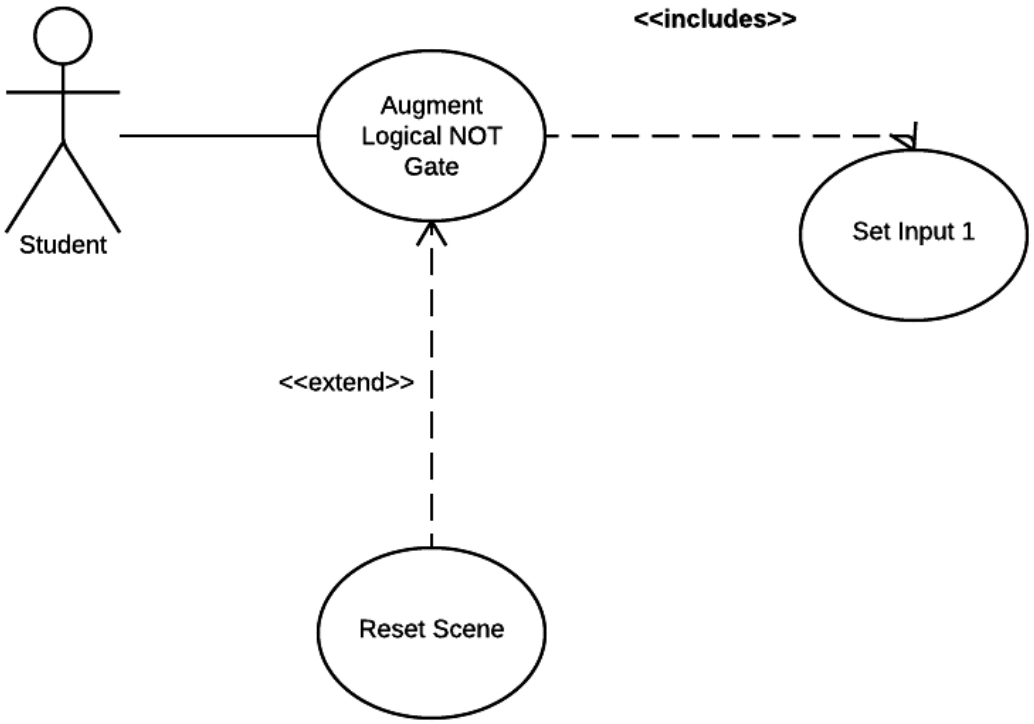


Fig 3. Logical Not Use Case Diagram

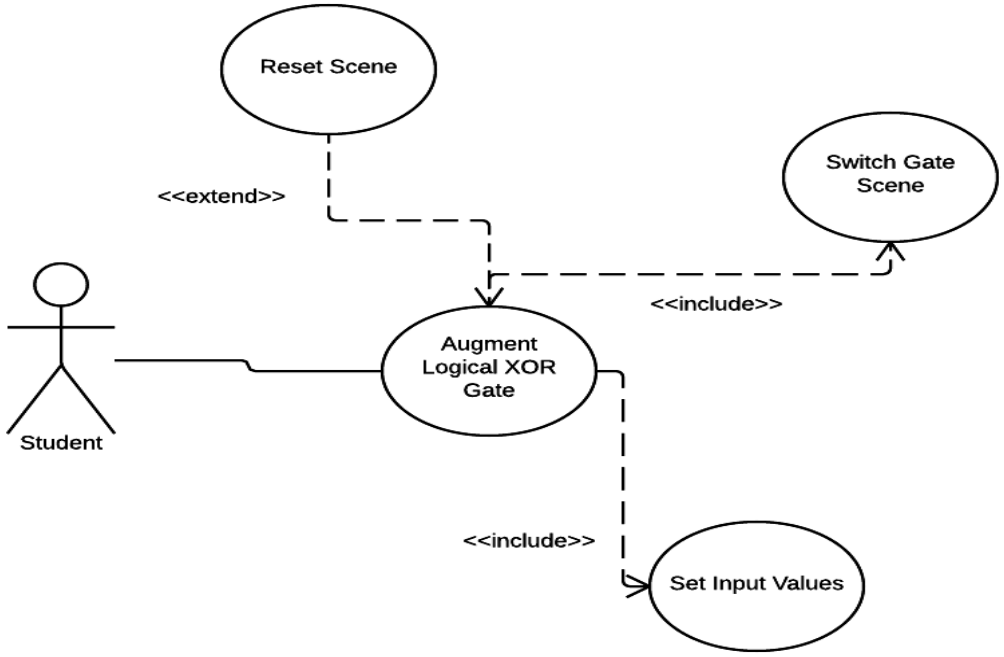


Fig. 4 Logical XOR Gate Use Case Diagram

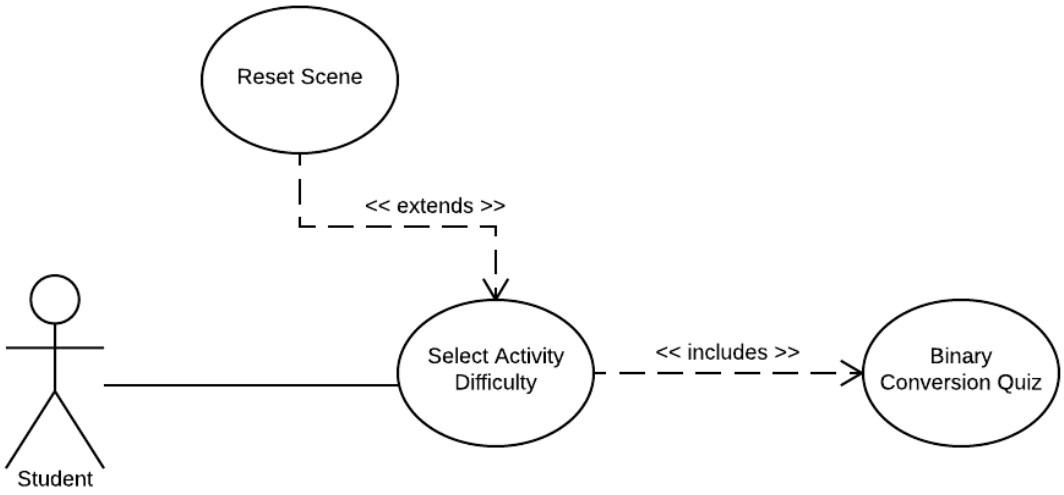


Fig. 5 Binary Conversion Menu Use Case

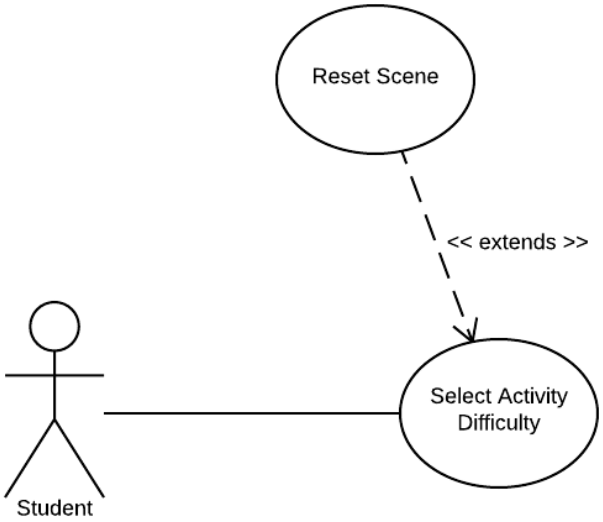


Fig. 6 Select Activity Difficulty Use Case Diagram

Sequence Diagrams

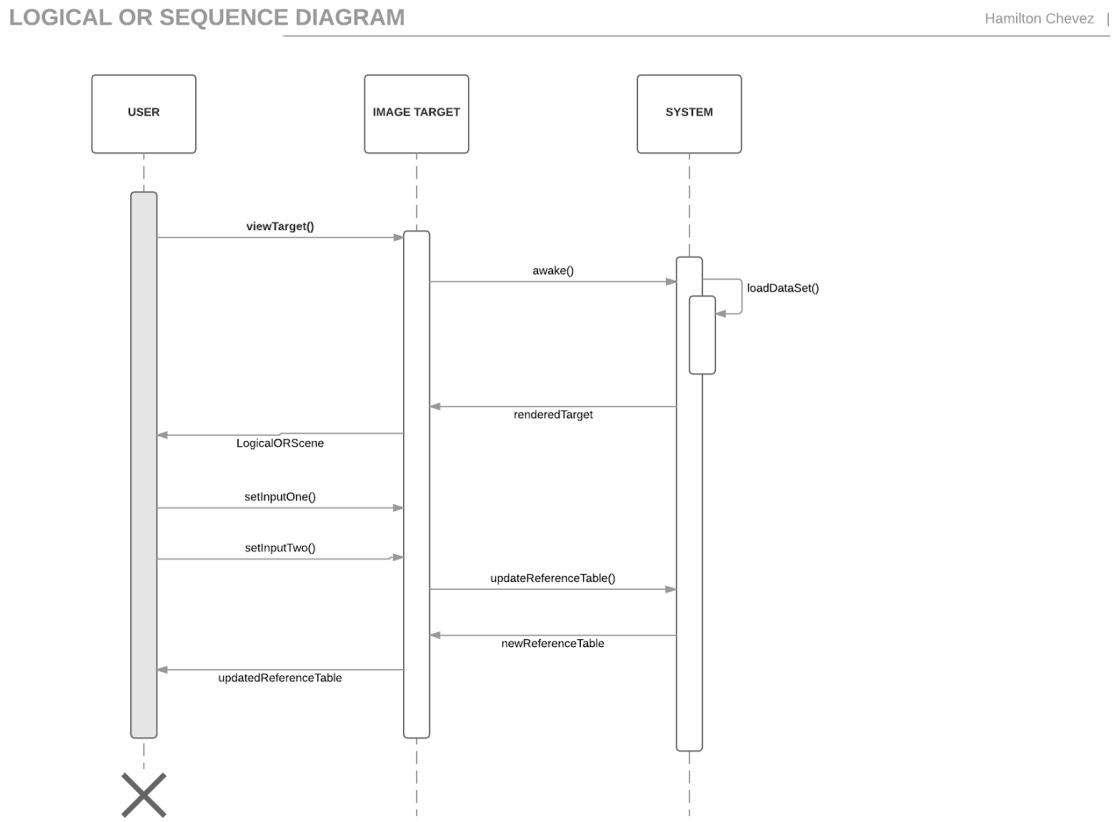


Fig. 7 Logical OR Sequence Diagram

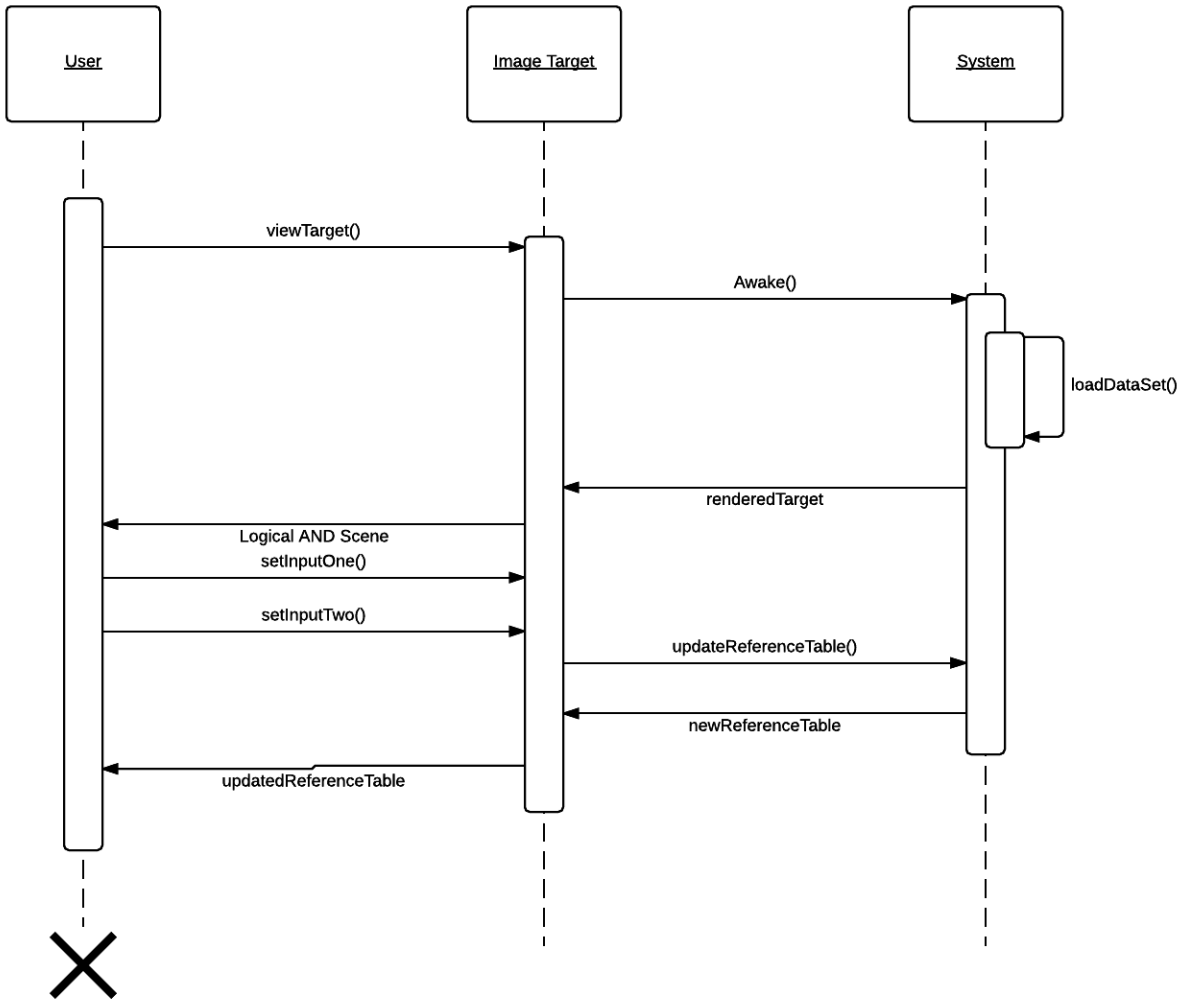


Fig 8. Logical AND Sequence Diagram

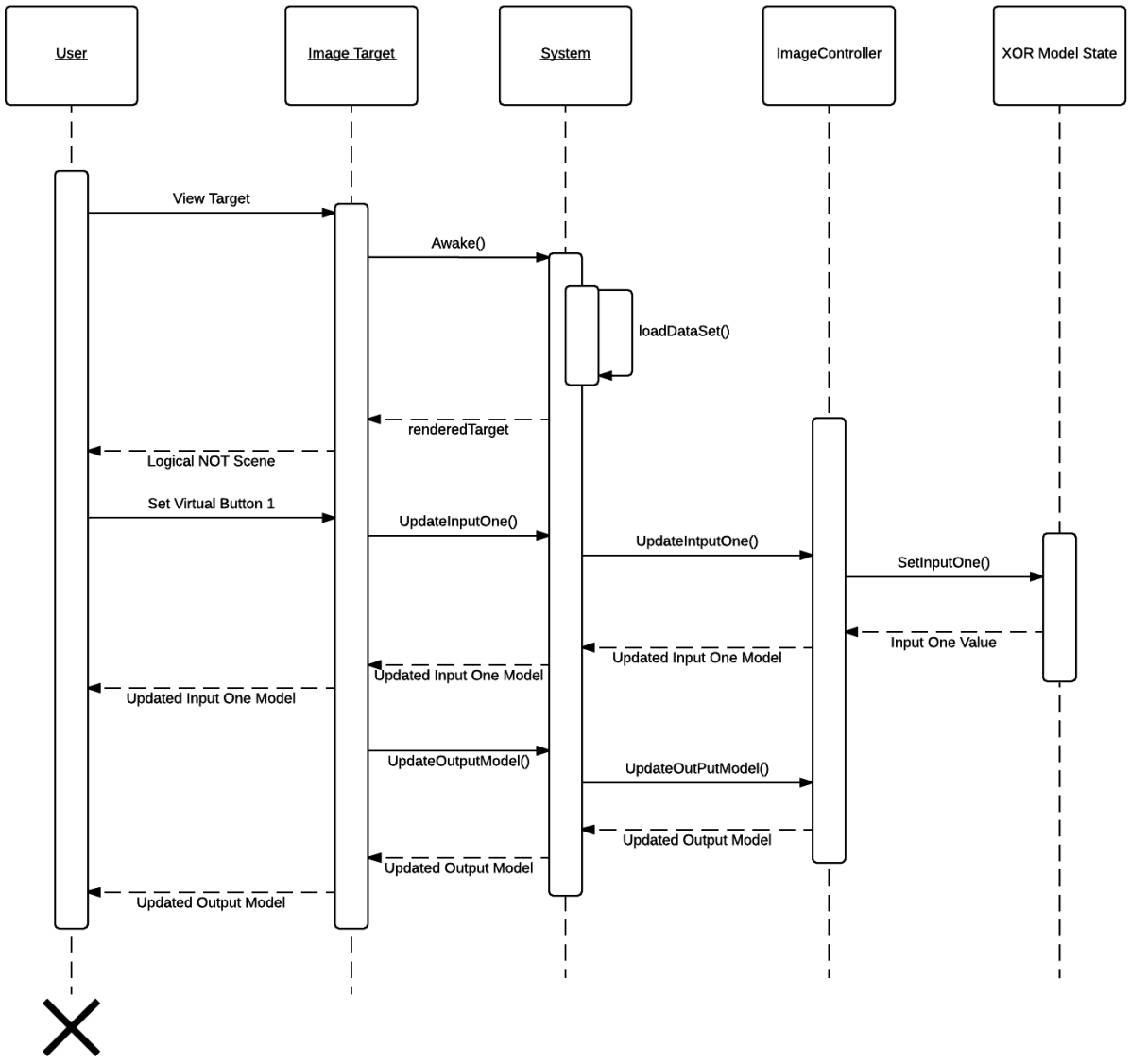


Fig. 9 Logical NOT Sequence Diagram

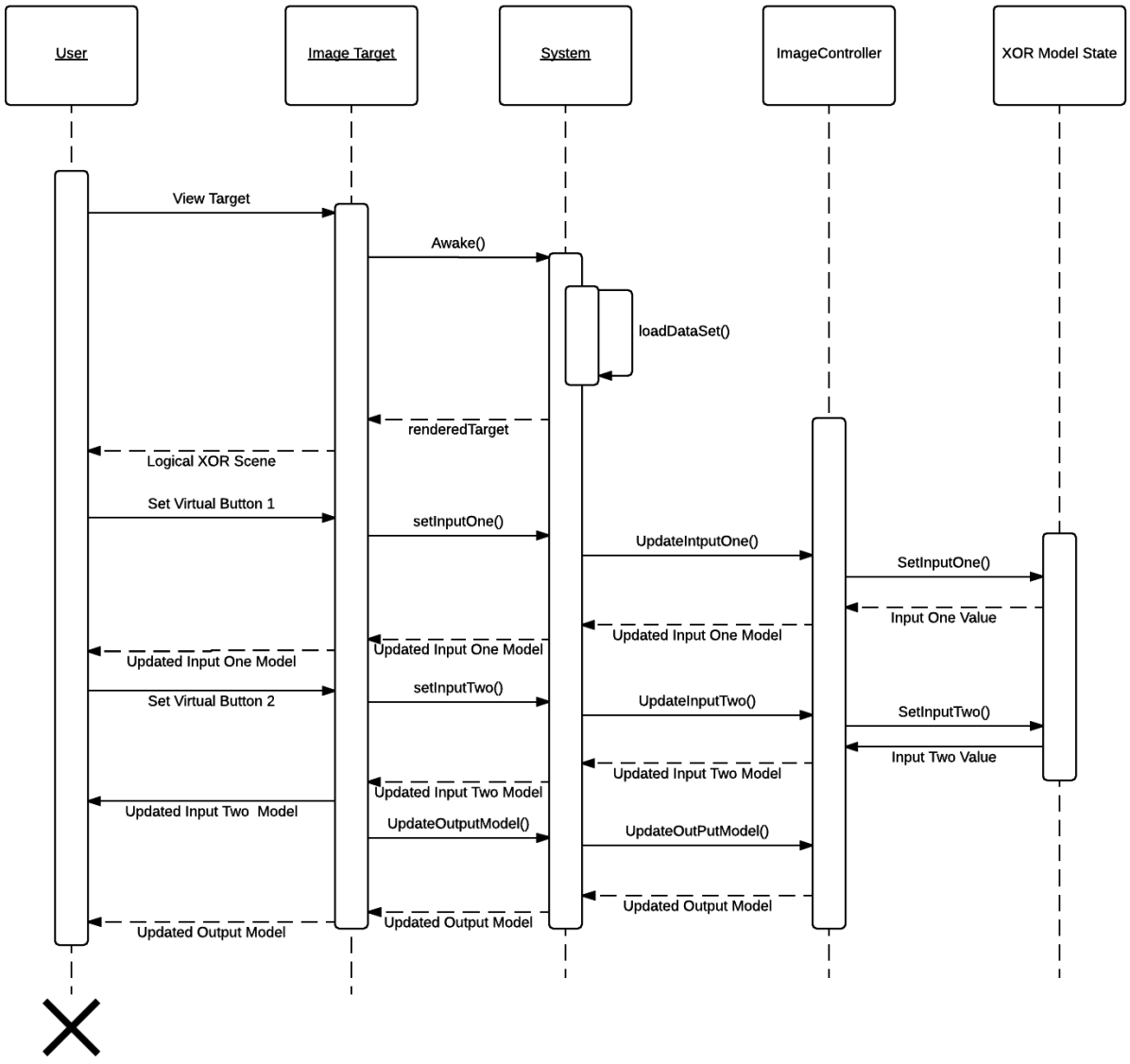


Fig.10 Logical XOR Sequence Diagram

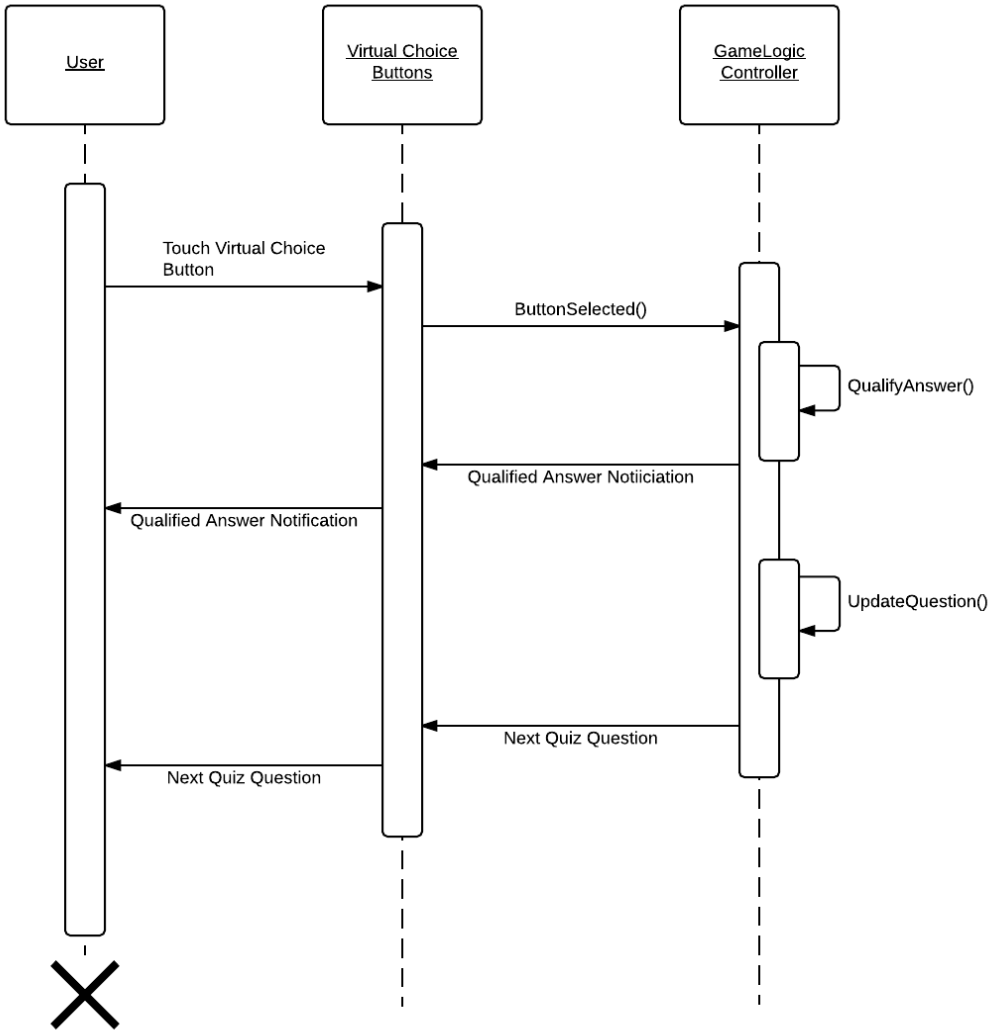


Fig. 11 Binary Conversion Activity Sequence Diagram

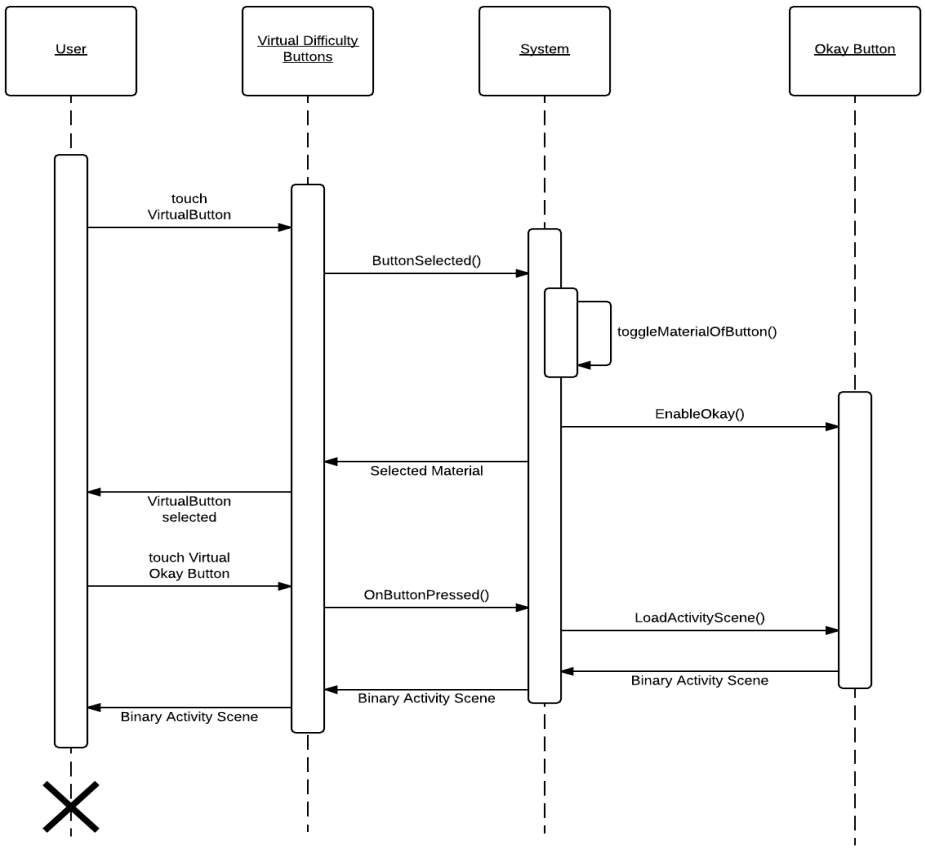


Fig. 12 Select Difficulty Sequence Diagram

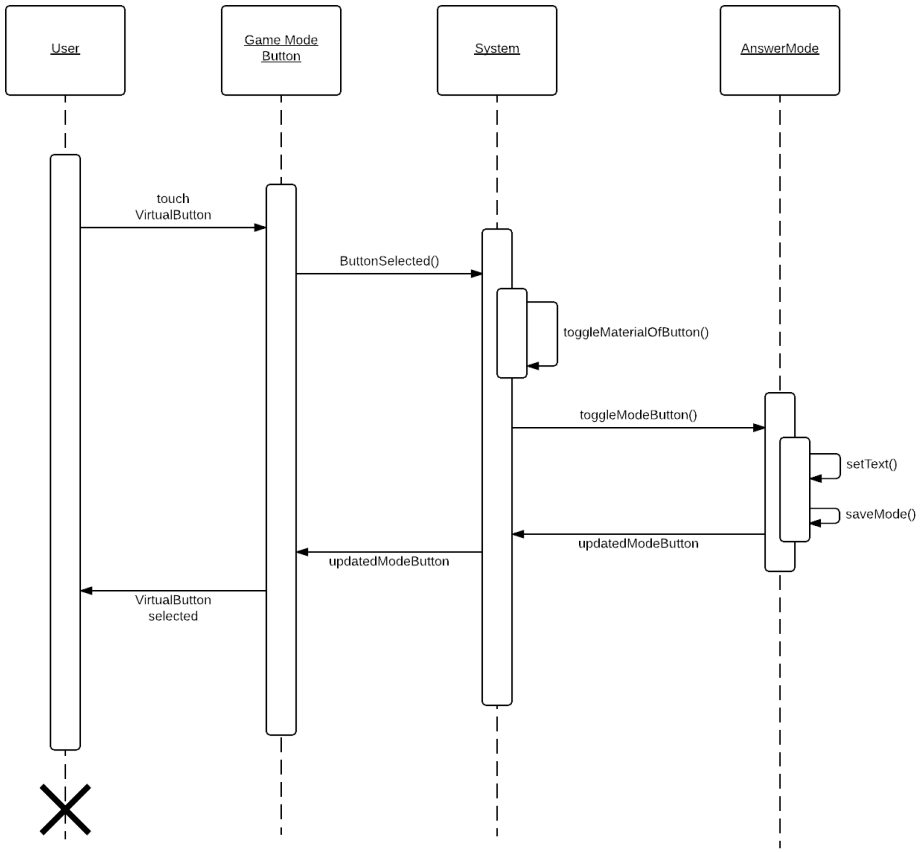


Fig. 13 Select Game Mode Sequence Diagram

Class Diagrams:

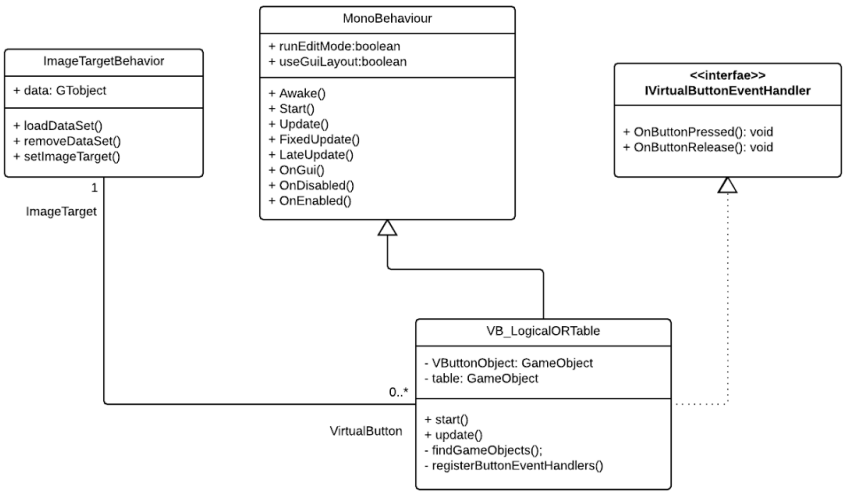


Fig. 14 Logical OR Class Diagram

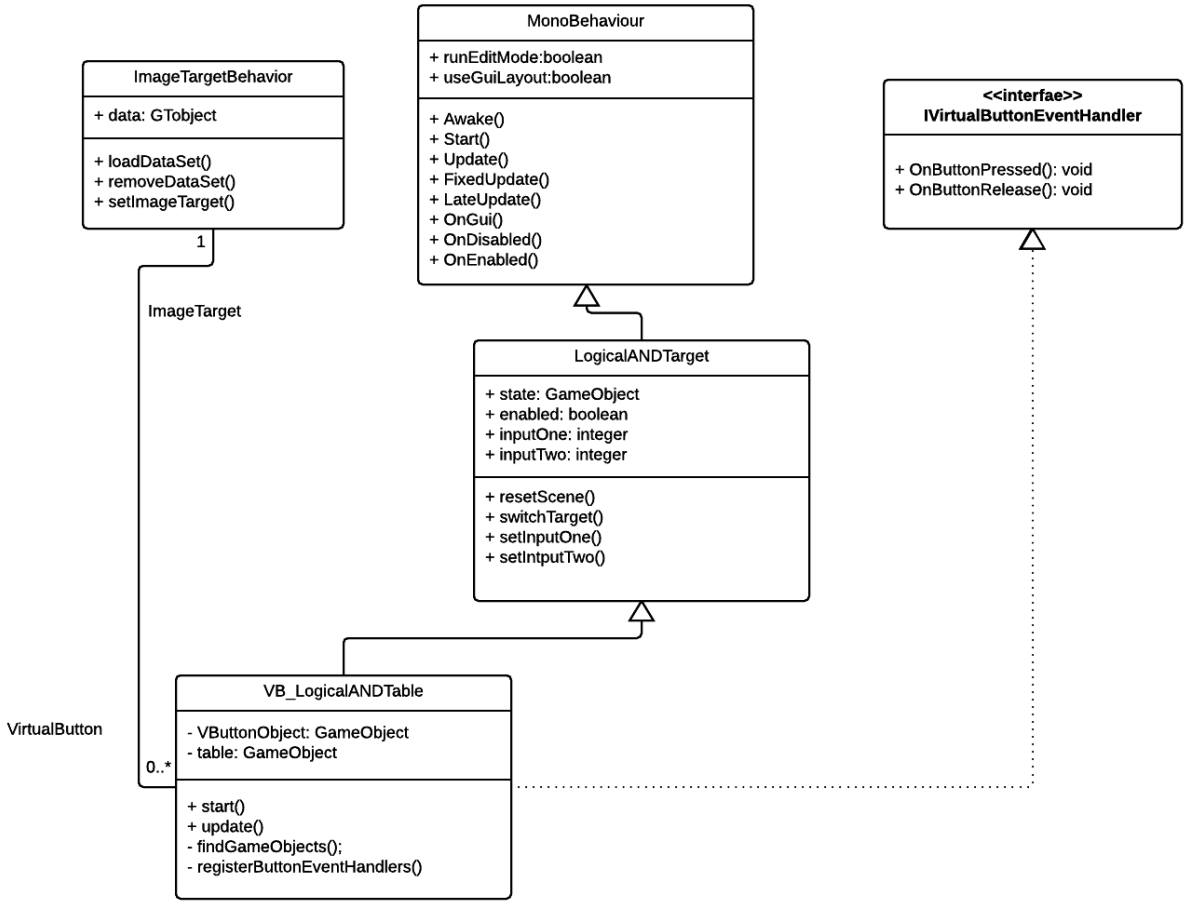


Fig. 15 Logical AND Class Diagram

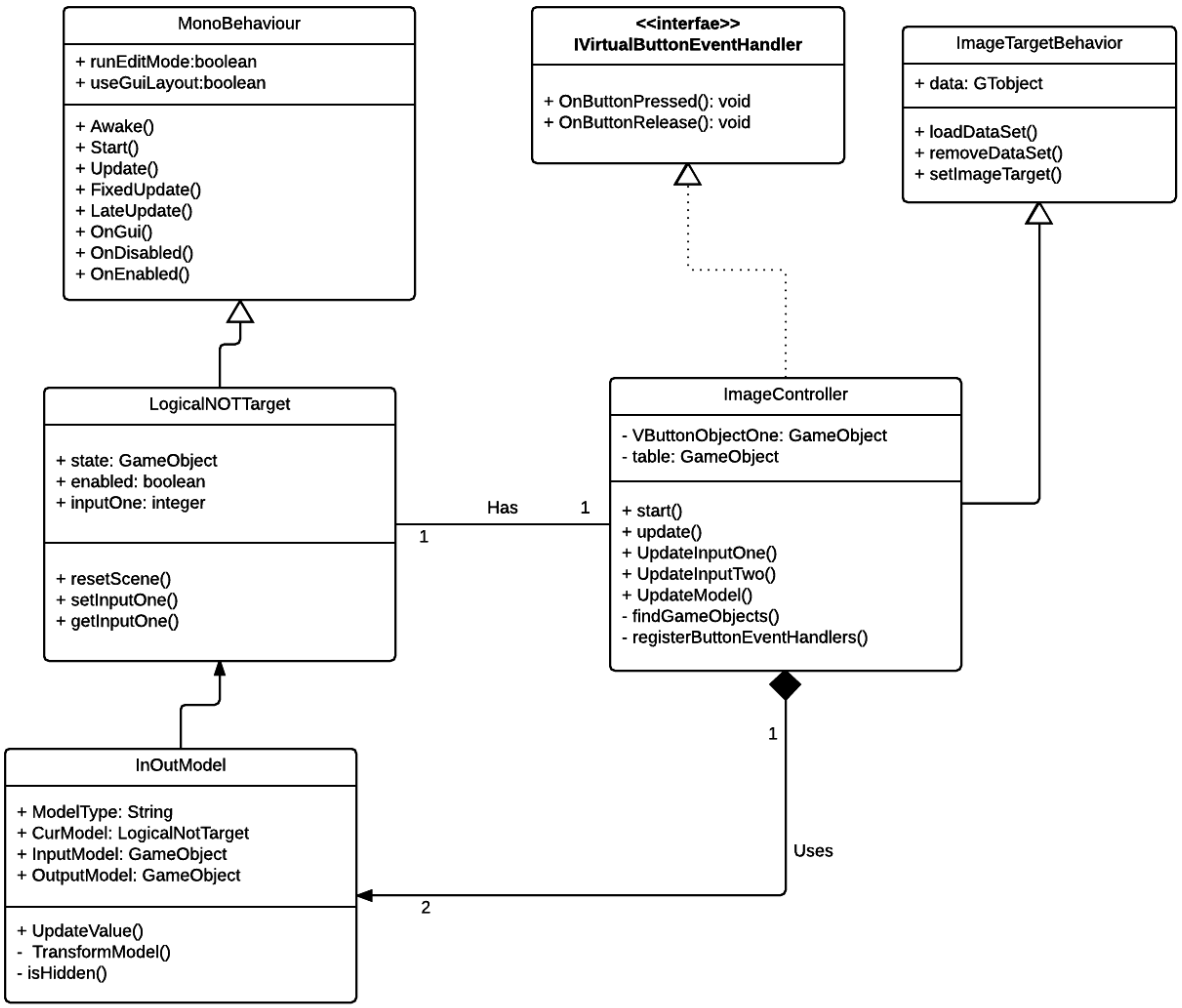


Fig. 16 Logical NOT Class Diagram

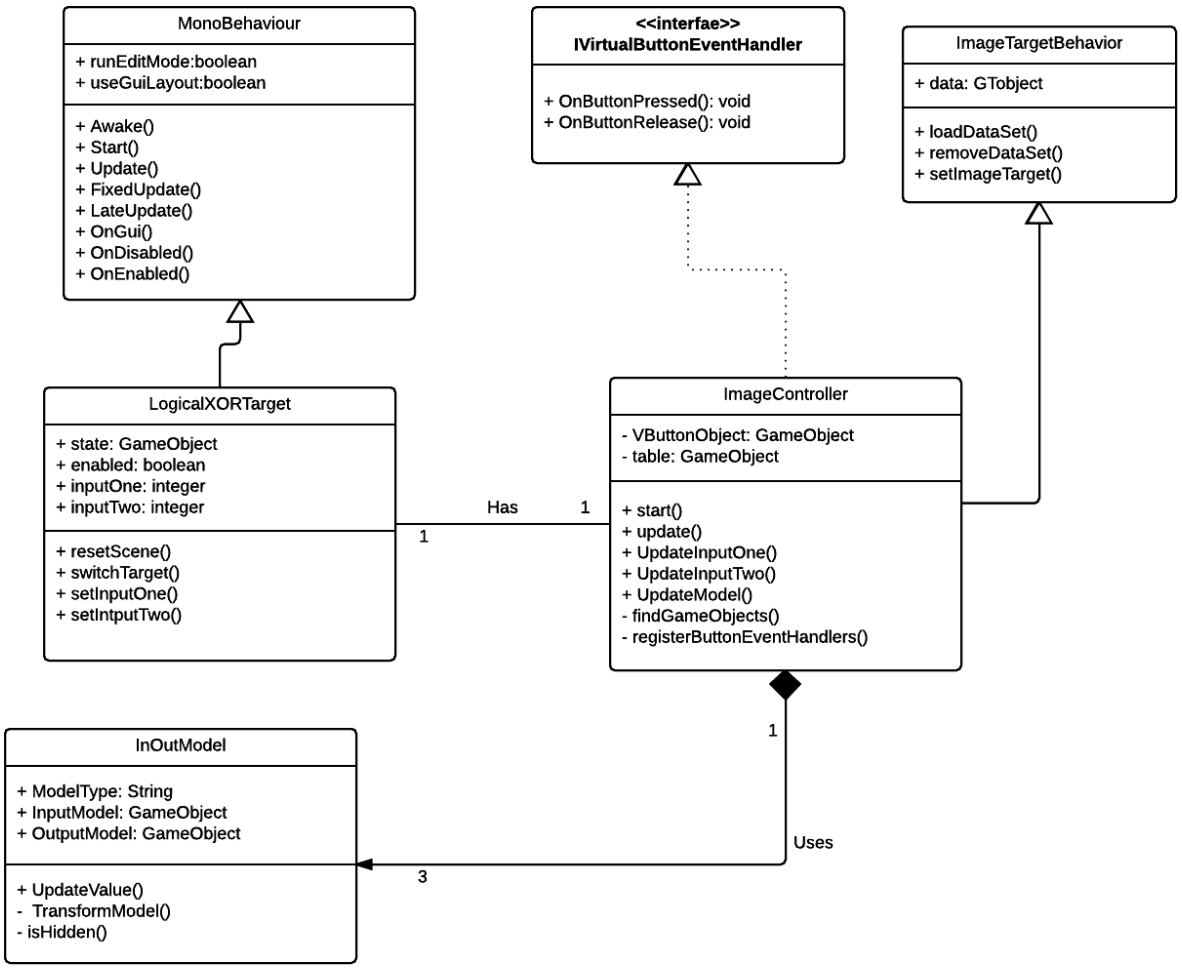


Fig. 17 Logical XOR Class Diagram

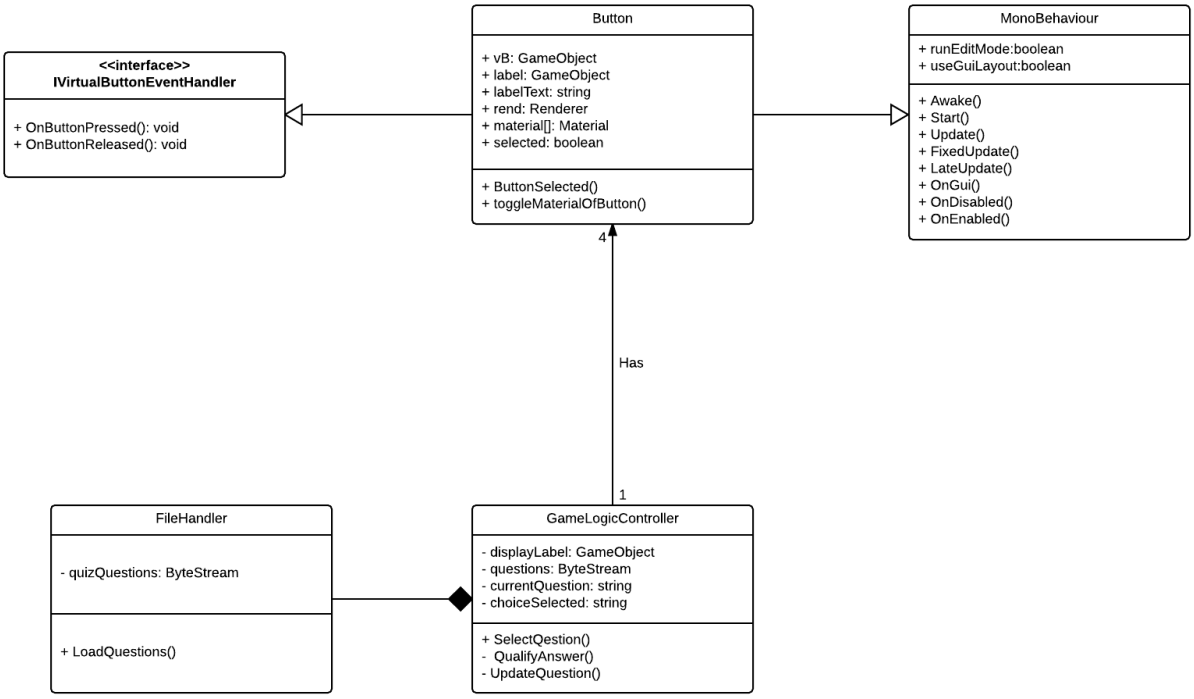


Fig. 18 Binary Conversion Class Diagram

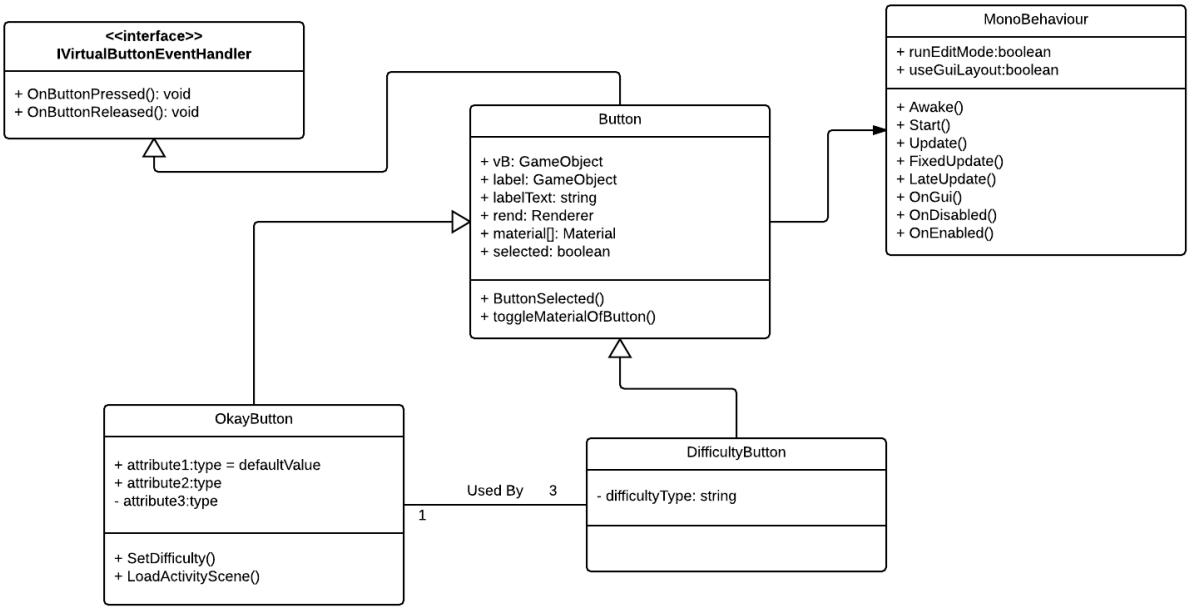


Fig. 19 Select Difficulty Class Diagram

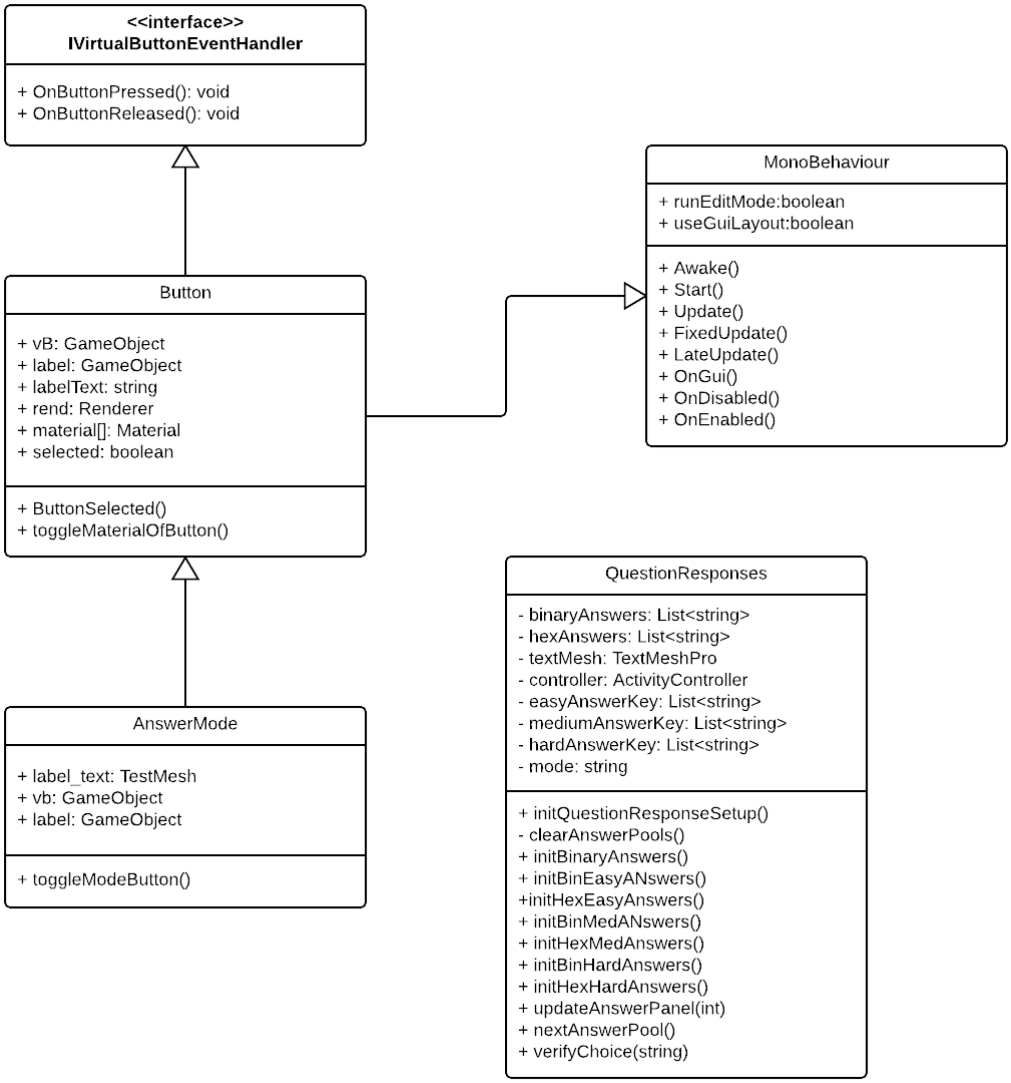


Fig. 20 Select Game Mode Class Diagram

Code Adventures Team:

Use Case Diagrams

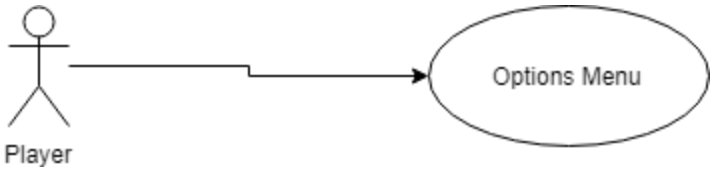


Fig 1 - Options Menu Use Case Diagram

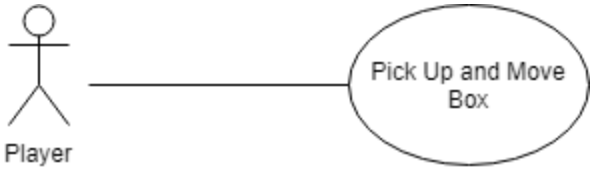


Fig 2 - Add an Indicator Use Case diagram

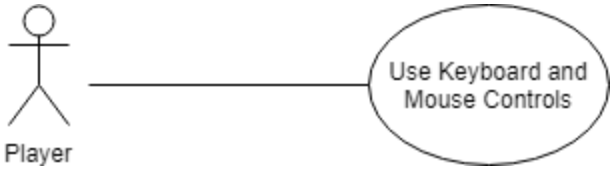


Fig 3 - Use Keyboard and Mouse Controls Use Case Diagram

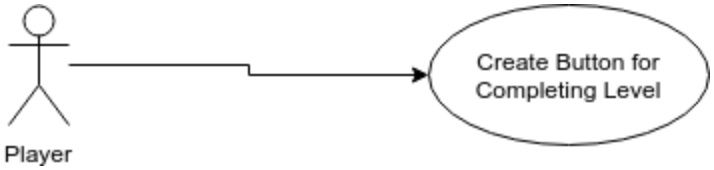


Fig 4 - Create Button for Completing Level Use Case Diagram

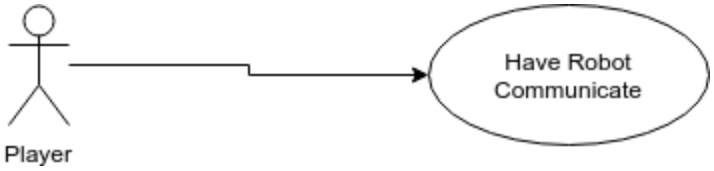


Fig 5 - Have Robot Communicate Use Case Diagram

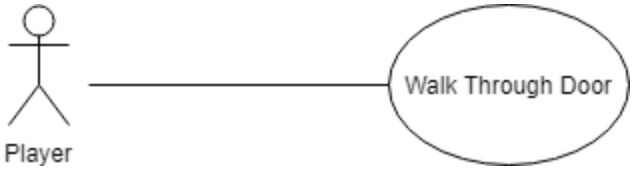


Fig 6 - Create Lightning Door Use Case Diagram

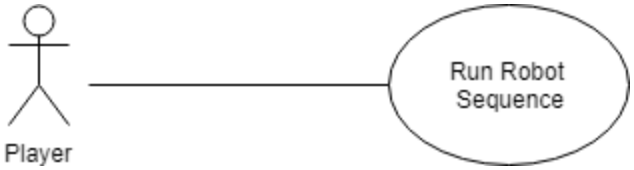


Fig 7 - Prevent Robot From Going Through Walls Use Case Diagram

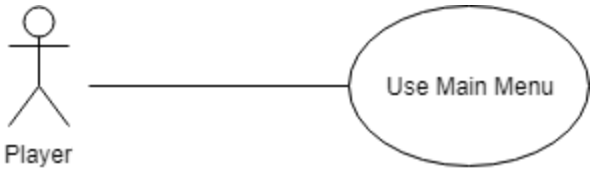


Fig 8 - Add a main menu Use Case Diagram

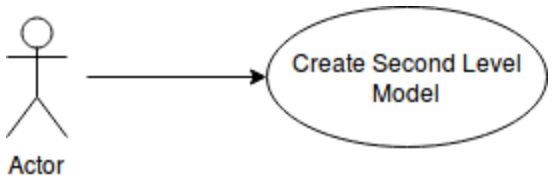


Fig 9 - Create Second Level Model

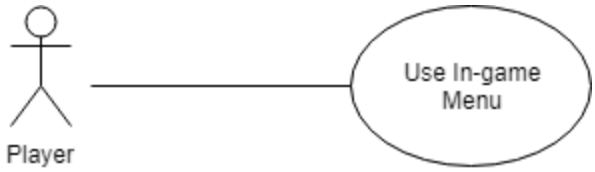


Fig 10 - Create an In-Game Menu

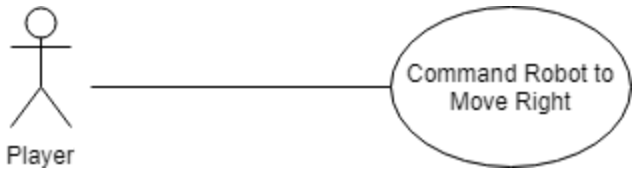


Fig 11 - Add Right Turn Module Use Case Diagram

Sequence Diagrams

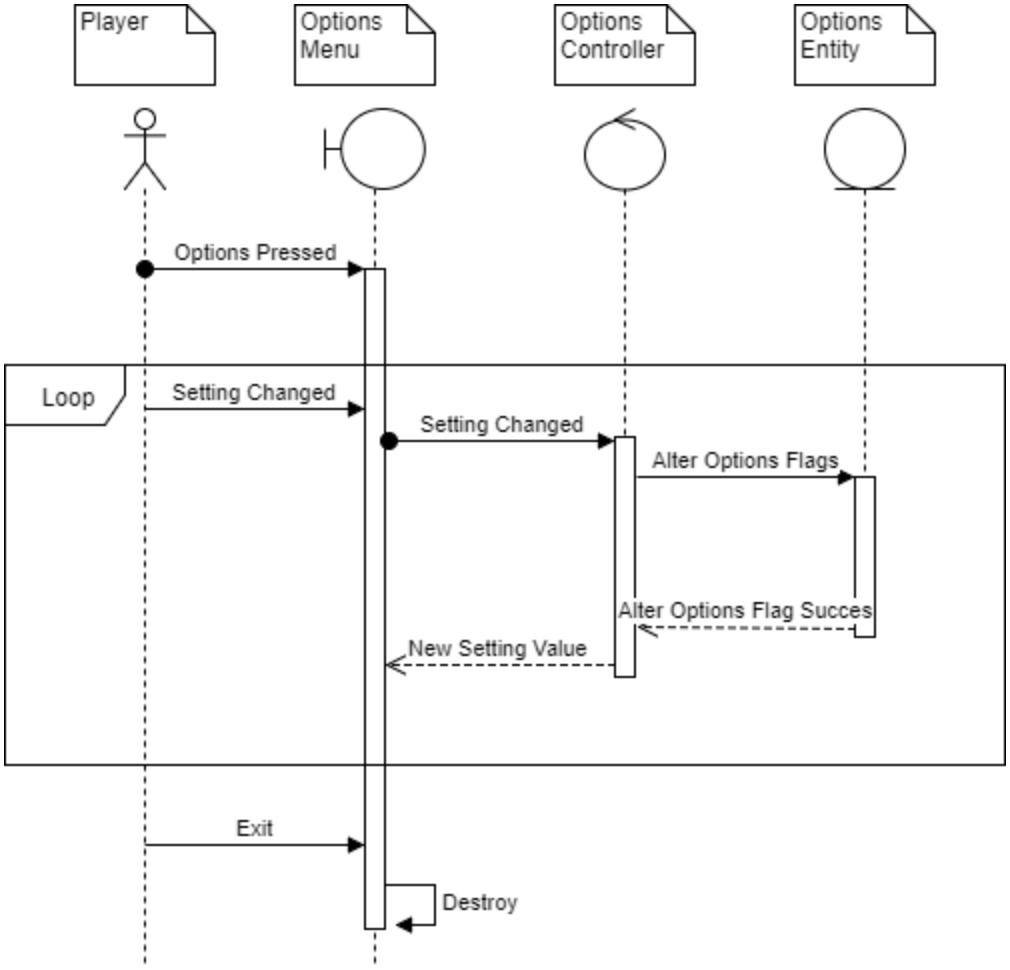


Fig 1 - Options Menu Sequence Diagram

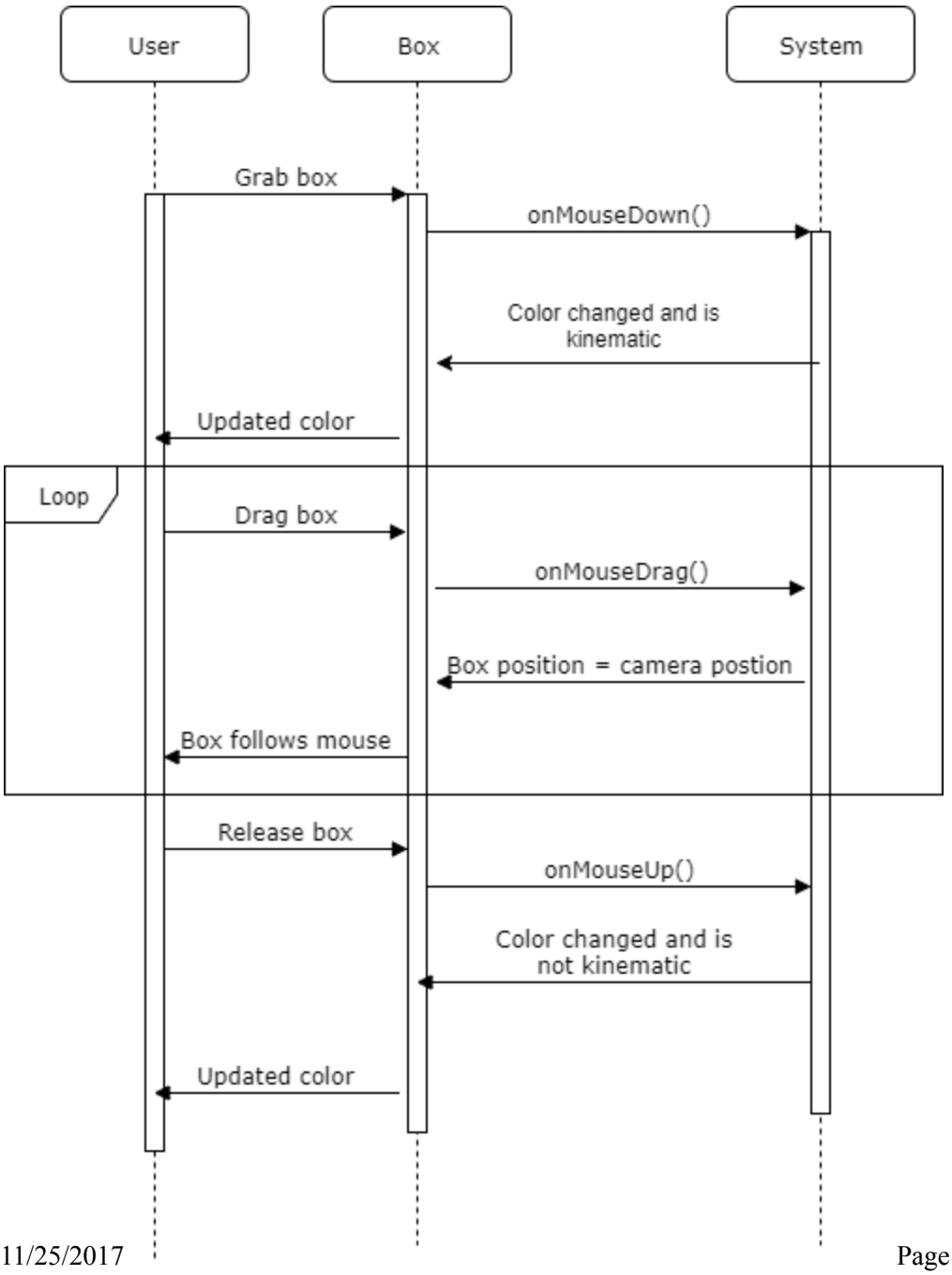

User
Box
System
Grab box
onMouseDown()
Color changed and is kinematic
Updated color
Loop
Drag box
onMouseDrag()
Box position = camera postion
Box follows mouse
Release box
onMouseUp()
Color changed and is not kinematic
Updated color

Fig 2 - Add an indicator Sequence Diagram

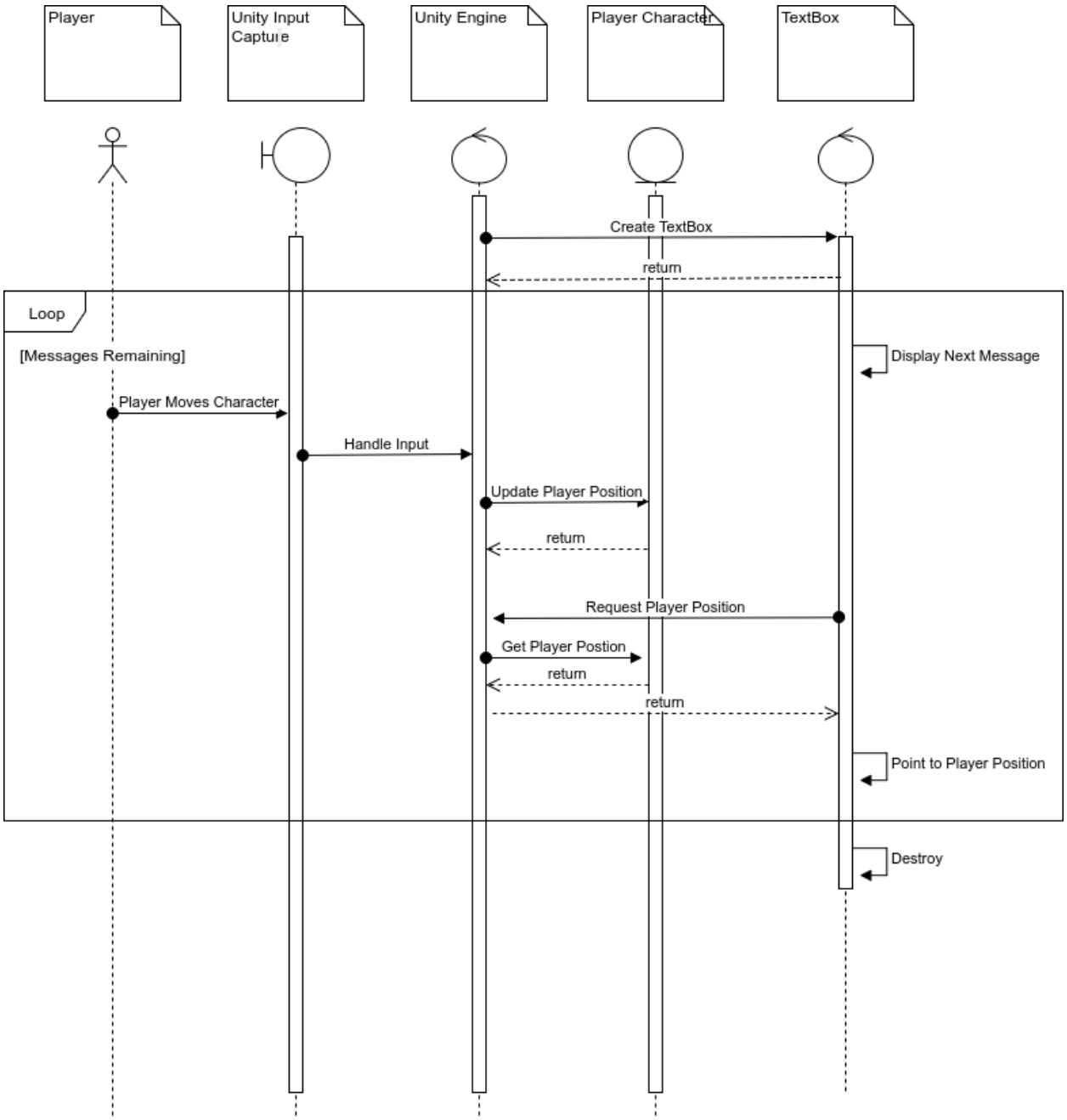

Player
Unity Input Capture
Unity Engine
Player Character
TextBox
Create TextBox
return
Loop
[Messages Remaining]
Display Next Message
Player Moves Character
Handle Input
Update Player Position
return
Request Player Position
Get Player Postion
return
return
Point to Player Position
Destroy

Fig 3 - Have Robot Communicate Sequence Diagram

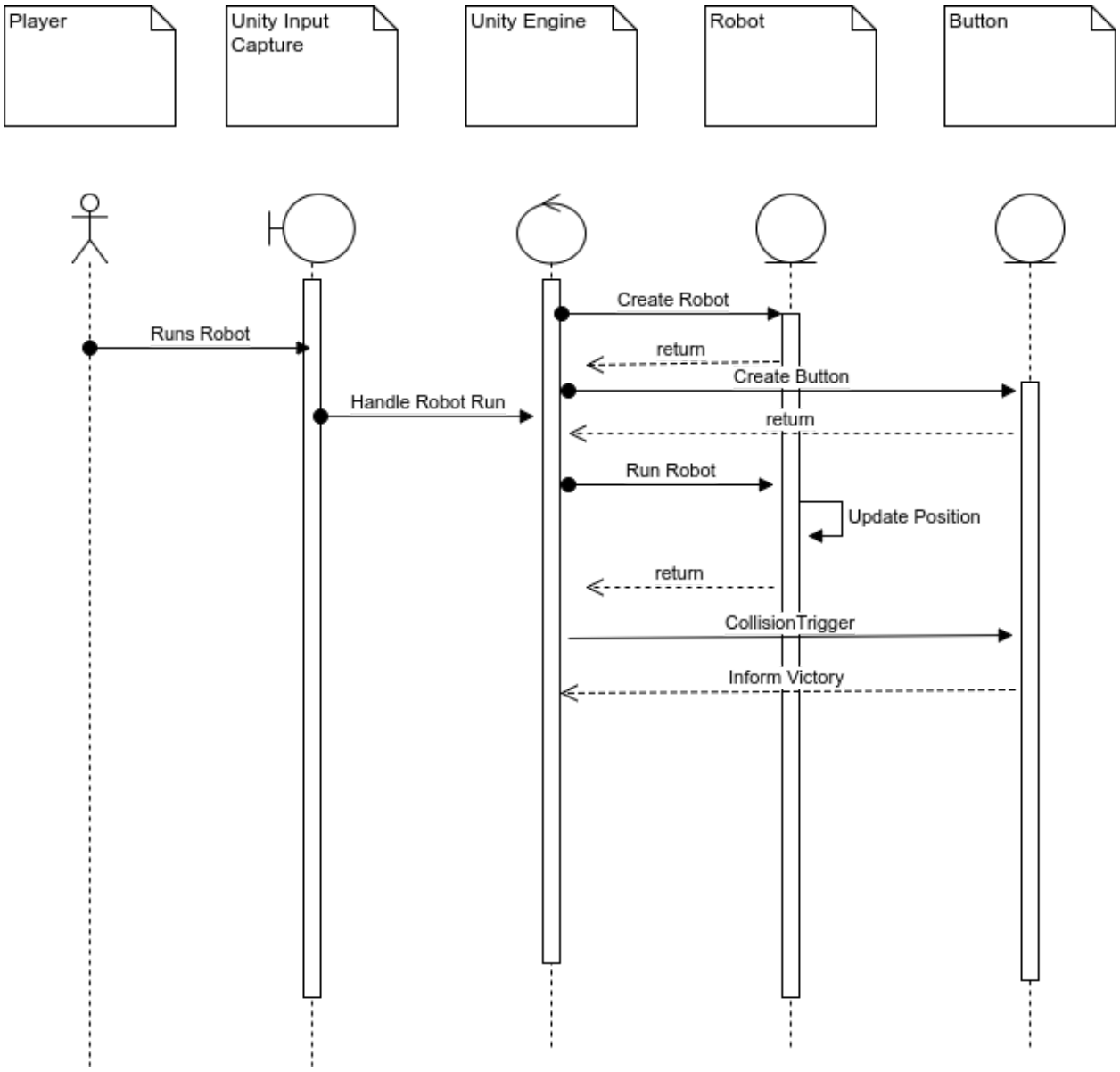

Player
Unity Input Capture
Unity Engine
Robot
Button
Runs Robot
Handle Robot Run
Create Robot
return
Create Button
return
Run Robot
Update Position
return
CollisionTrigger
Inform Victory

Fig 4 - Create Button For Completing Level Sequence Diagram

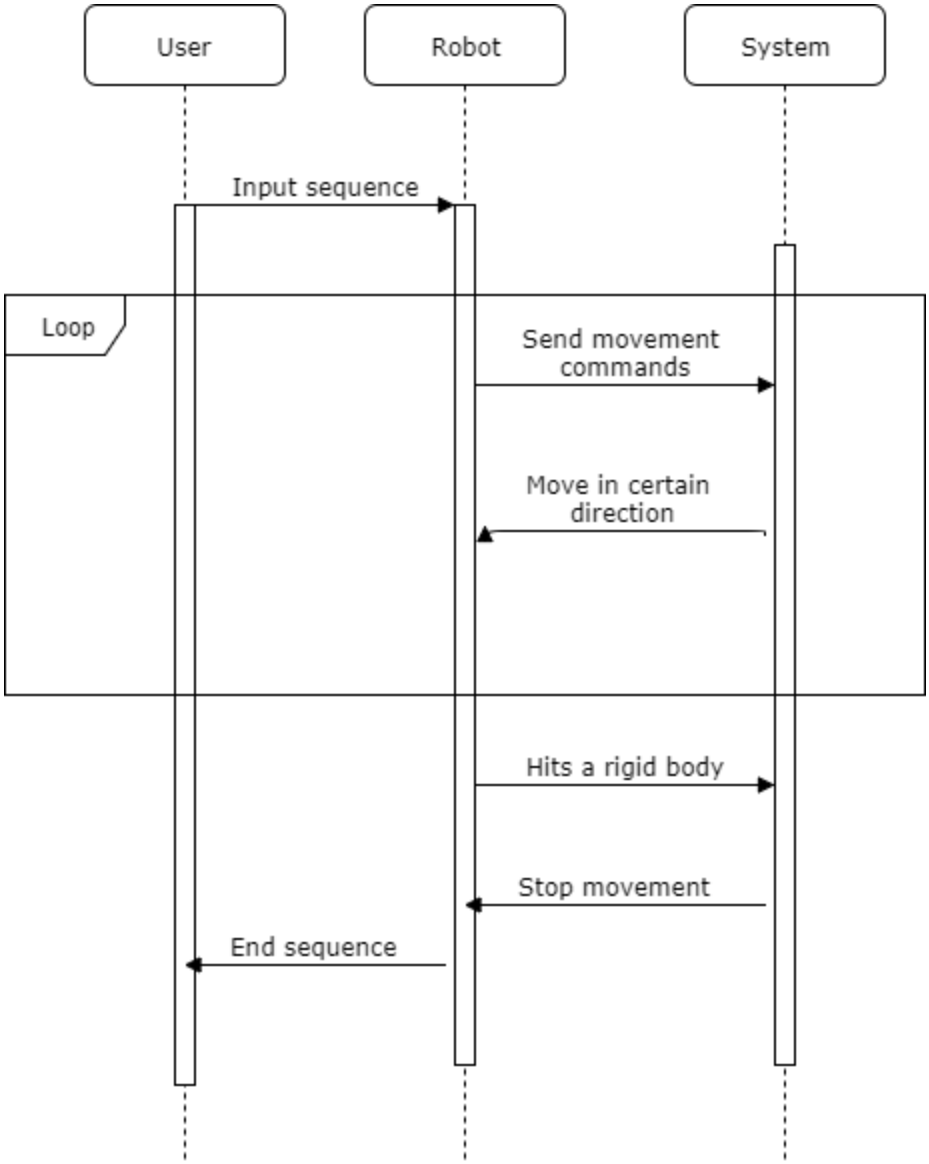


Fig 5 - Prevent Robot from going through walls sequence Diagrams

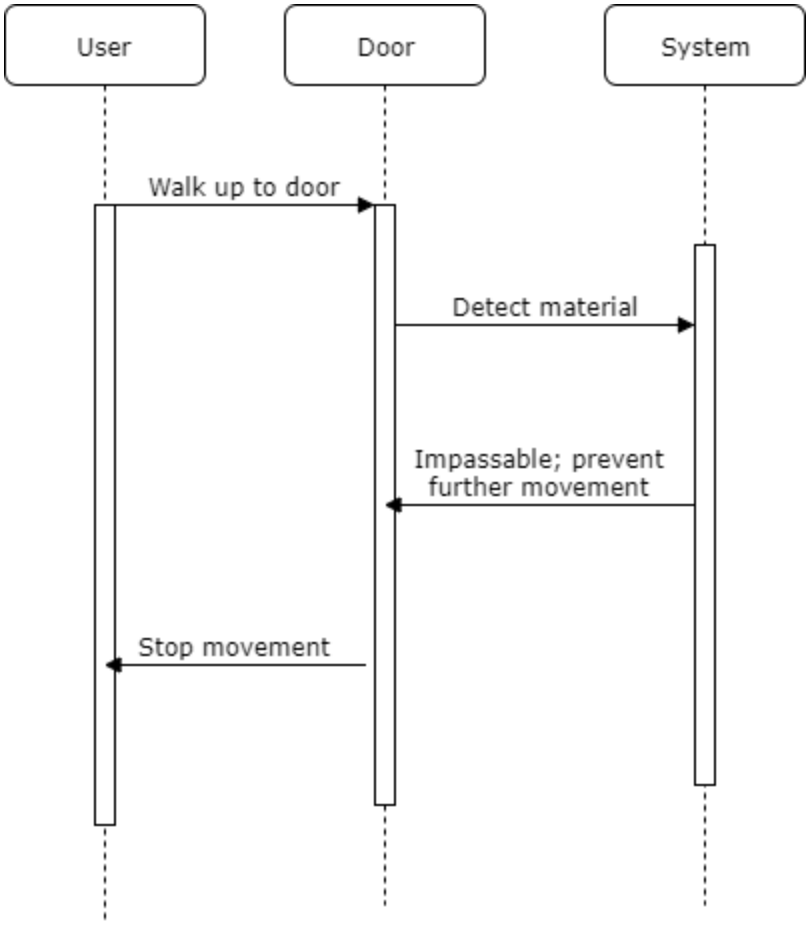


Fig 6 - Create Lightning Door

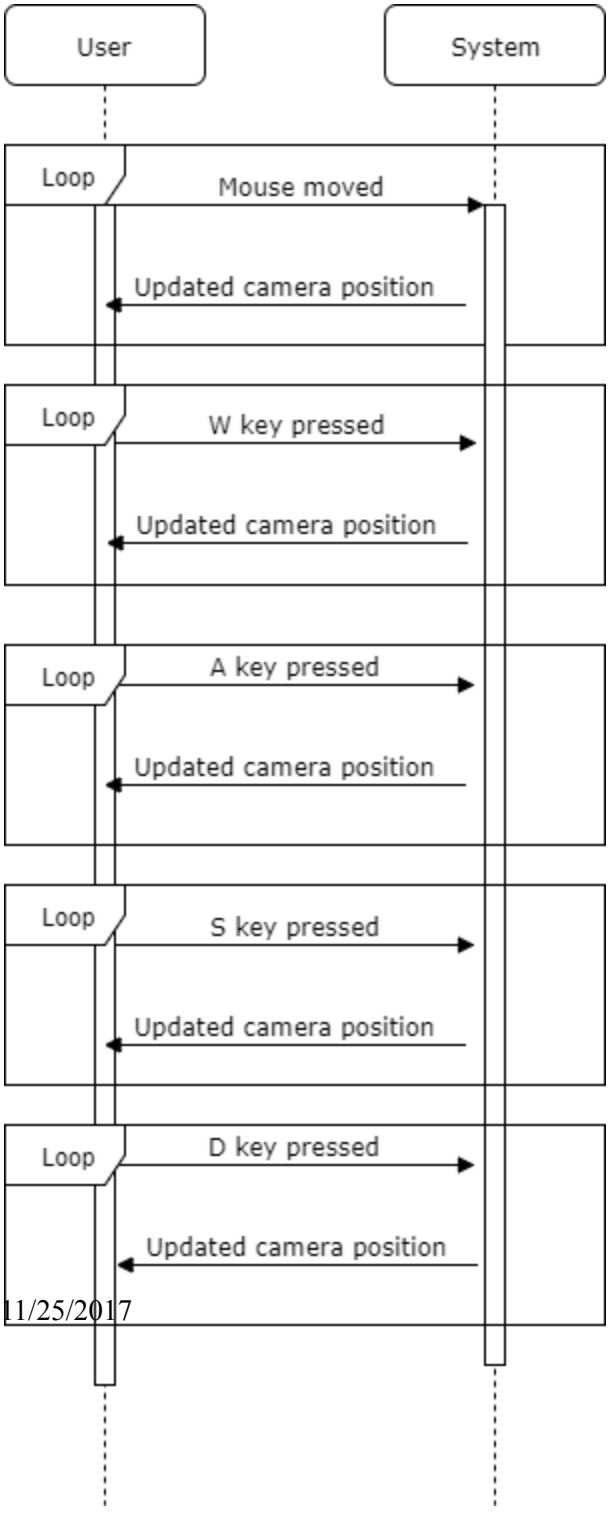

User
System
Loop
Mouse moved
Updated camera position
Loop
W key pressed
Updated camera position
Loop
A key pressed
Updated camera position
Loop
S key pressed
Updated camera position
Loop
D key pressed
Updated camera position

Fig 3 - Use Keyboard and Mouse Controls Sequence Diagram

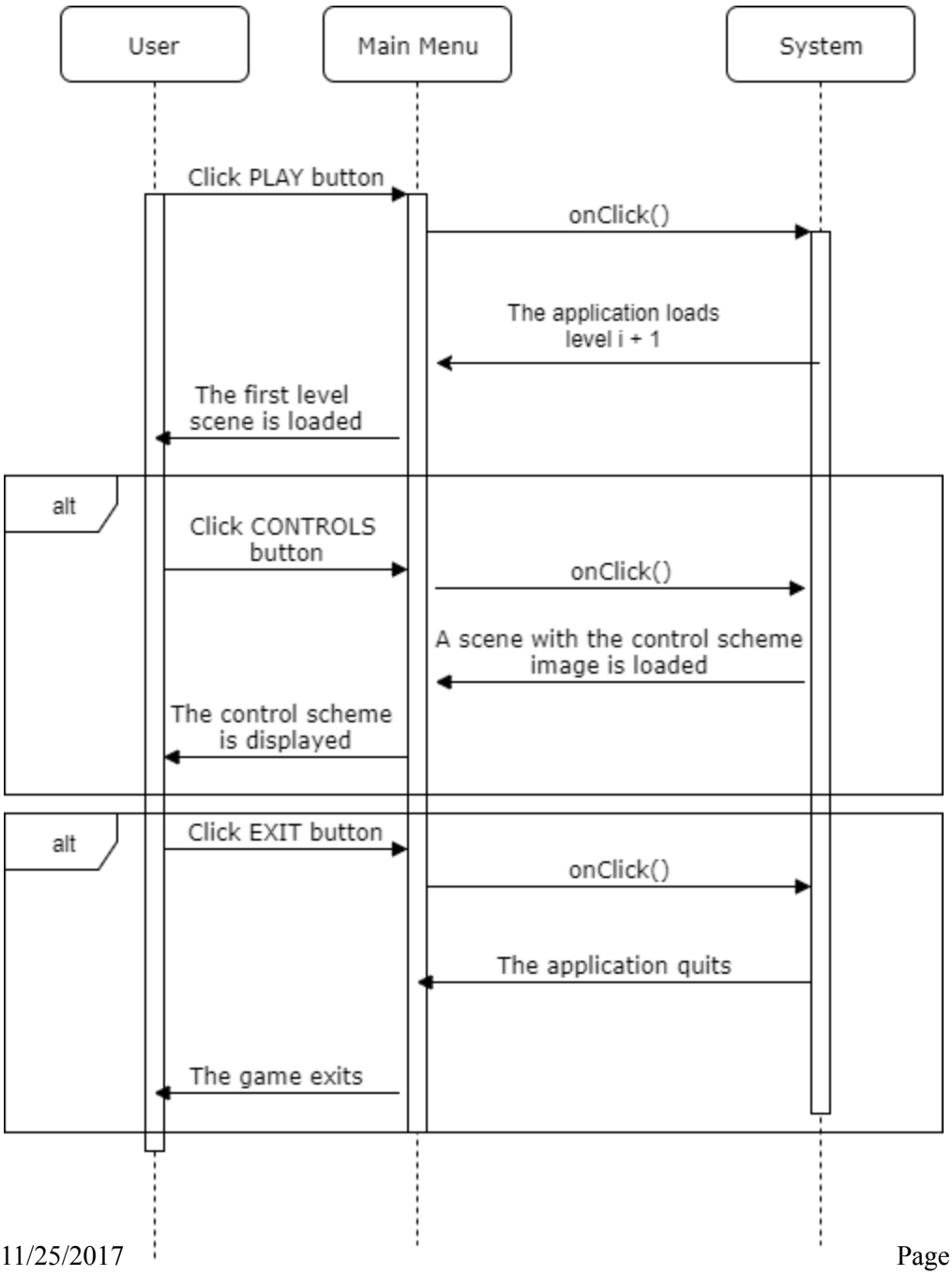

User
Main Menu
System
Click PLAY button
onClick()
The application loads level i + 1
The first level scene is loaded
alt
Click CONTROLS button
onClick()
A scene with the control scheme image is loaded
The control scheme is displayed
alt
Click EXIT button
onClick()
The application quits
The game exits

Fig 7 - Add a Main Menu Sequence Diagram

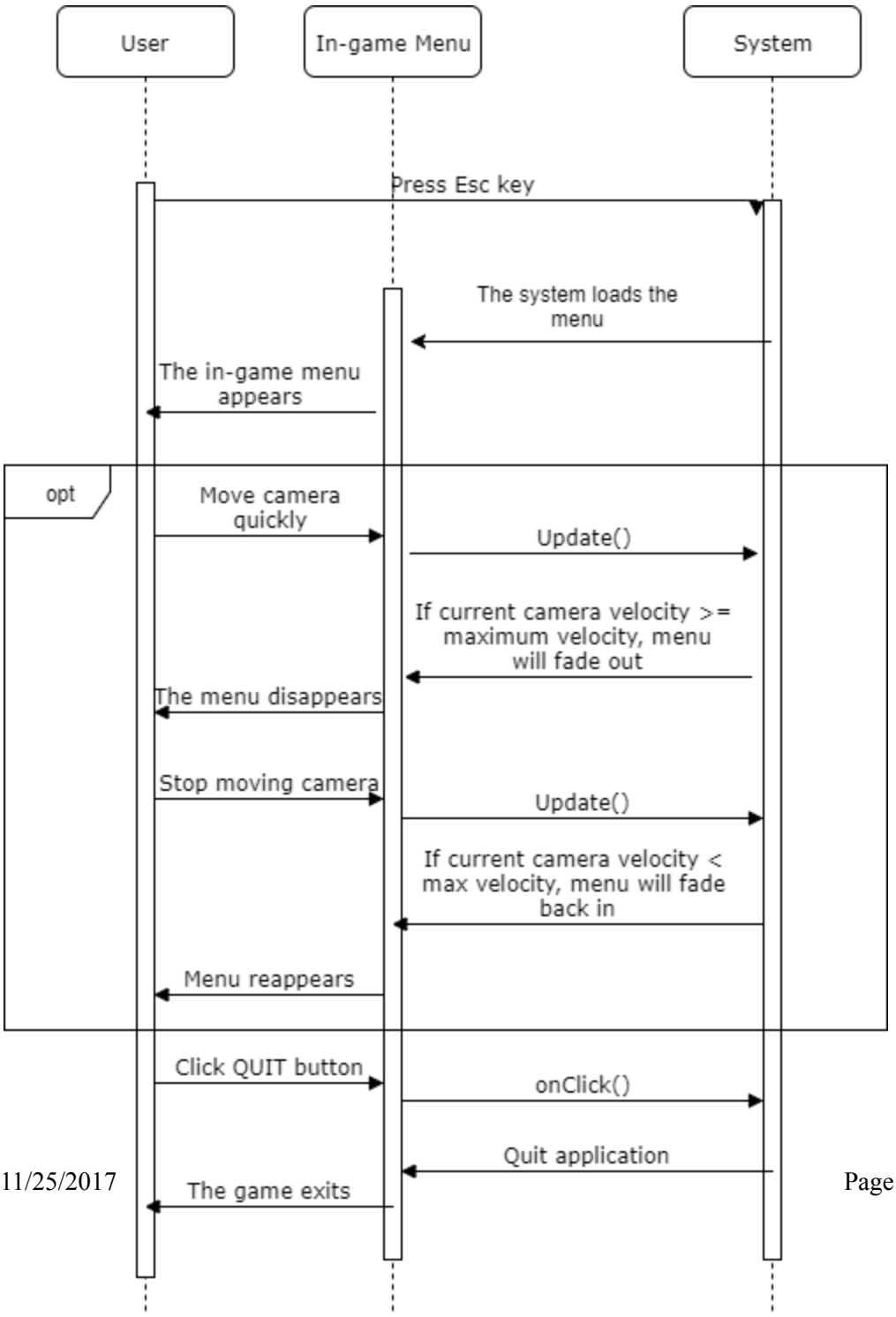

User
In-game Menu
System
Press Esc key
The system loads the menu
The in-game menu appears
opt
Move camera quickly
Update()
If current camera velocity >= maximum velocity, menu will fade out
The menu disappears
Stop moving camera
Update()
If current camera velocity < max velocity, menu will fade back in
Menu reappears
Click QUIT button
onClick()
Quit application
The game exits

Fig 8 - Add an in-game Menu Sequence Diagram

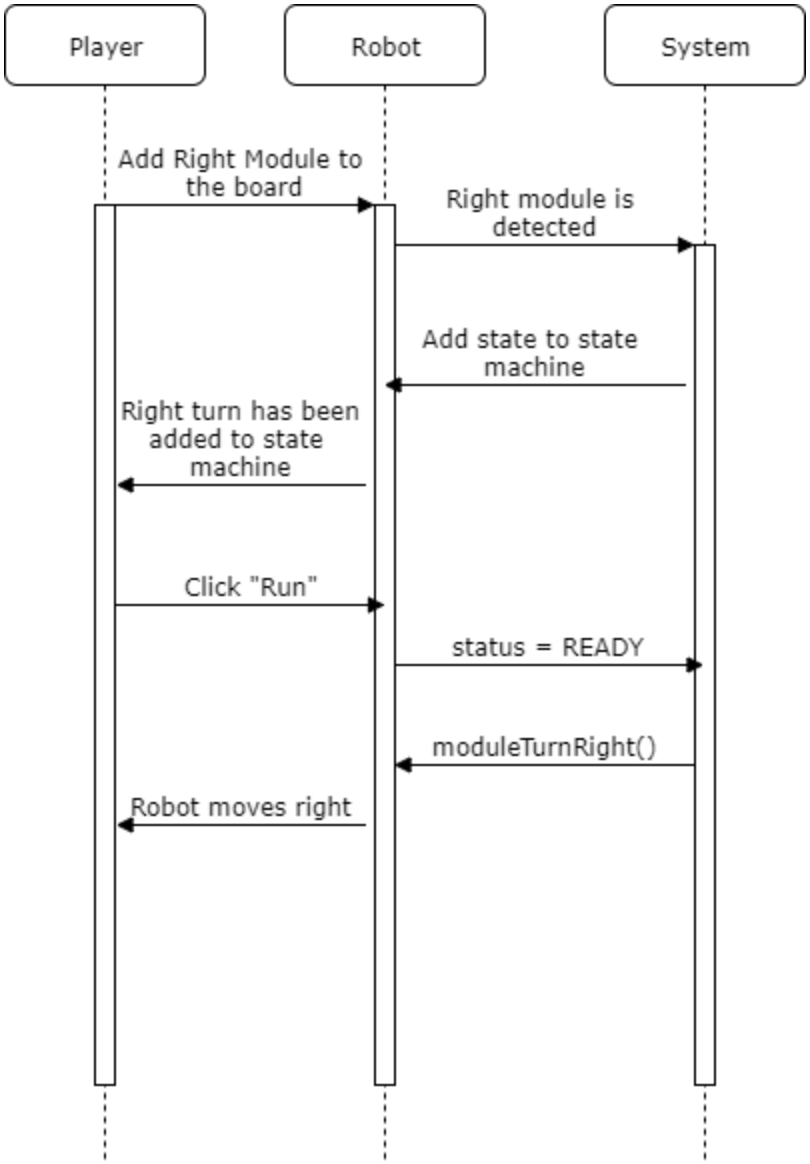


Fig 9 - Add Right Turn Module Sequence Diagram

Class Diagrams

Fig 1 - Options Menu Class Diagram

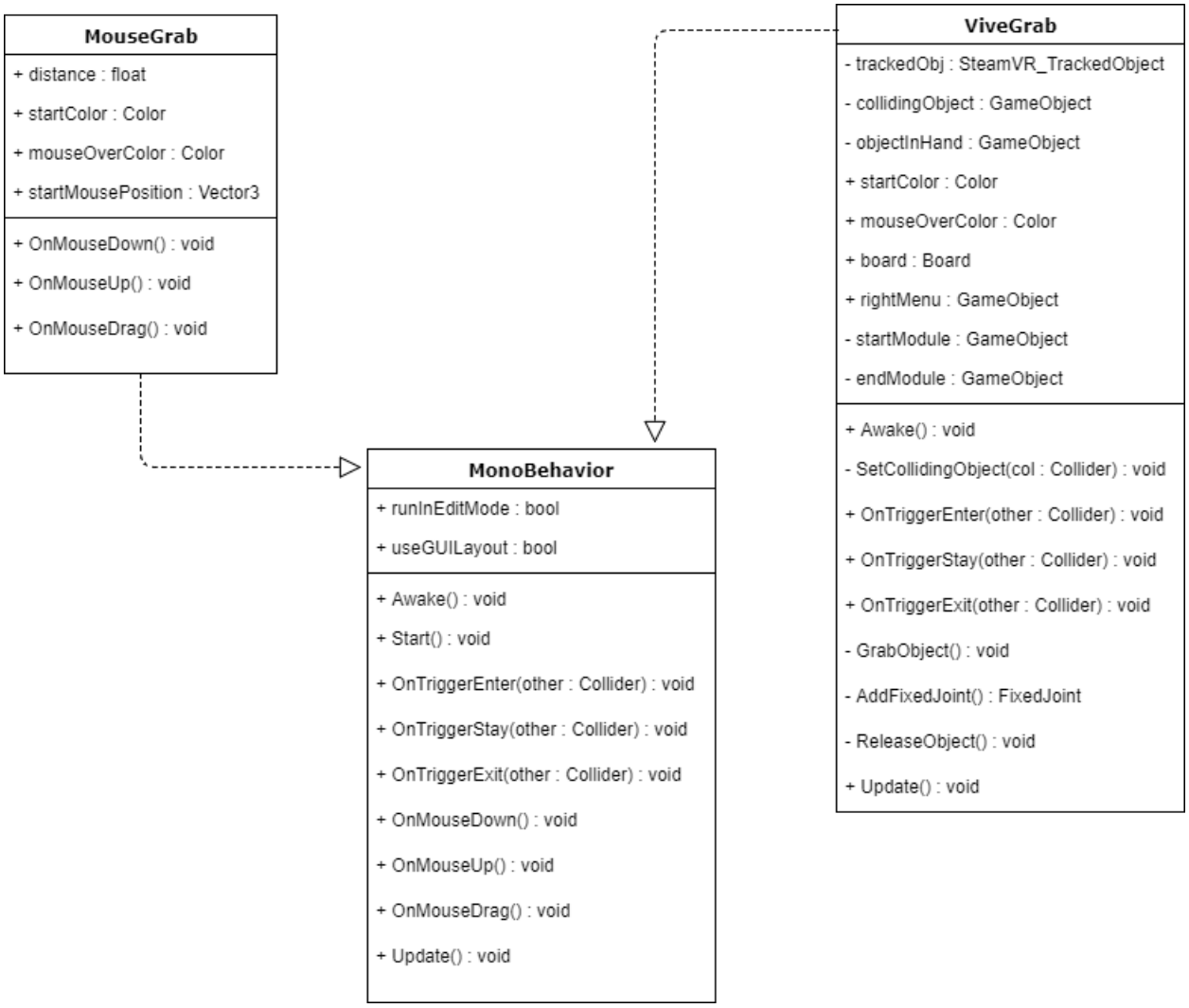


Fig 2 - Add an Indicator Class Diagram

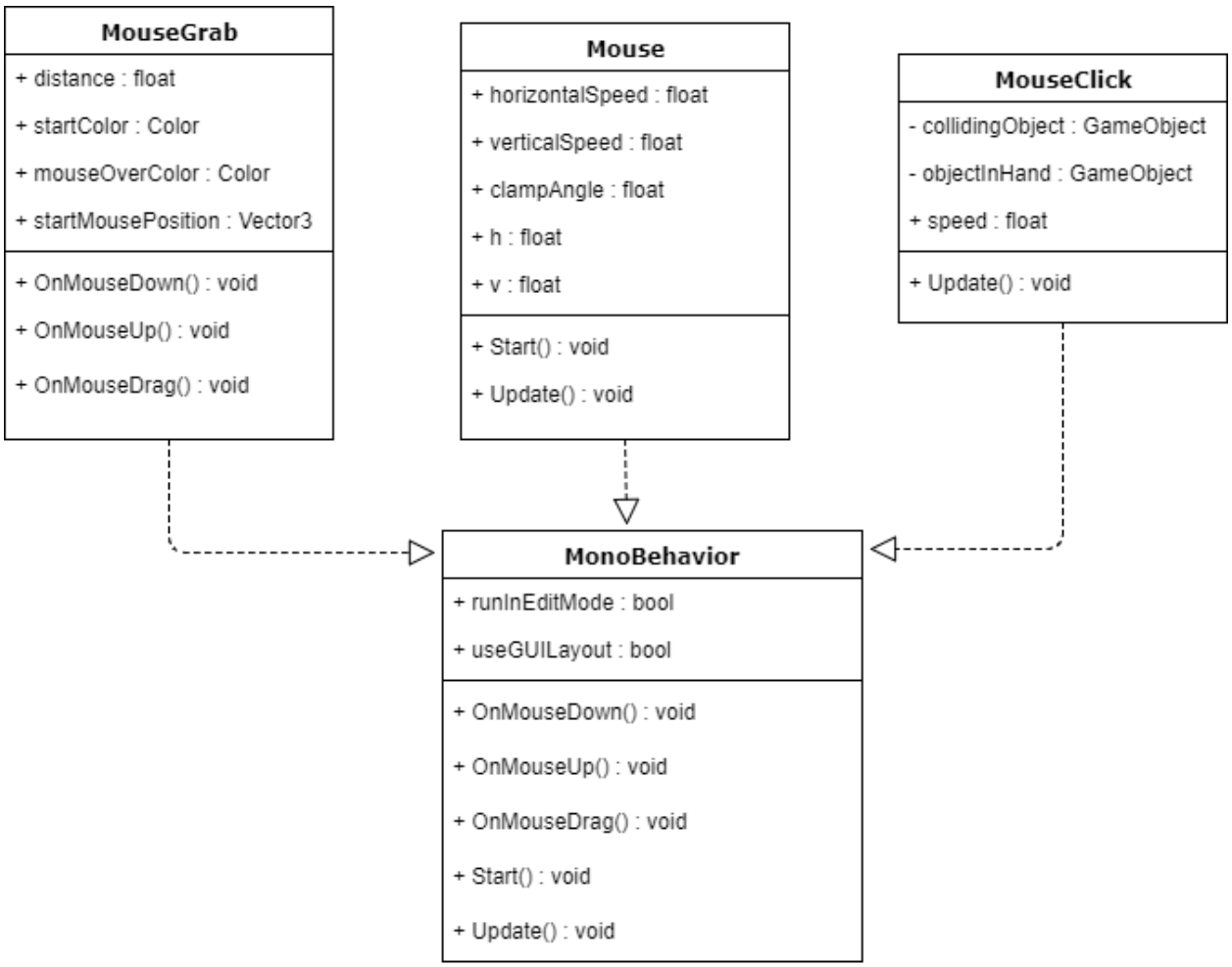


Fig 3 - Use Keyboard and Mouse Controls Class Diagram

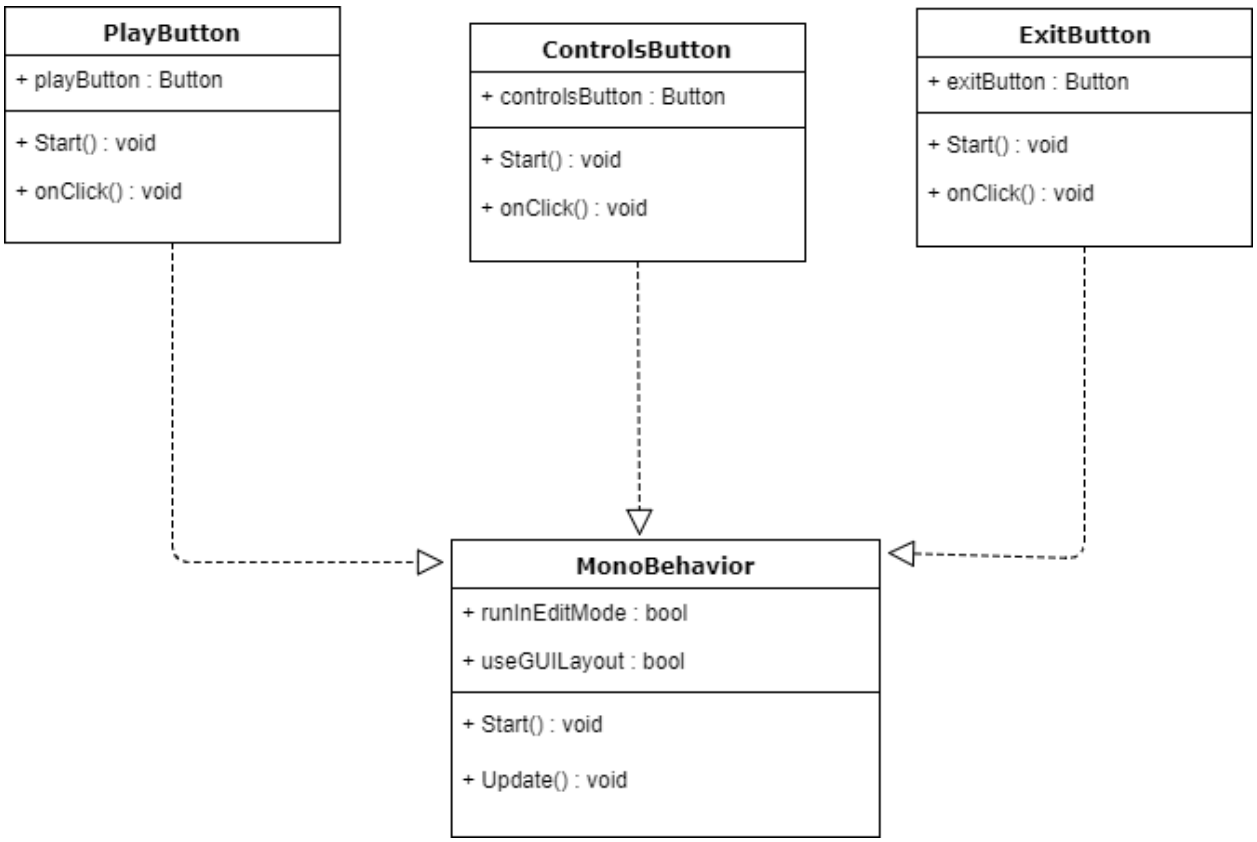


Fig 4 - Add a Main Menu Class Diagram

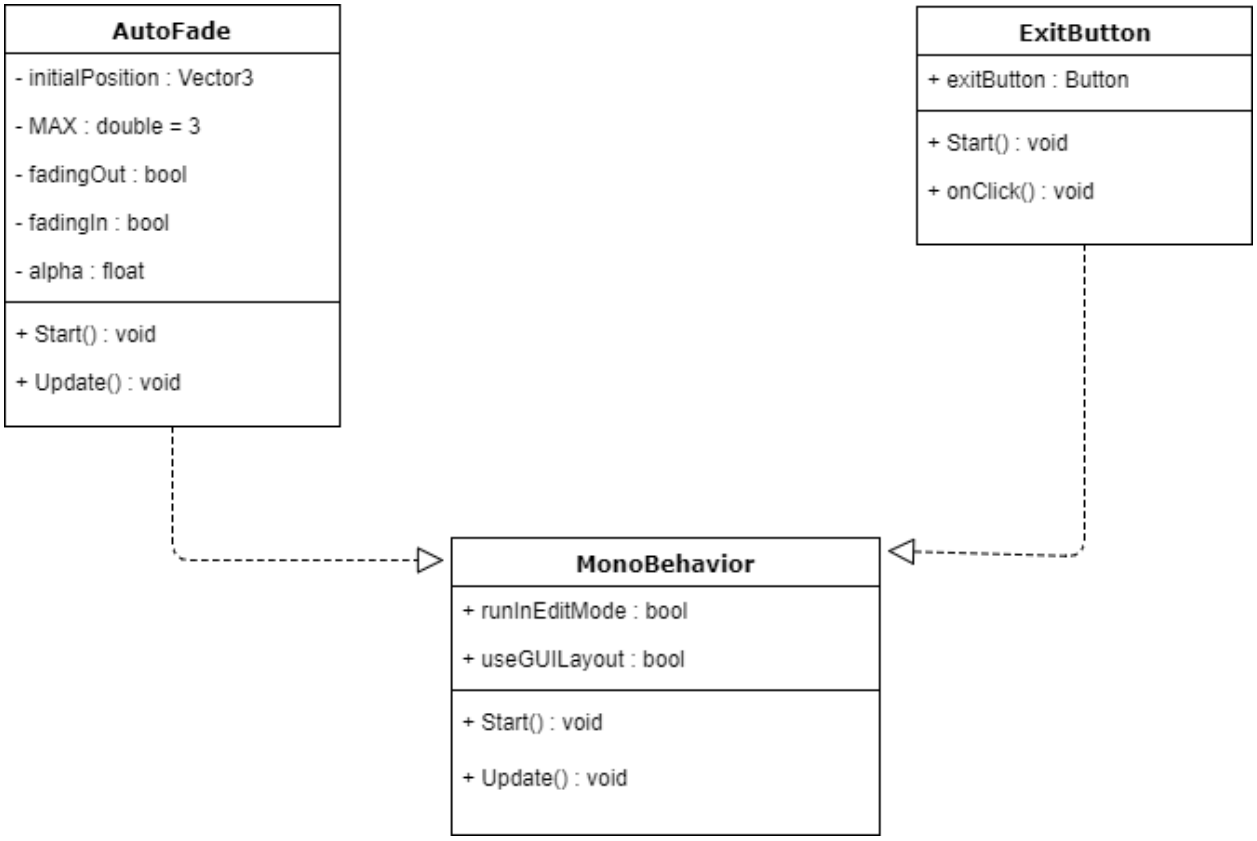


Fig 5 - Add an In-Game Menu Class Diagram

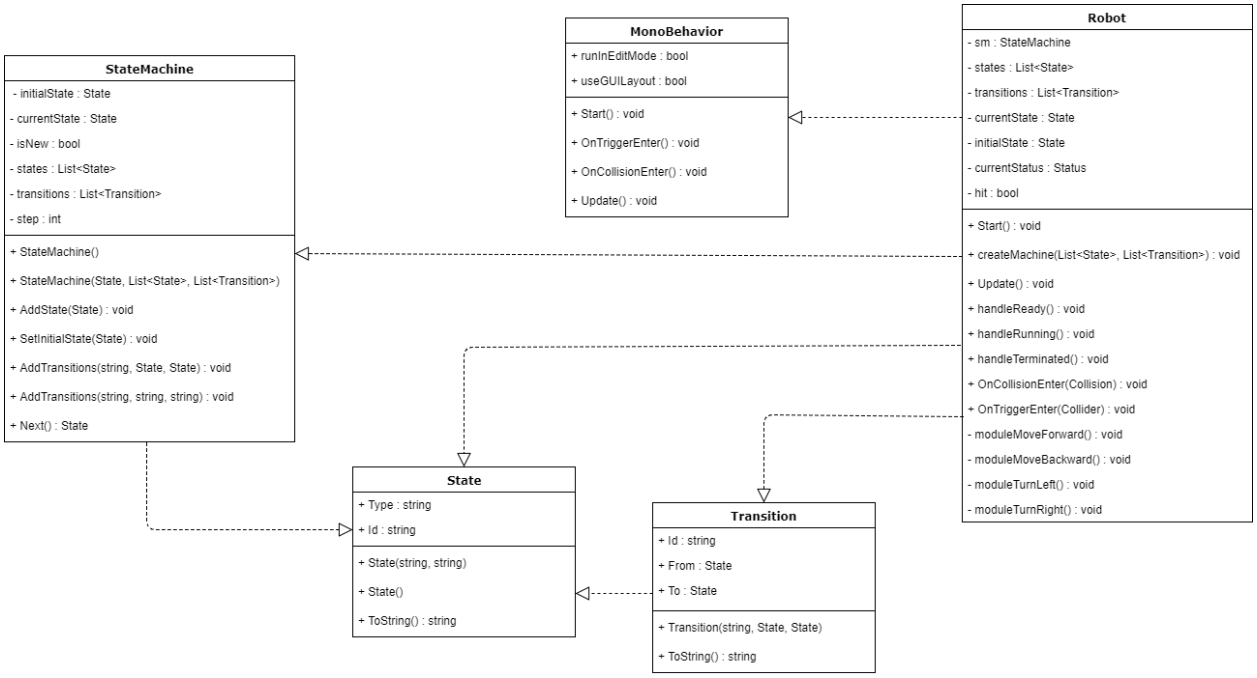


Fig 6 - Add right Turn Module Class Diagram

CircGR Team:

- **Use Case Diagrams**

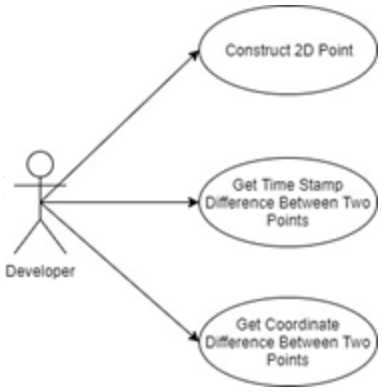


Fig 1 - Implementing Point class

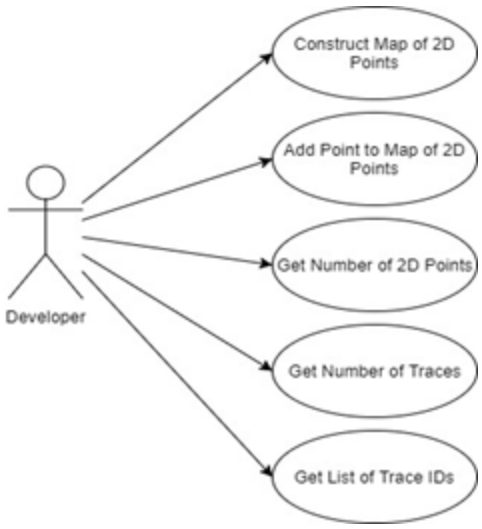


Fig 2 - Implementing PointMap class

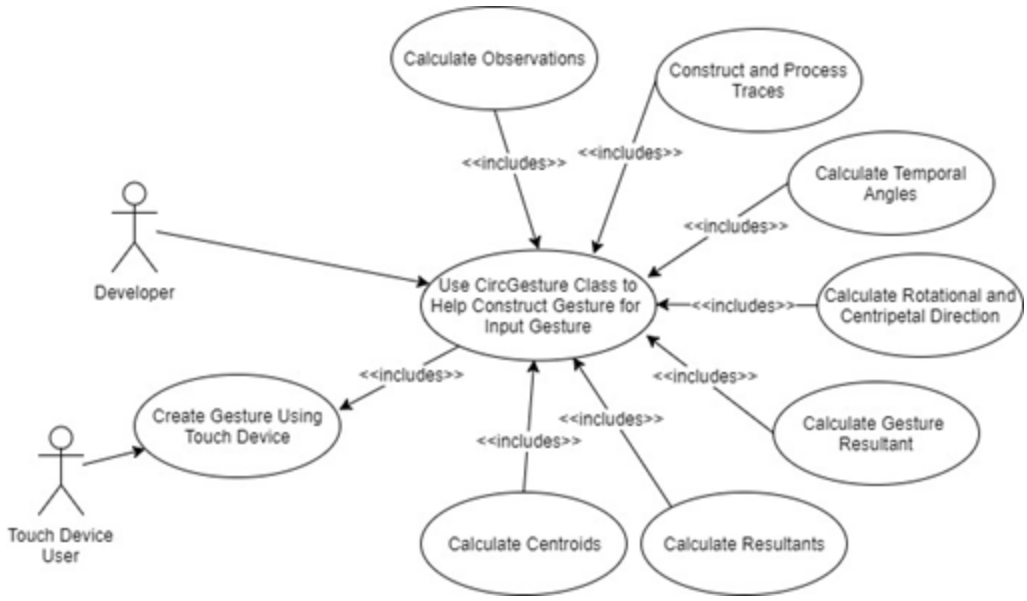


Fig 3 - Implementing CircGesture class

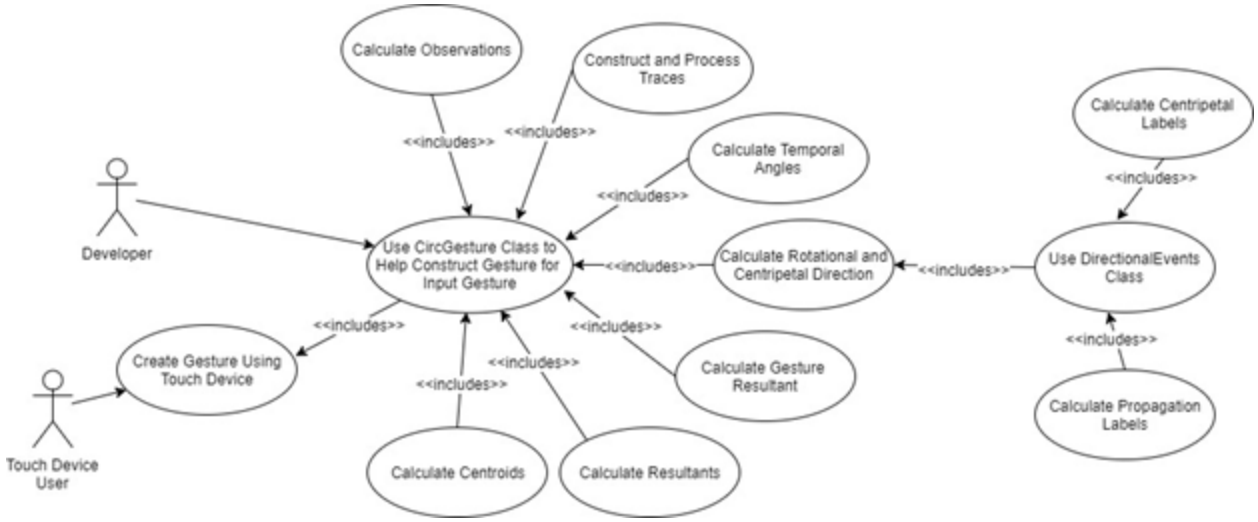


Fig 4 - Implementing DirectionalEvents class

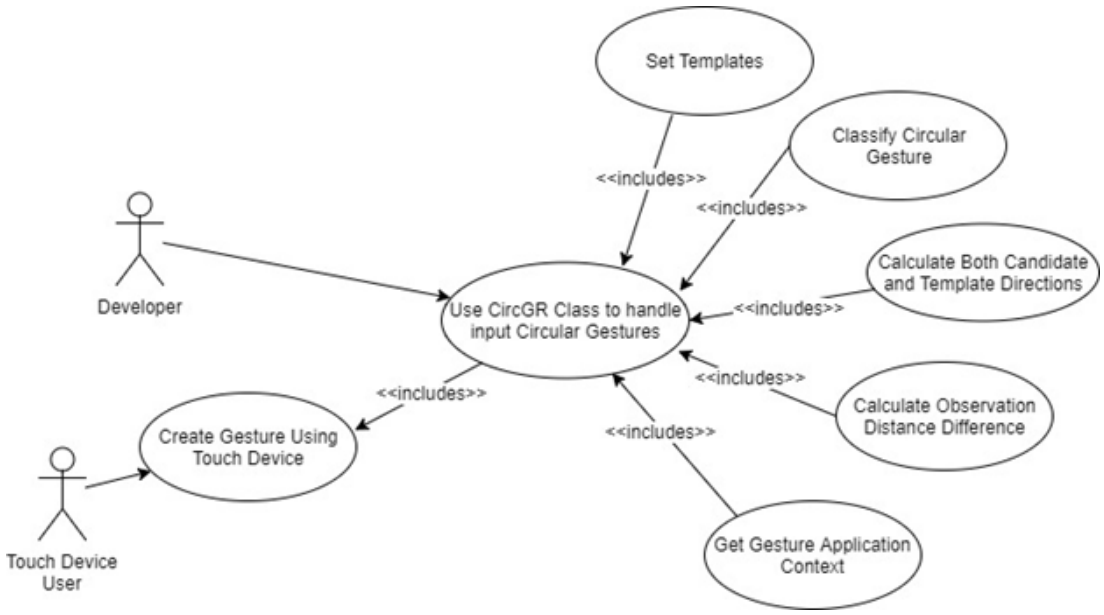


Fig 5 - Implementing CircGR class

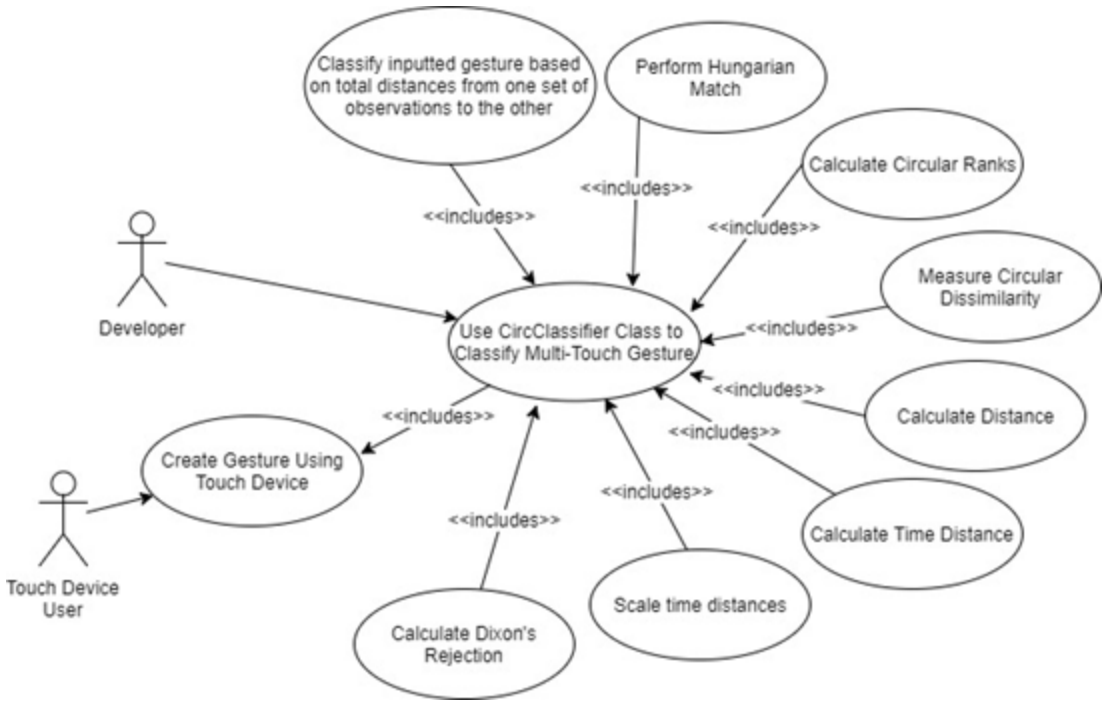


Fig 6 - Implementing CircClassifier class

- **Sequence Diagrams**

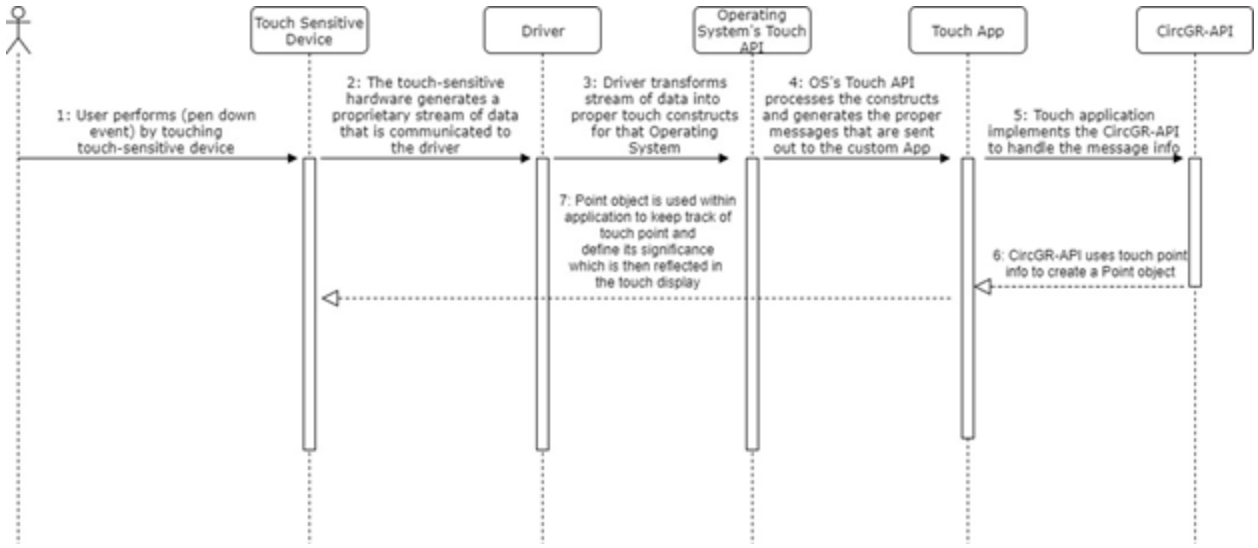


Fig 1 - Point class sequence diagram

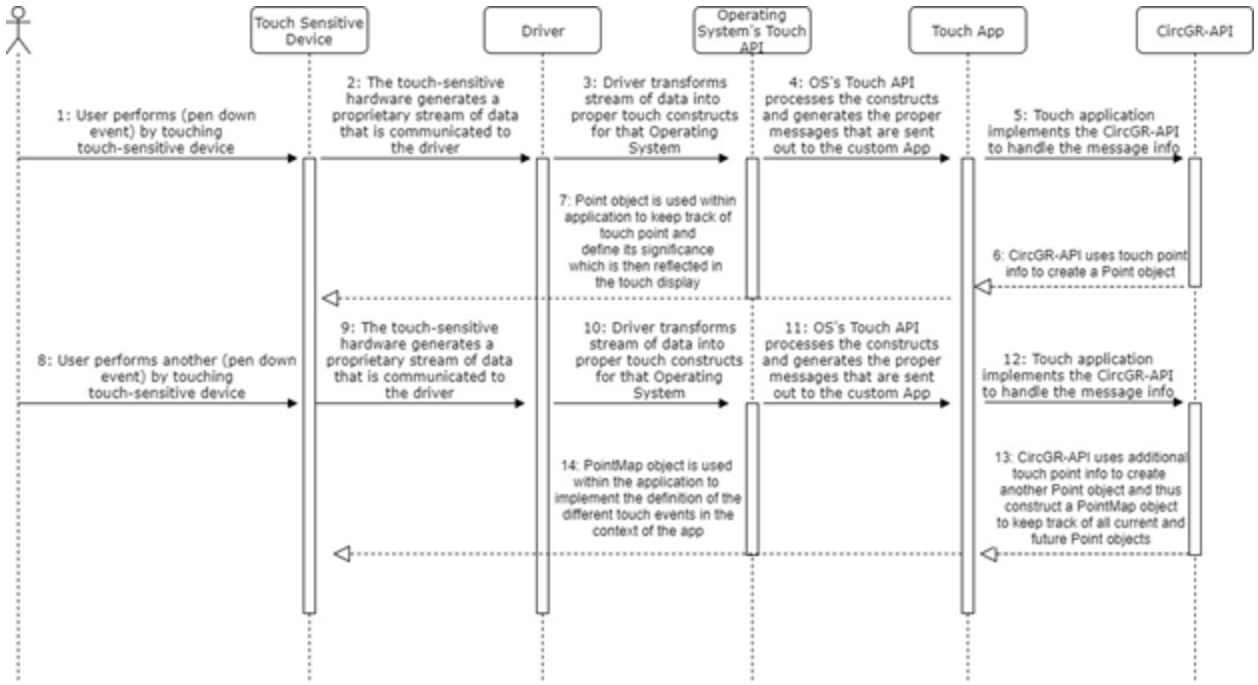


Fig 2 - PointMap sequence diagram

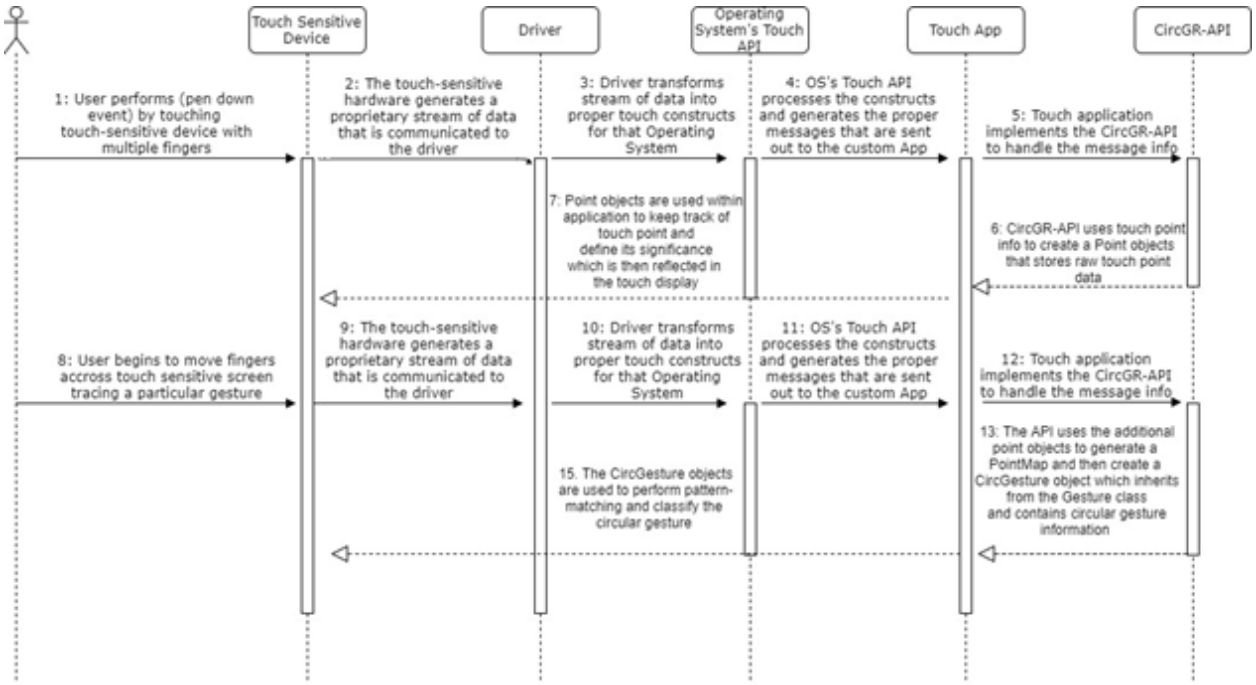


Fig 3 - CircGesture sequence diagram

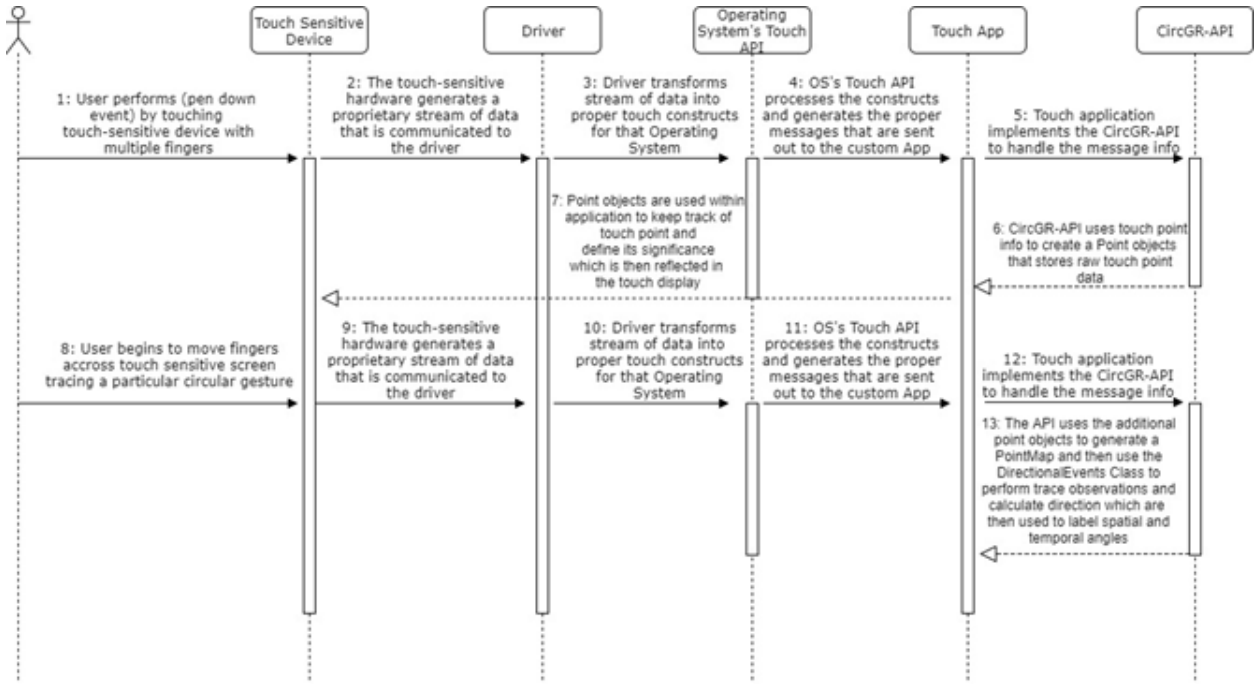


Fig. 4 - DirectionalEvents sequence diagram

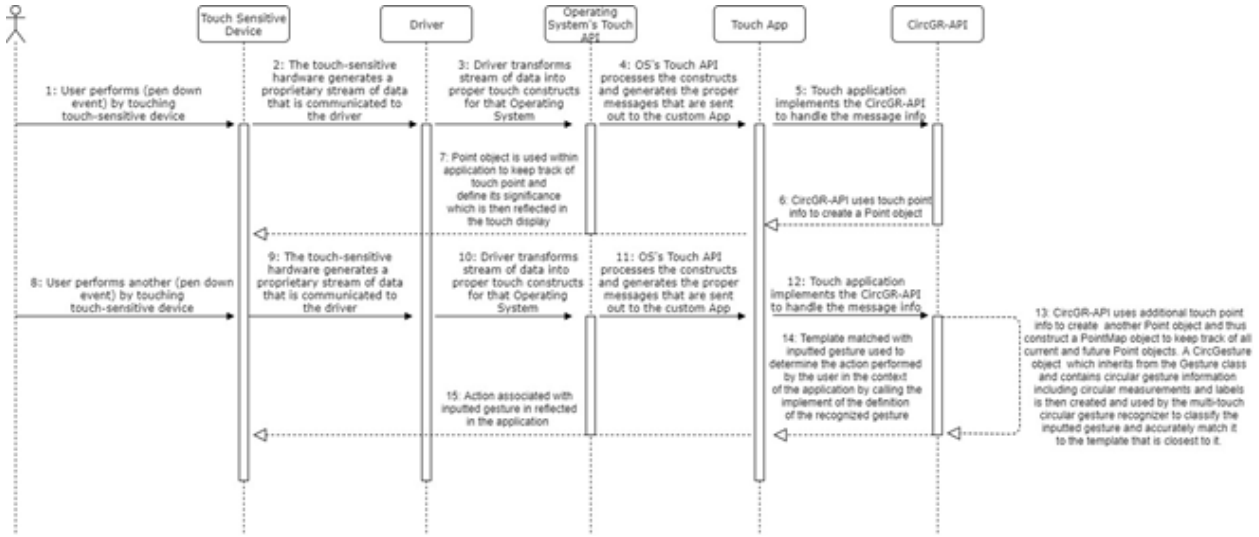


Fig. 5 - CircGR sequence diagram

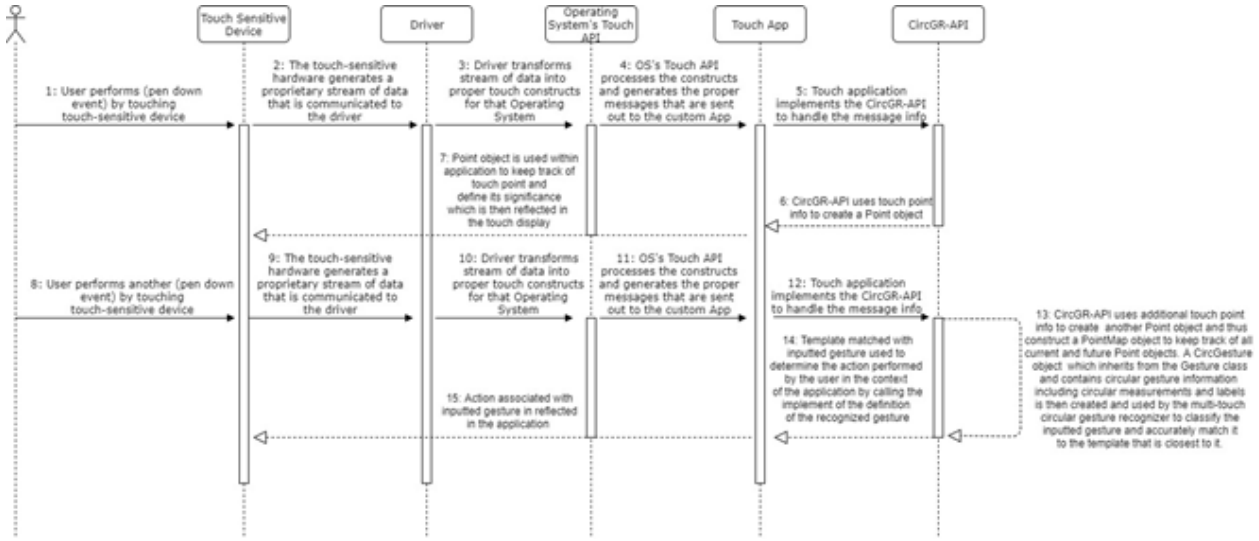


Fig 6 - CircClassifier sequence diagram

- **Class Diagrams**

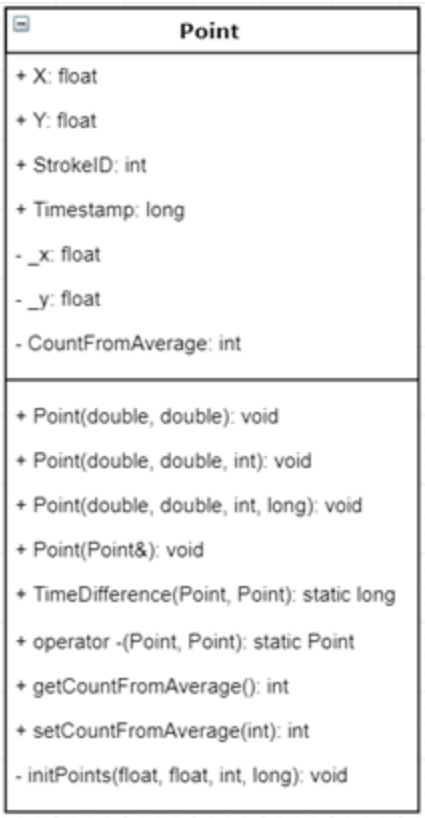


Fig. 1 - Point Class

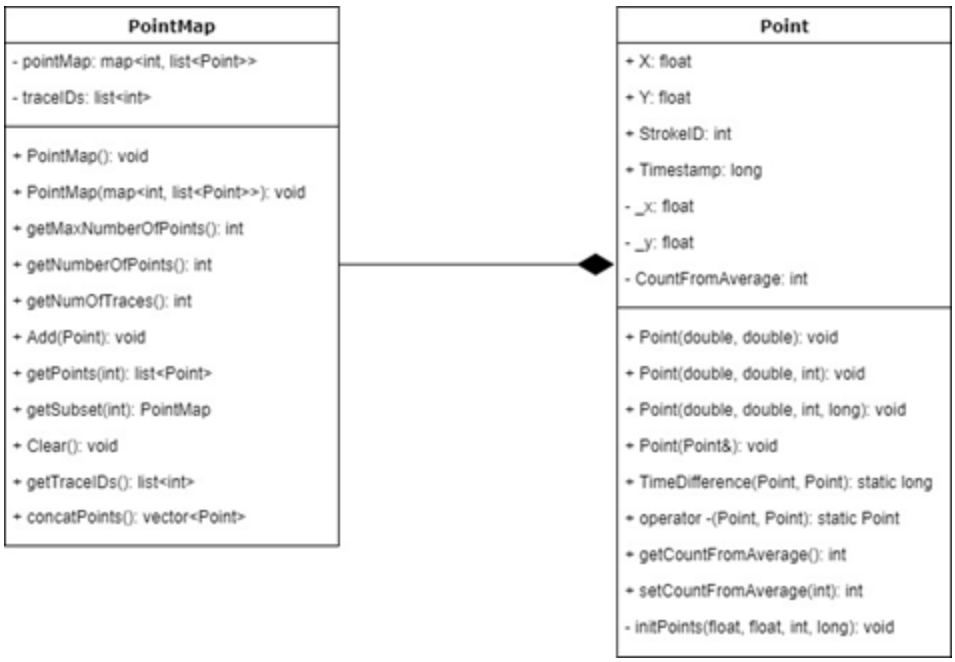


Fig 2. - PointMap Class

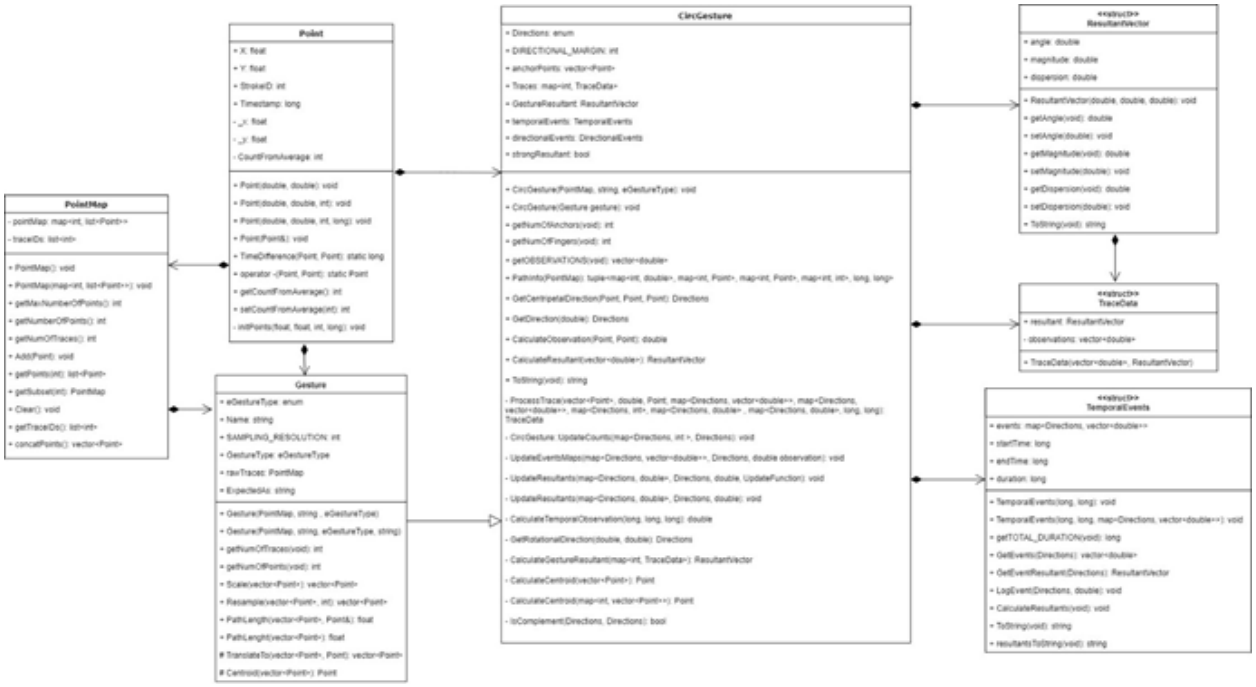


Fig 3. - CircGesture Class

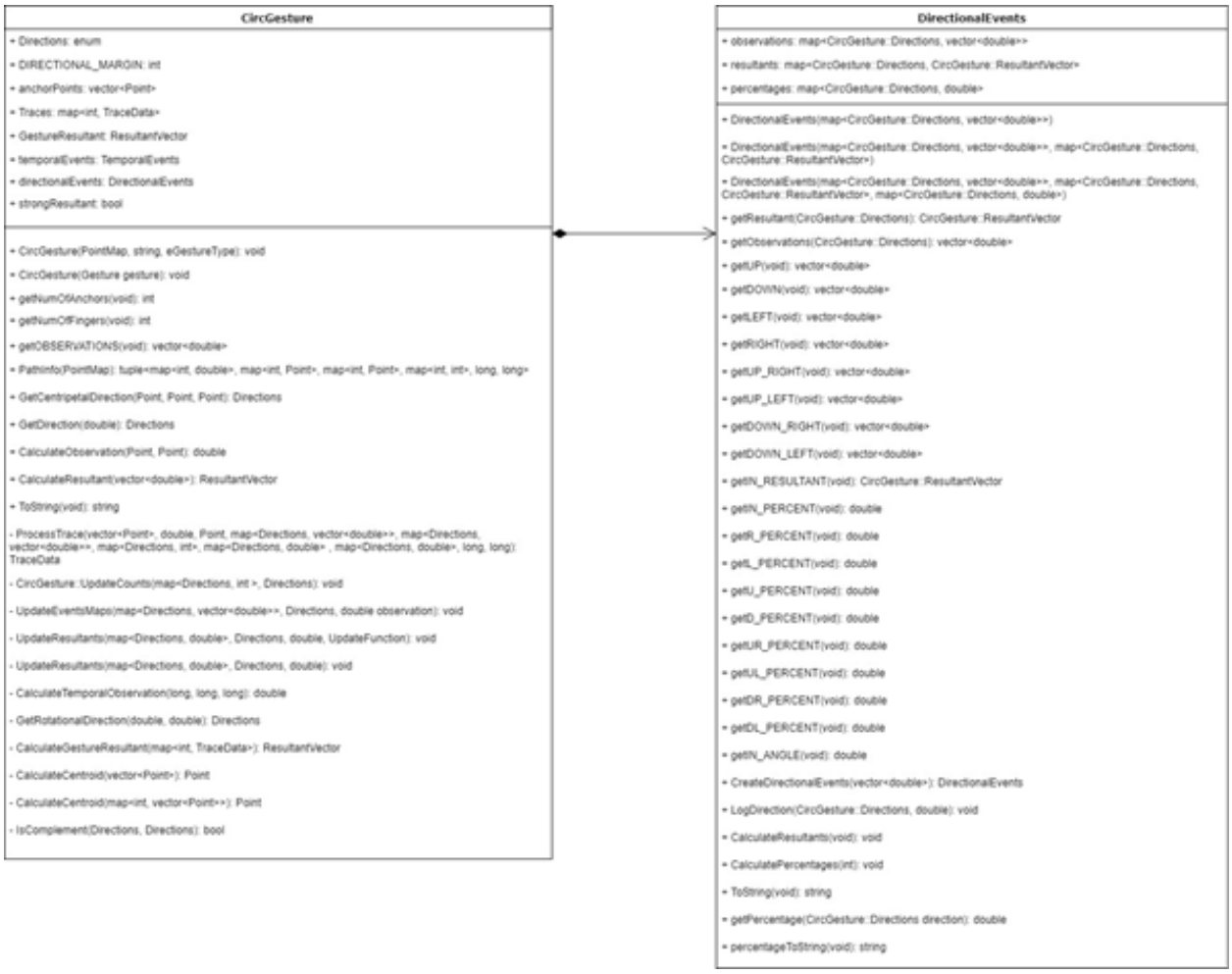


Fig 4. - DirectionalEvents Class

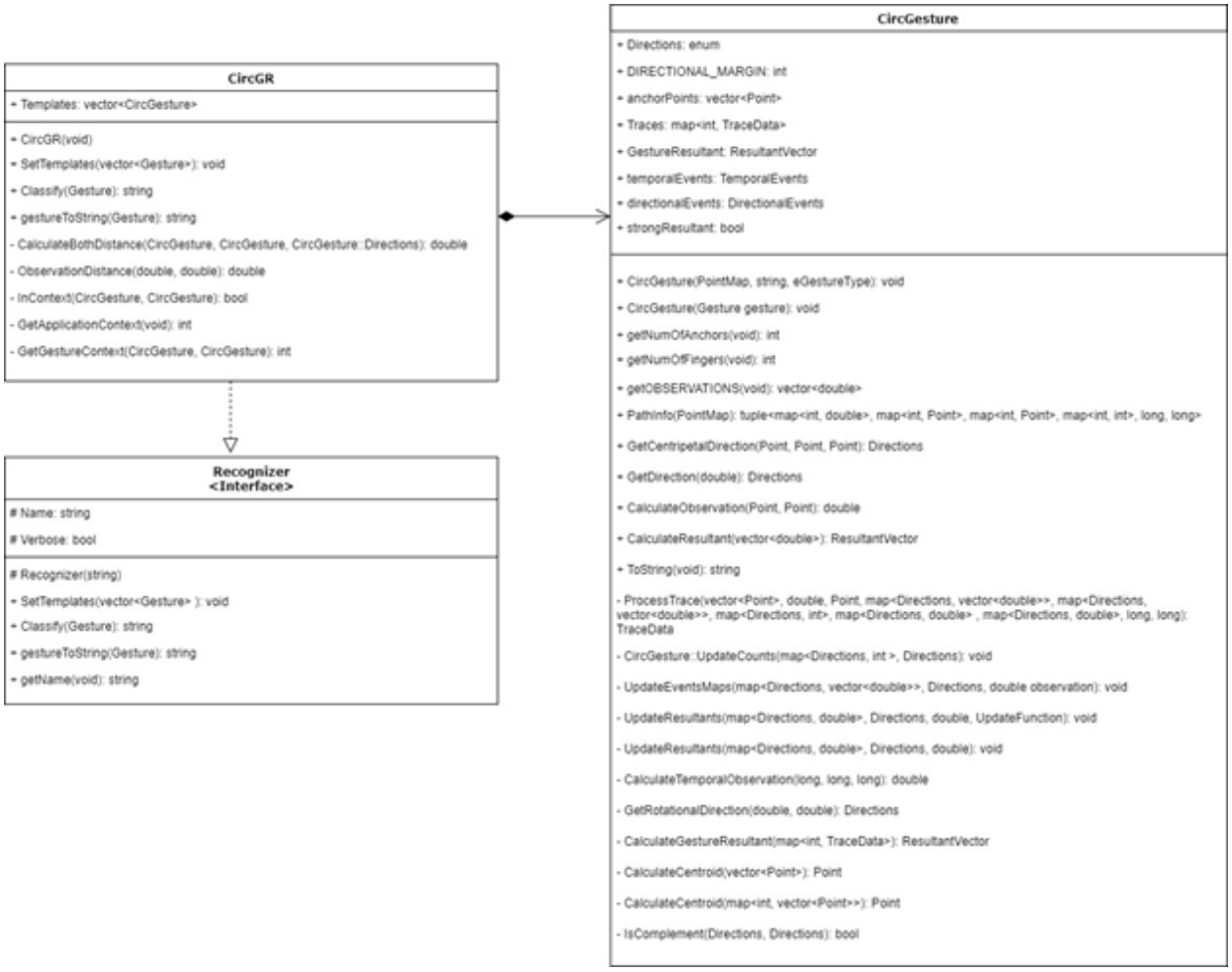


Fig. 5 - CircGR Class Diagram

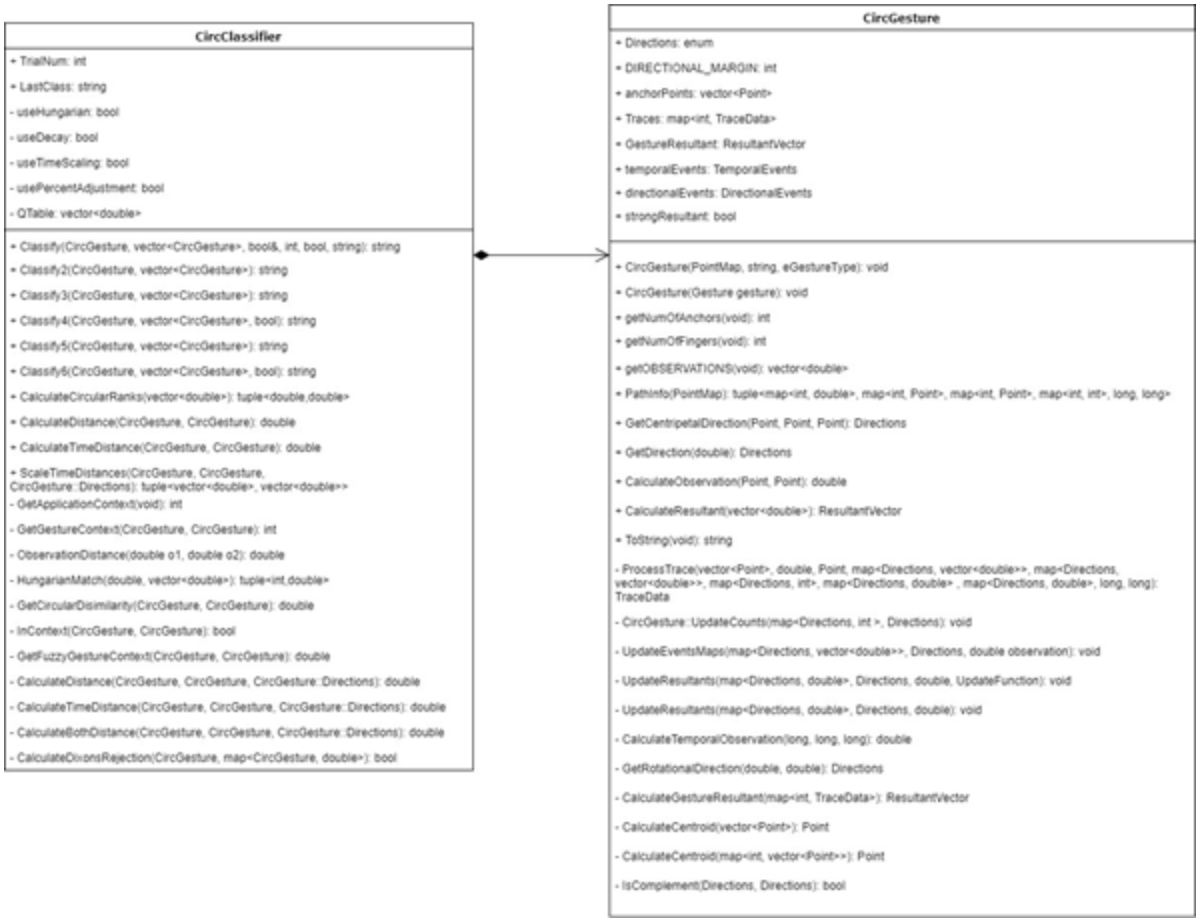
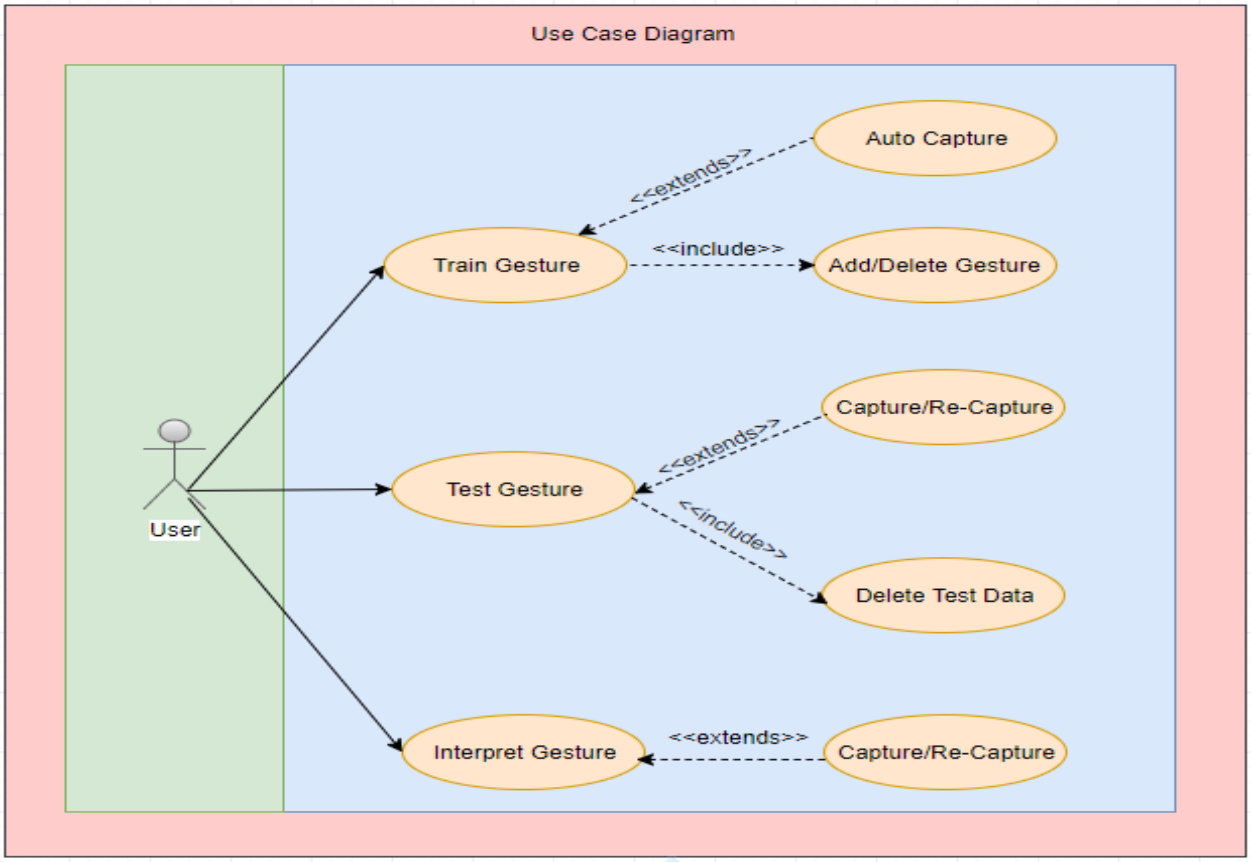


Fig. 6 - CircClassifier Class

LEAP-Trainer Team:

- **Use Case Diagram**



- **Sequence Diagrams**

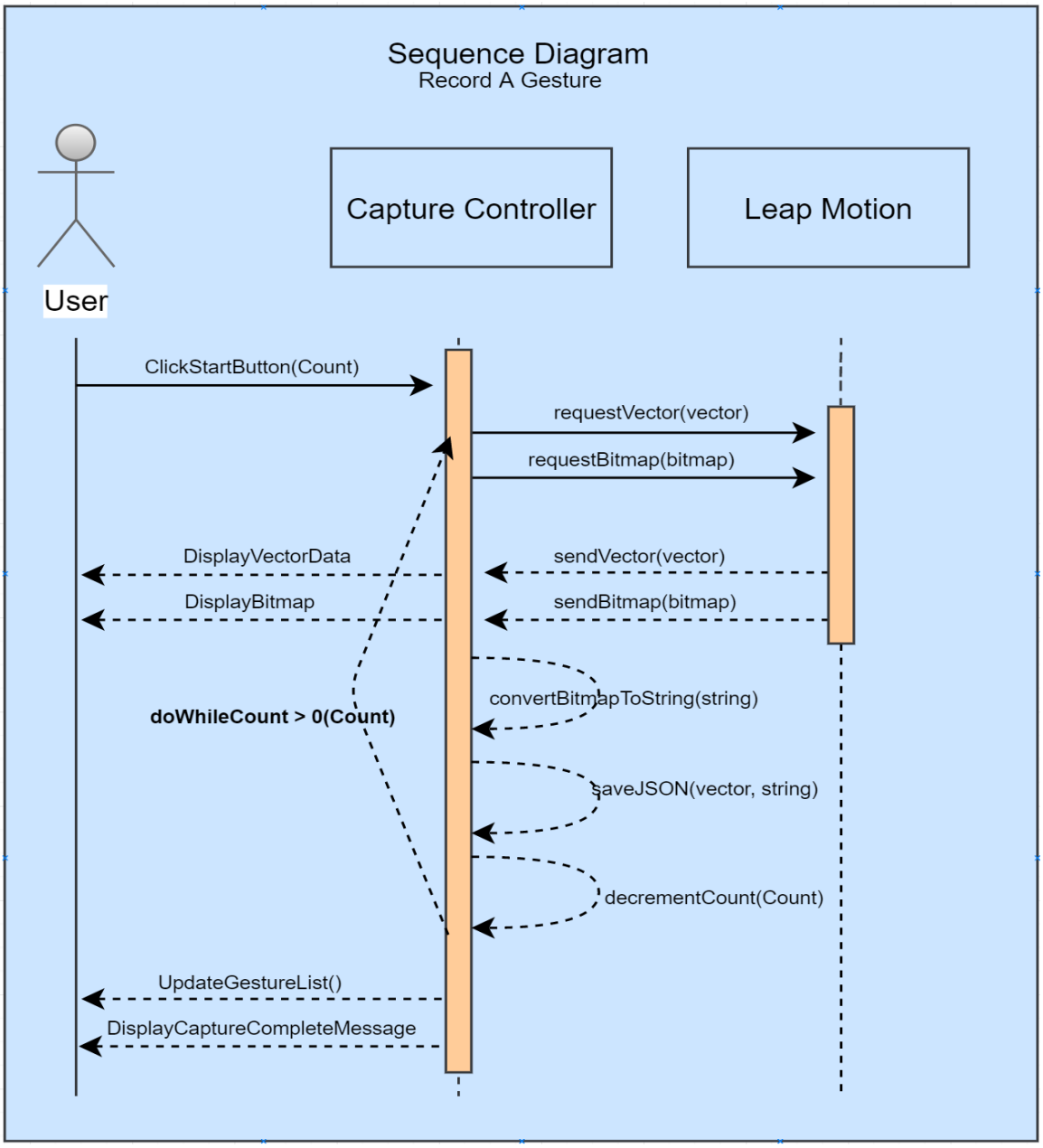

Sequence Diagram
Record A Gesture
User
Capture Controller
Leap Motion
ClickStartButton(Count)
requestVector(vector)
requestBitmap(bitmap)
DisplayVectorData
sendVector(vector)
DisplayBitmap
sendBitmap(bitmap)
convertBitmapToString(string)
doWhileCount > 0(Count)
saveJSON(vector, string)
decrementCount(Count)
UpdateGestureList()
DisplayCaptureCompleteMessage

Fig 1. Record A Gesture

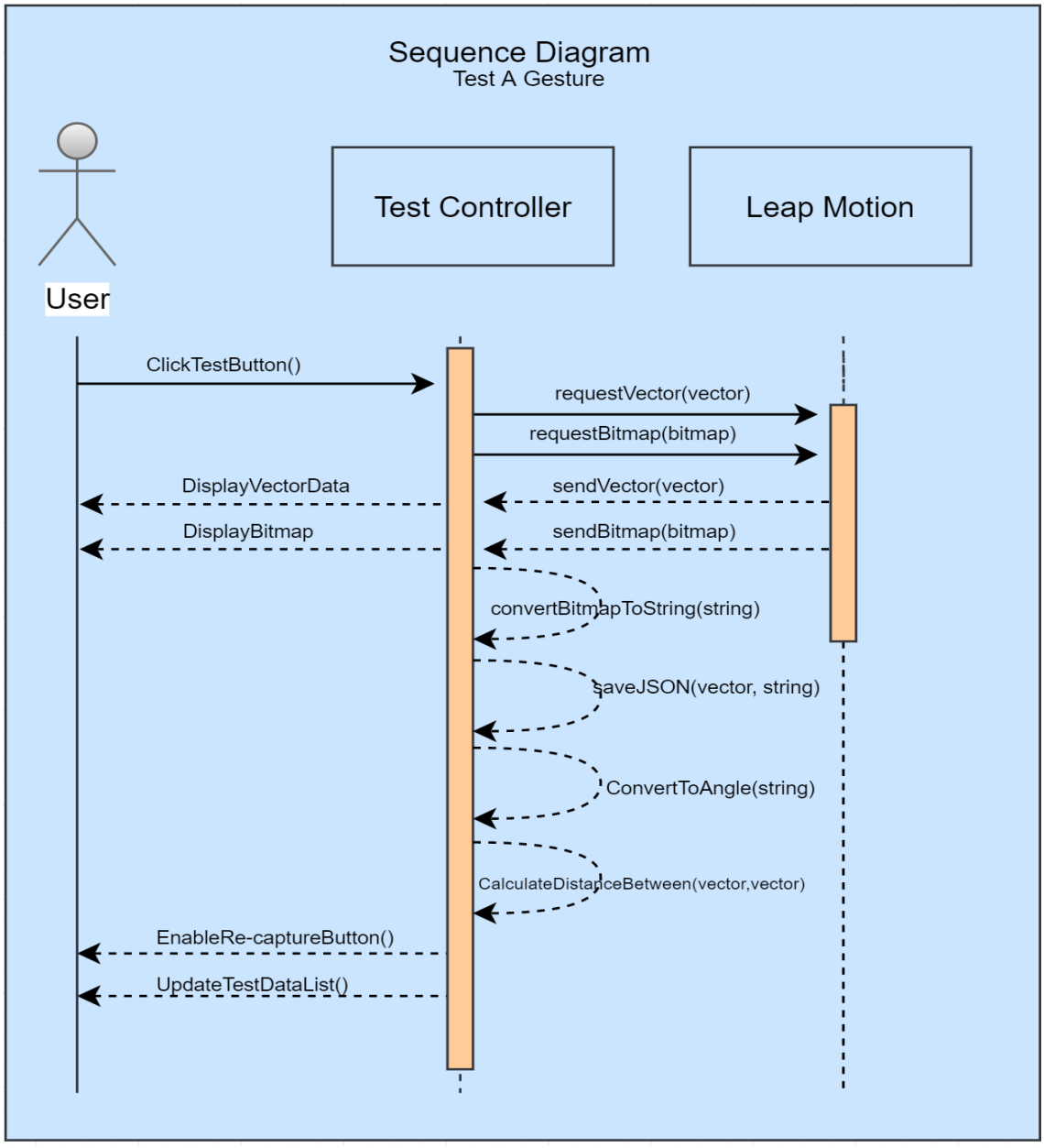

Sequence Diagram
Test A Gesture
User
Test Controller
Leap Motion
ClickTestButton()
requestVector(vector)
requestBitmap(bitmap)
DisplayVectorData
sendVector(vector)
DisplayBitmap
sendBitmap(bitmap)
convertBitmapToString(string)
saveJSON(vector, string)
ConvertToAngle(string)
CalculateDistanceBetween(vector,vector)
EnableRe-captureButton()
UpdateTestDataList()

Fig 2. Test A Gesture

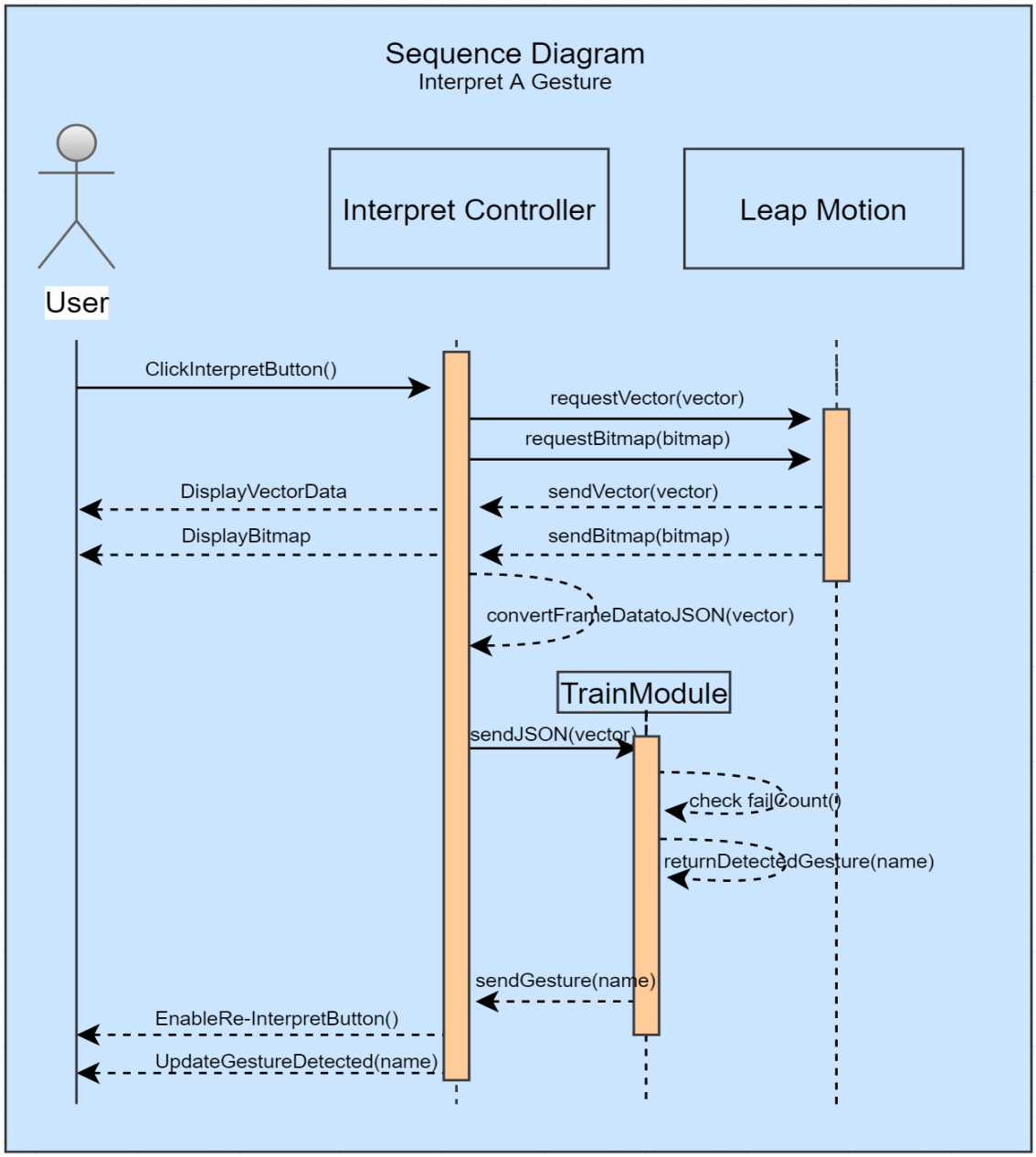

Sequence Diagram
Interpret A Gesture
User
Interpret Controller
Leap Motion
ClickInterpretButton()
requestVector(vector)
requestBitmap(bitmap)
DisplayVectorData
sendVector(vector)
DisplayBitmap
sendBitmap(bitmap)
convertFrameDatatoJSON(vector)
TrainModule
sendJSON(vector)
check fail count()
returnDetectedGesture(name)
sendGesture(name)
EnableRe-InterpretButton()
UpdateGestureDetected(name)

Fig 3. Interpret A Gesture

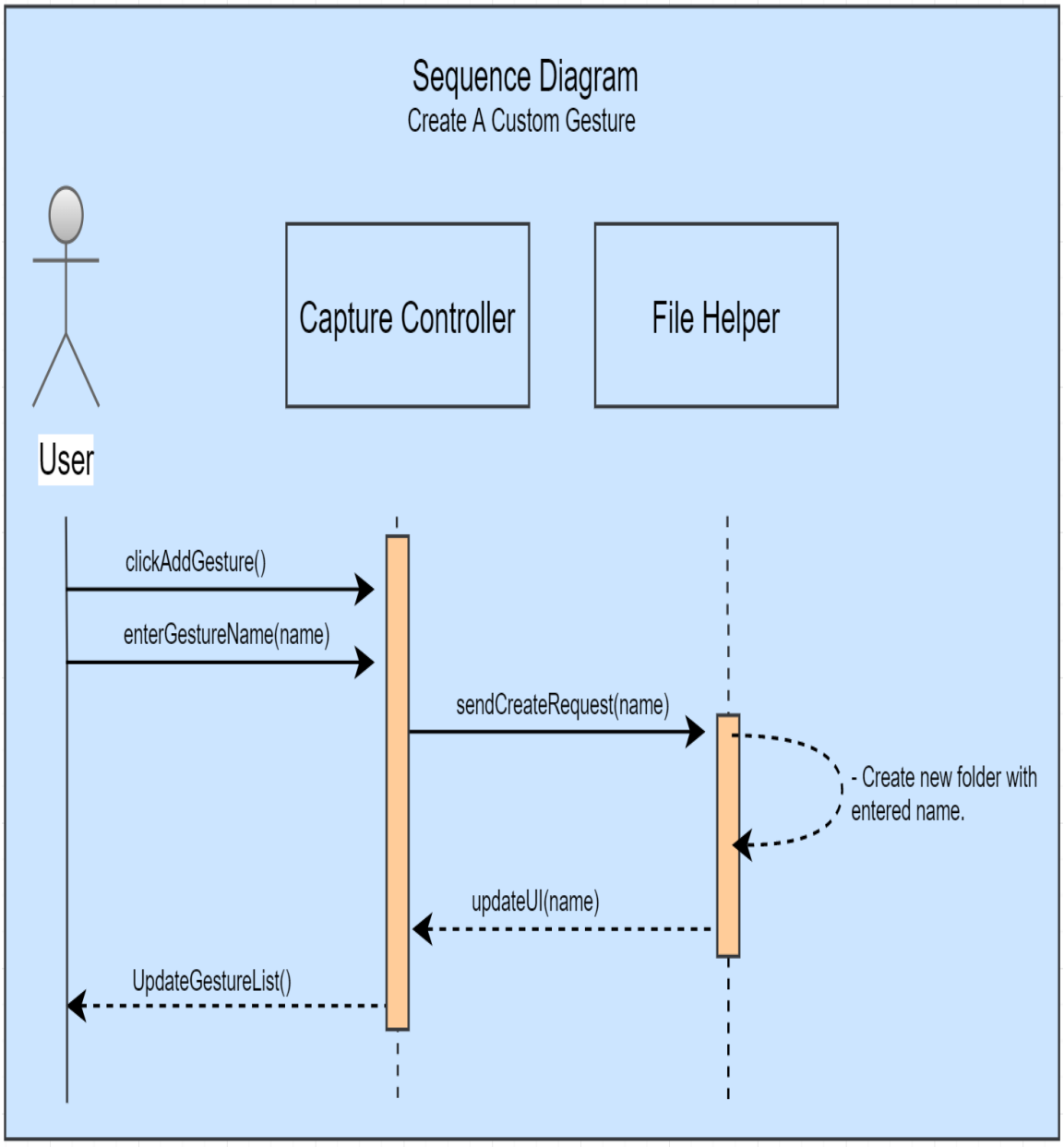

Sequence Diagram
Create A Custom Gesture
Capture Controller
File Helper
User
clickAddGesture()
enterGestureName(name)
sendCreateRequest(name)
- Create new folder with entered name.
updateUI(name)
UpdateGestureList()

Fig 4. Create A Custom Gesture

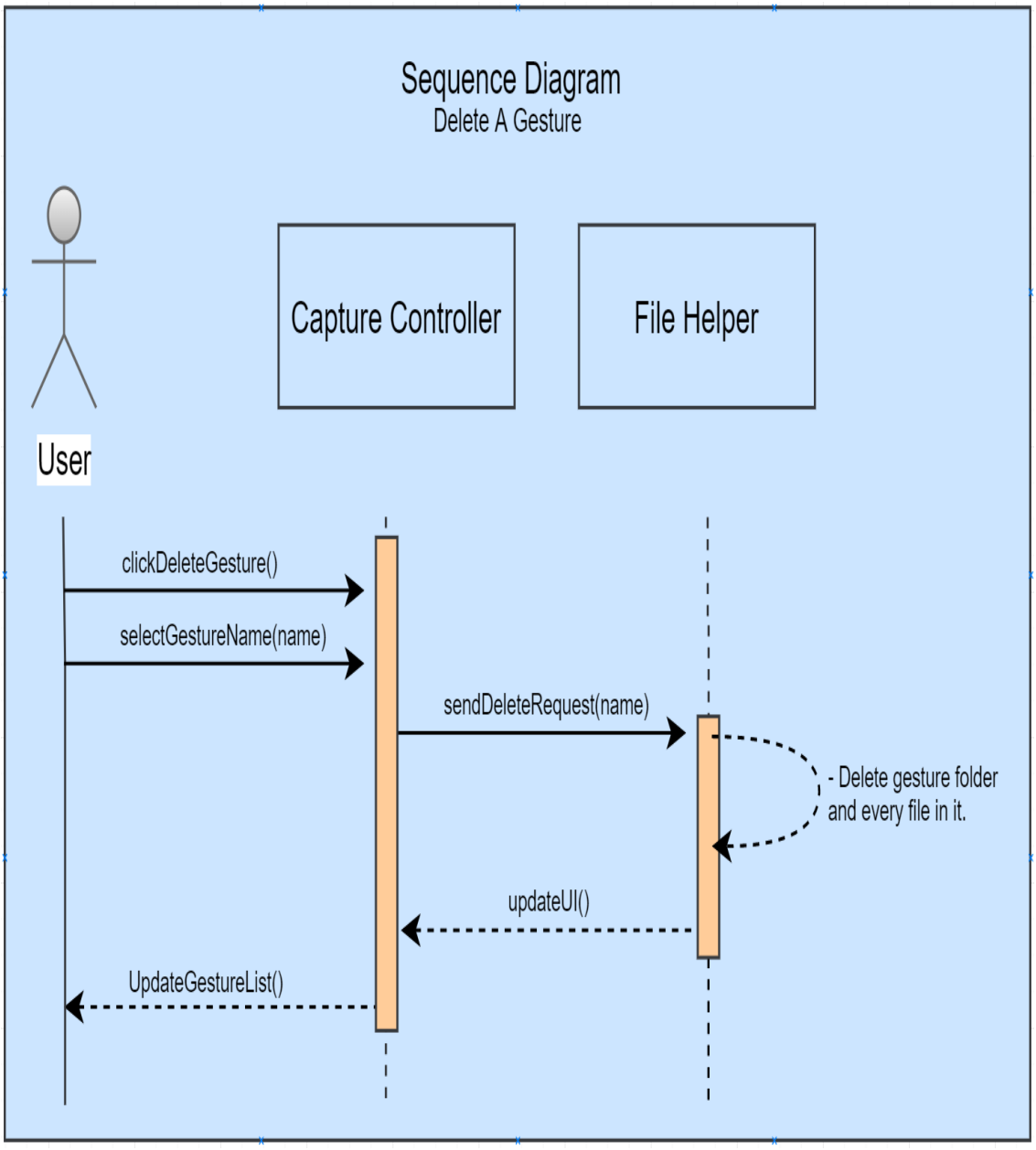

Sequence Diagram
Delete A Gesture
Capture Controller
File Helper
User
clickDeleteGesture()
selectGestureName(name)
sendDeleteRequest(name)
- Delete gesture folder and every file in it.
updateUI()
UpdateGestureList()

Fig. 5 Delete A Gesture

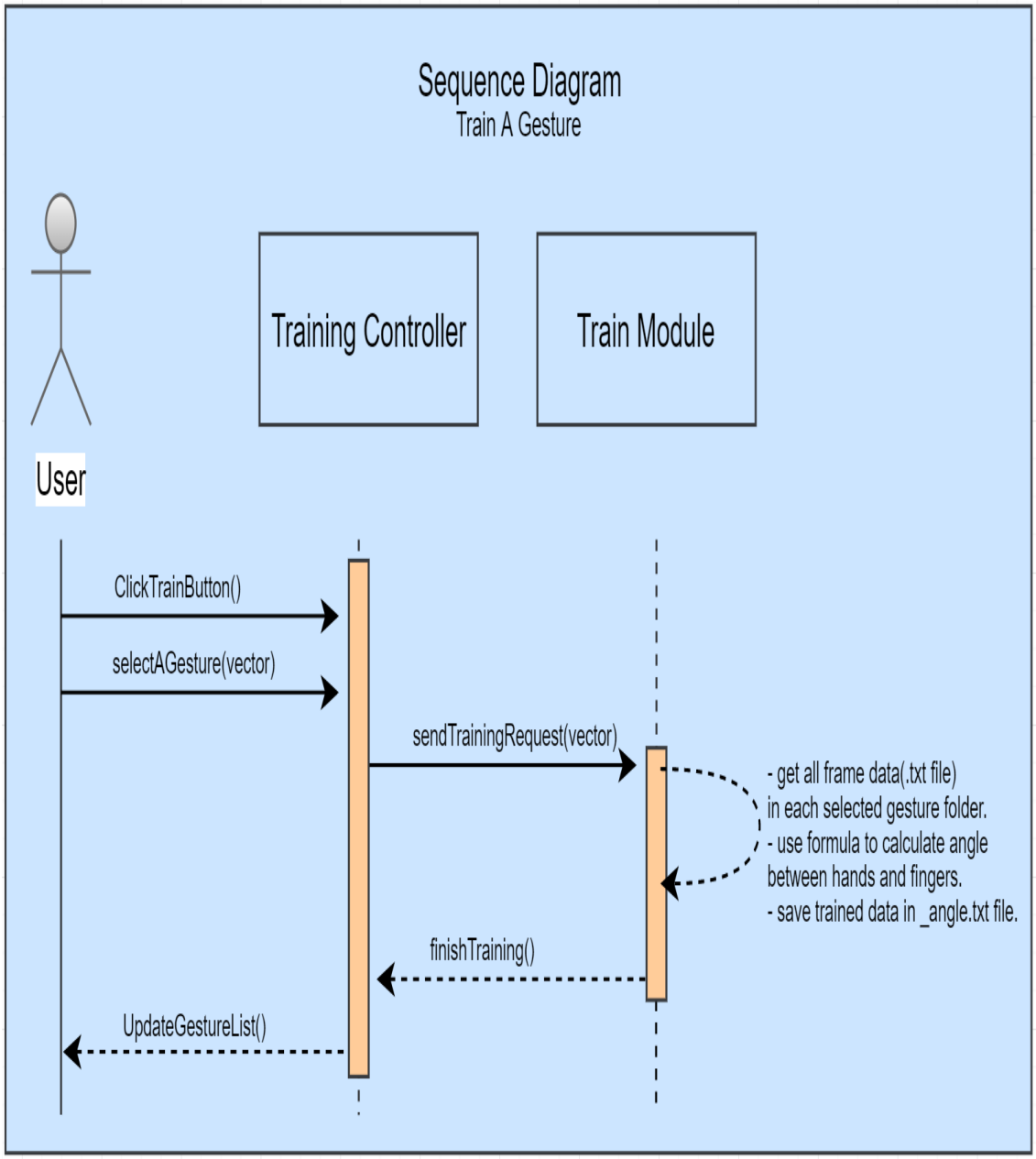

Sequence Diagram
Train A Gesture
Training Controller
Train Module
User
ClickTrainButton()
selectAGesture(vector)
sendTrainingRequest(vector)
- get all frame data(.txt file)
in each selected gesture folder.
- use formula to calculate angle
between hands and fingers.
- save trained data in _angle.txt file.
finishTraining()
UpdateGestureList()

Fig 6. Train A Gesture

- **Class Diagram**

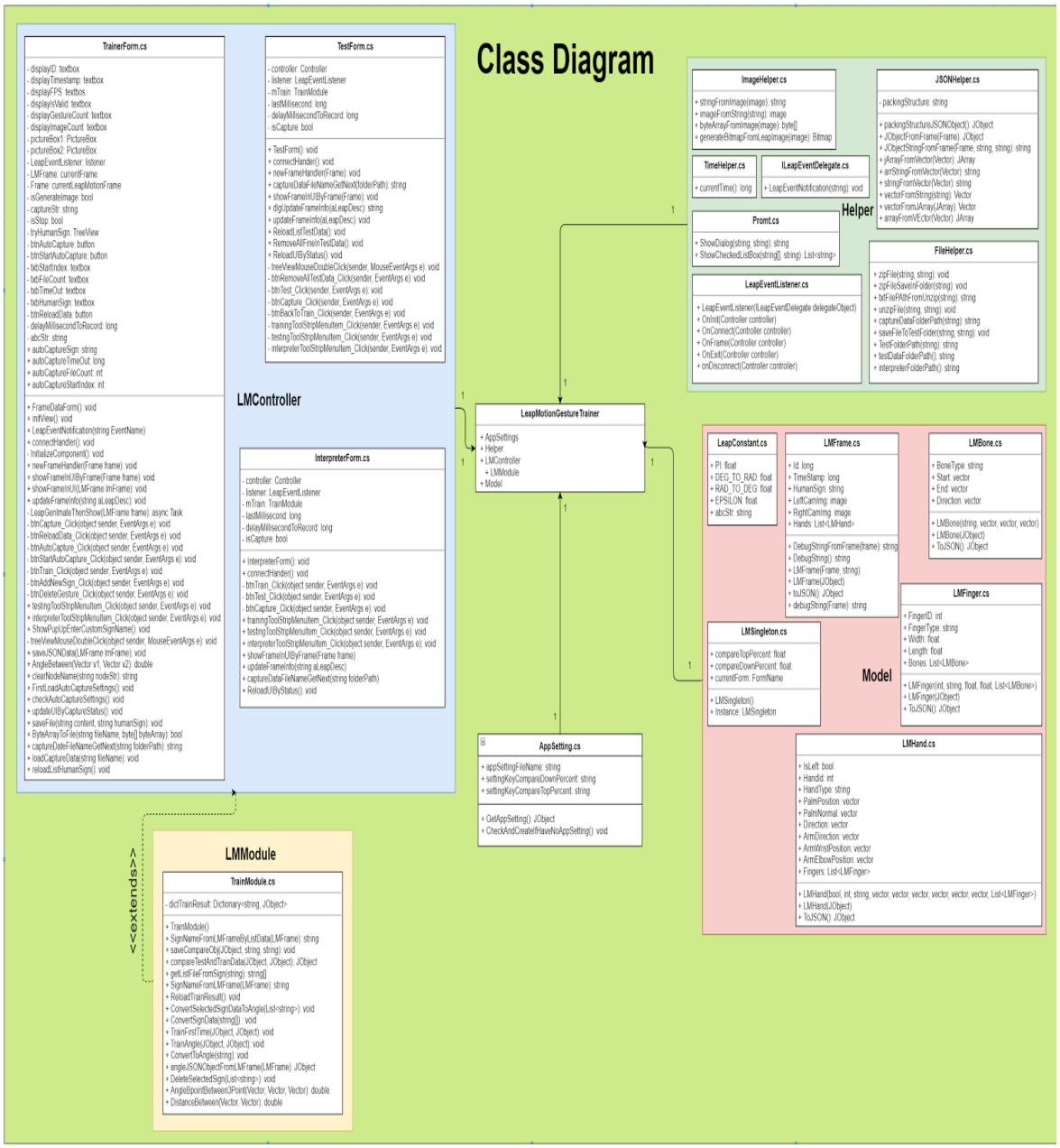

Class Diagram
TrainerForm.cs
TestForm.cs
LMController
InterpreterForm.cs
LeapMotionGestureTrainer
AppSetting.cs
ImageHelper.cs
JSONHelper.cs
TimeHelper.cs
ILeapEventDelegate.cs
Helper
Promt.cs
FileHelper.cs
LeapEventListener.cs
LeapConstant.cs
LMFrame.cs
LMBone.cs
LMFinger.cs
LMSingleton.cs
Model
LMHand.cs
<<extends>>
LMModule
TrainModule.cs

Fig 1. LEAP-Trainer Class Diagram

Appendix B - User Interface Design

1

2

CodeAdventures Team:

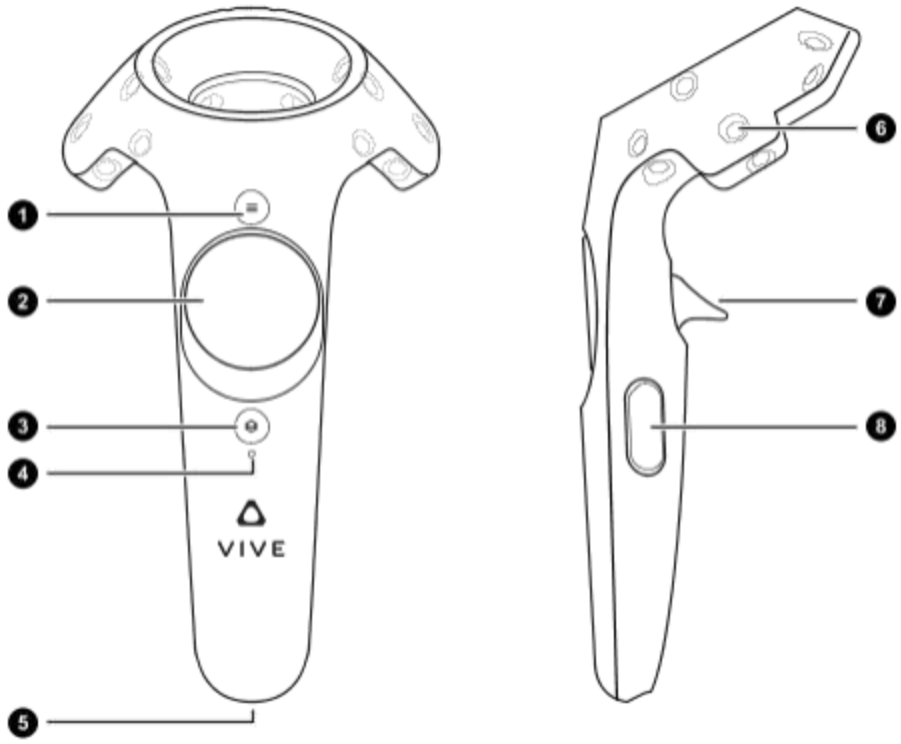


Fig. 1 HTC Vive controller scheme

1. Toggle Trackpad Menu

2. Touchpad (Options will be displayed when Button1 is pressed):
    - Up: Run State Machine
    - Down: Toggle Board
    - Left: Restart Level
    - Right: Quit Application
3. HTC Vive Pause Menu
4. Battery Light Indicator
5. Charging Port
6. IR Tracking Receiver
7. Teleport Movement Controls
8. Grip Button (used to grab objects)

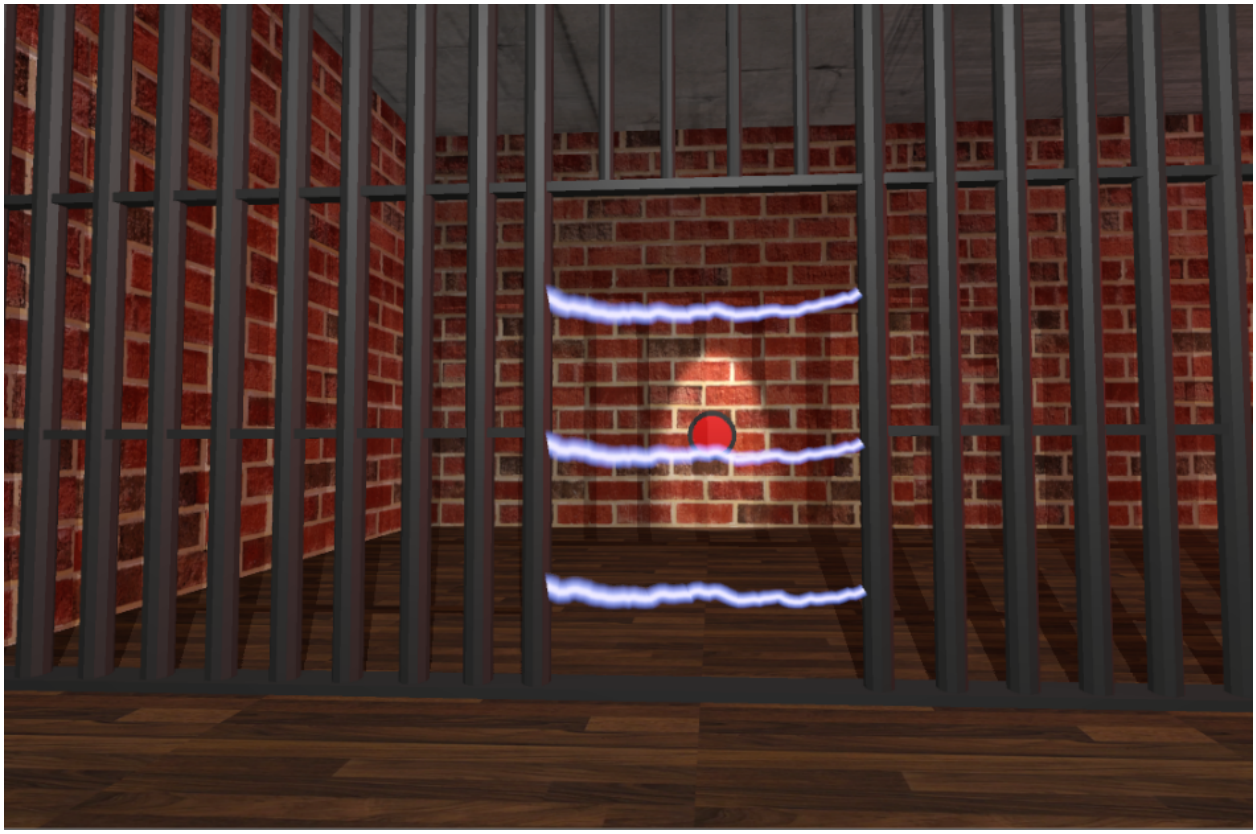


Fig. 2 The main objective

Fig. 3 Board with a forward and left module linked

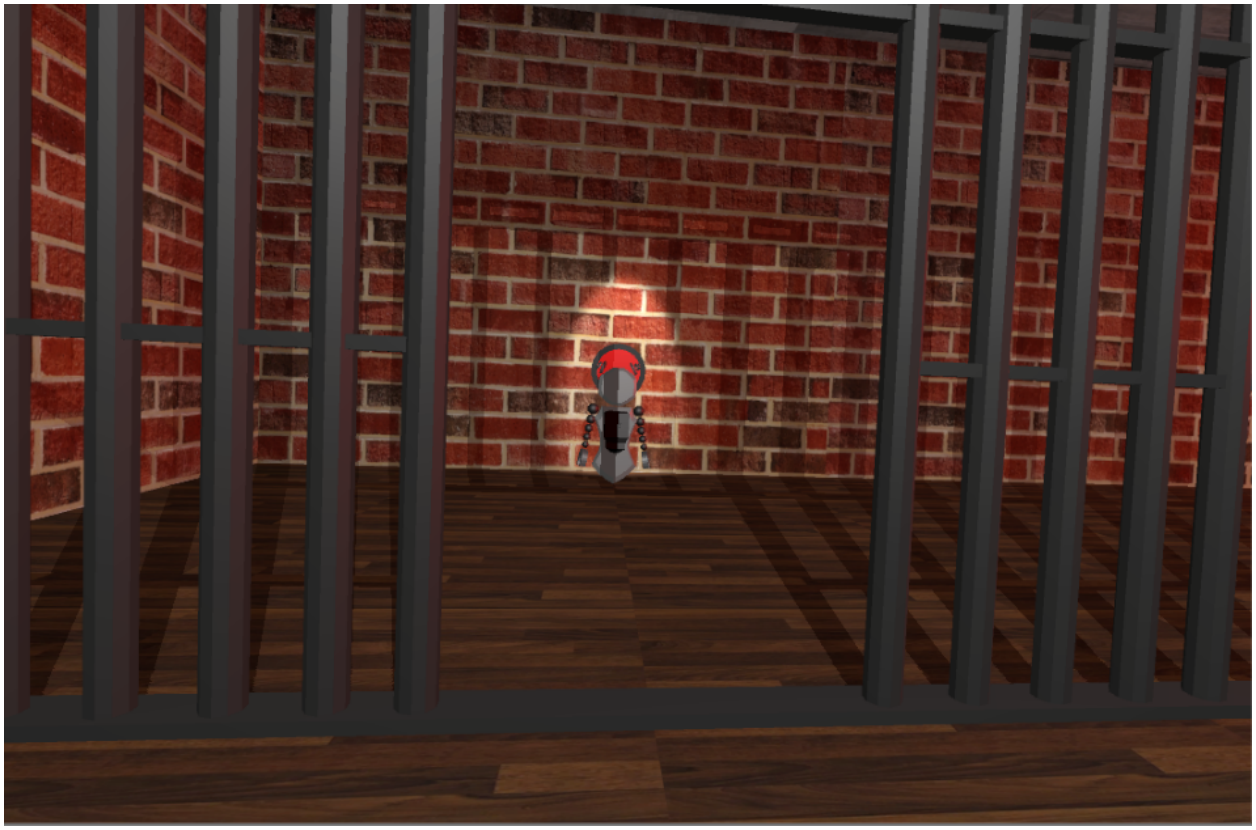


Fig. 3 Objective completed

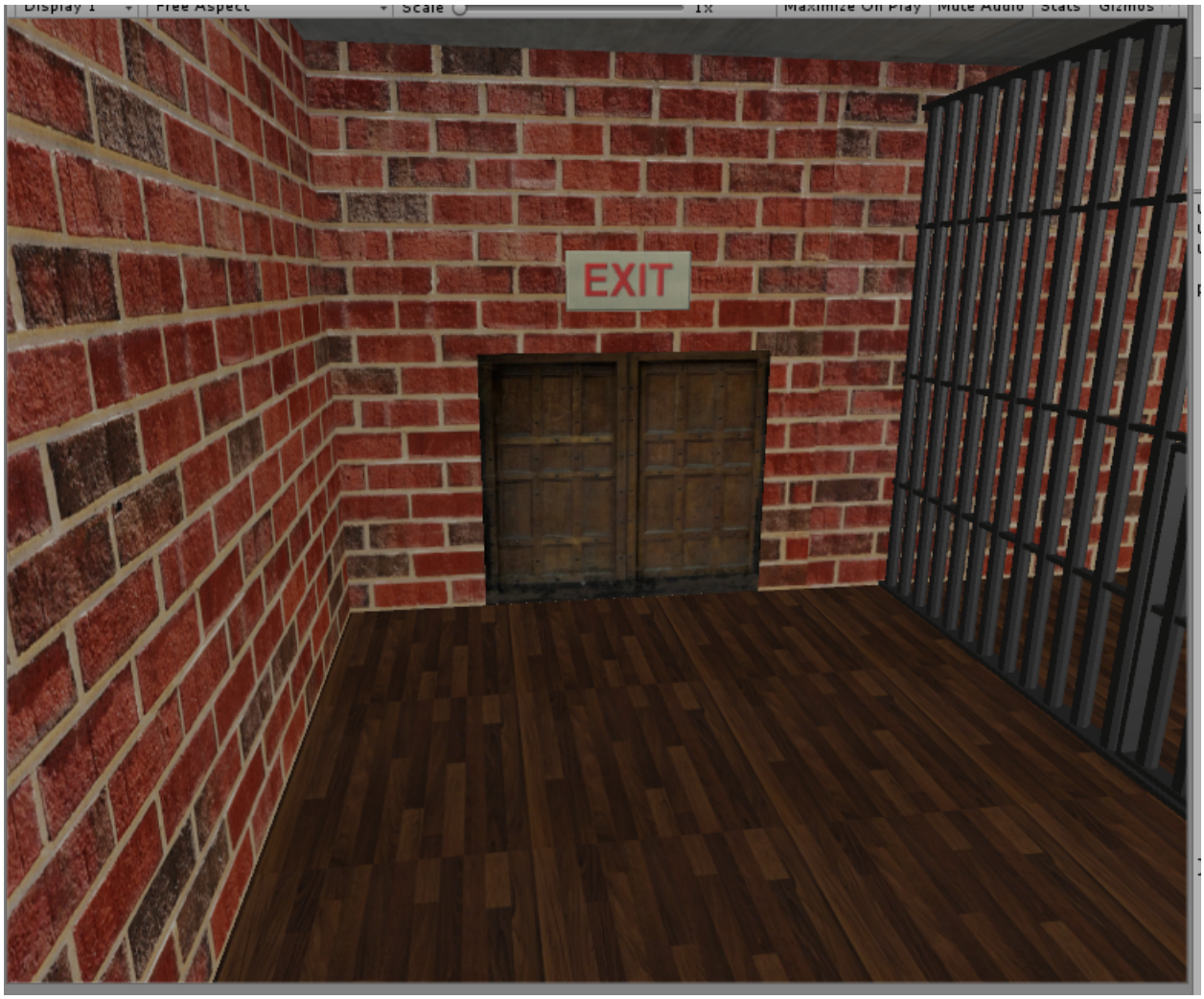


Fig. 4 Door which leads to the next level

CircGR Team:

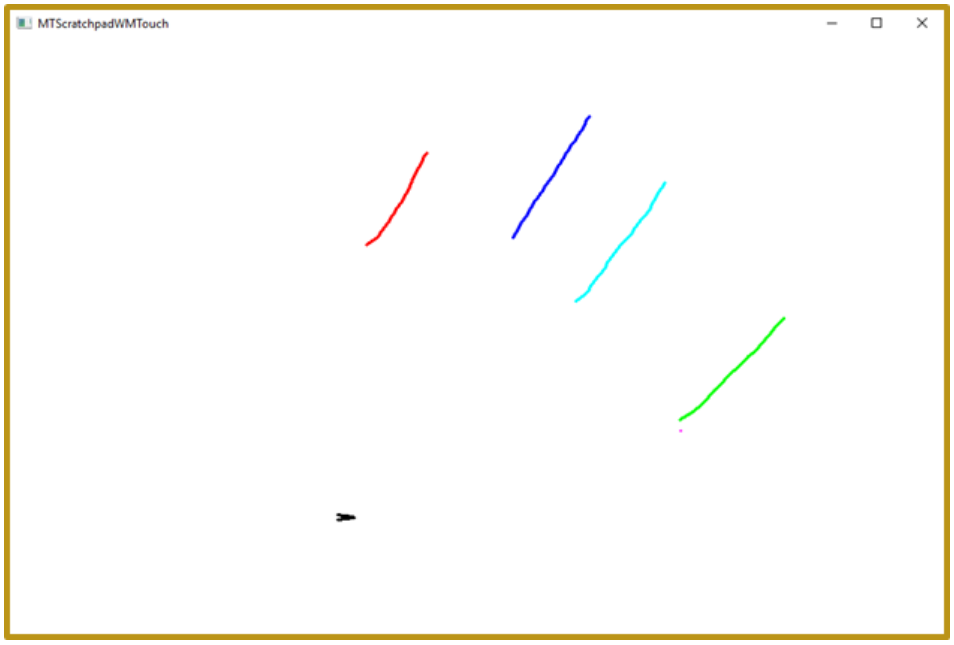


Fig. 1 - Scratchpad Application Created to Input Candidate Gestures Using Touch-Capable Device

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<Gesture Name = "A5">
<Expected As = "A5"/>
    <Stroke>
        <Point X = "880" Y = "790" T = "88" Timestamp = "41095296" Pressure = "0" />
        <Point X = "879" Y = "790" T = "88" Timestamp = "41095514" Pressure = "0" />
        <Point X = "879" Y = "790" T = "88" Timestamp = "41096372" Pressure = "0" />
    </Stroke>
    <Stroke>
        <Point X = "1537" Y = "302" T = "89" Timestamp = "41095514" Pressure = "0" />
        <Point X = "1535" Y = "305" T = "89" Timestamp = "41095561" Pressure = "0" />
        <Point X = "1531" Y = "307" T = "89" Timestamp = "41095577" Pressure = "0" />
        <Point X = "1528" Y = "311" T = "89" Timestamp = "41095577" Pressure = "0" />
        <Point X = "1519" Y = "319" T = "89" Timestamp = "41095592" Pressure = "0" />
        <Point X = "1511" Y = "327" T = "89" Timestamp = "41095592" Pressure = "0" />
        <Point X = "1504" Y = "334" T = "89" Timestamp = "41095608" Pressure = "0" />
        <Point X = "1497" Y = "341" T = "89" Timestamp = "41095624" Pressure = "0" />
        <Point X = "1490" Y = "348" T = "89" Timestamp = "41095624" Pressure = "0" />
        <Point X = "1486" Y = "352" T = "89" Timestamp = "41095639" Pressure = "0" />
        <Point X = "1479" Y = "360" T = "89" Timestamp = "41095655" Pressure = "0" />
        <Point X = "1470" Y = "369" T = "89" Timestamp = "41095655" Pressure = "0" />
        <Point X = "1462" Y = "378" T = "89" Timestamp = "41095670" Pressure = "0" />
        <Point X = "1454" Y = "386" T = "89" Timestamp = "41095670" Pressure = "0" />
        <Point X = "1446" Y = "394" T = "89" Timestamp = "41095686" Pressure = "0" />
        <Point X = "1438" Y = "403" T = "89" Timestamp = "41095686" Pressure = "0" />
        <Point X = "1431" Y = "411" T = "89" Timestamp = "41095702" Pressure = "0" />
        <Point X = "1422" Y = "420" T = "89" Timestamp = "41095717" Pressure = "0" />
        <Point X = "1414" Y = "429" T = "89" Timestamp = "41095717" Pressure = "0" />
```

Fig. 2 - Serialized gesture using XML DOM structure

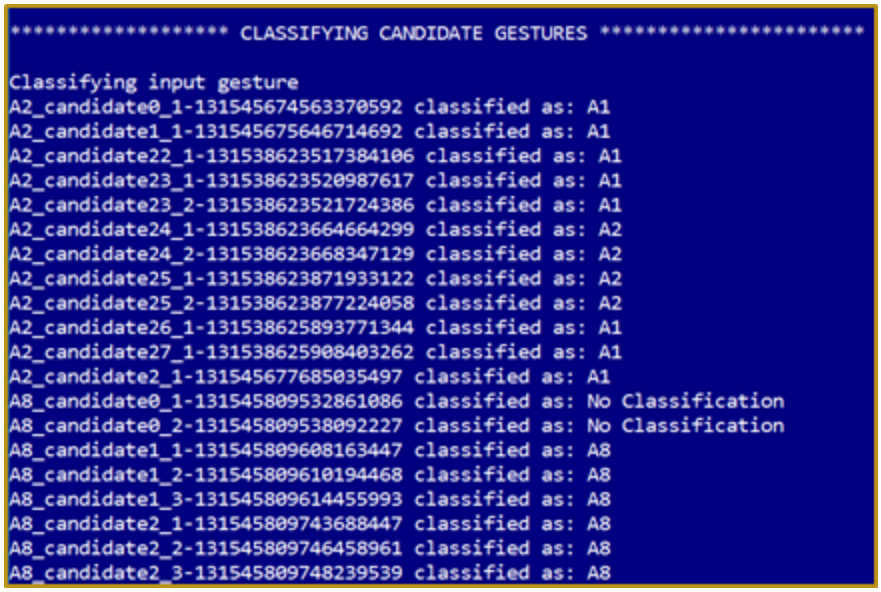


Fig. 3 - Classification Results Output Window for Comparison of Inputted Candidate Gestures against Template Gestures

LEAP-Trainer Team:

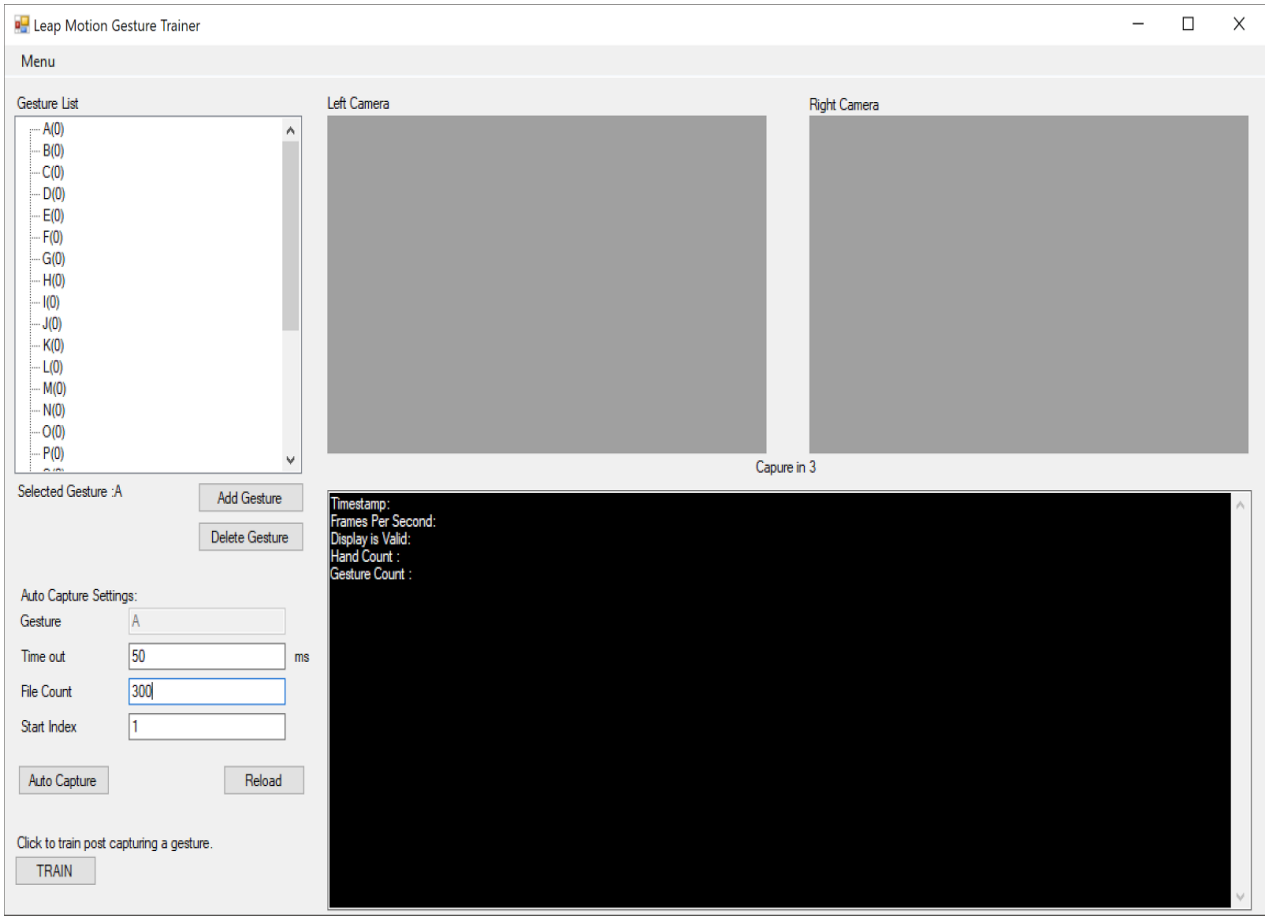


Fig 1. Training UI

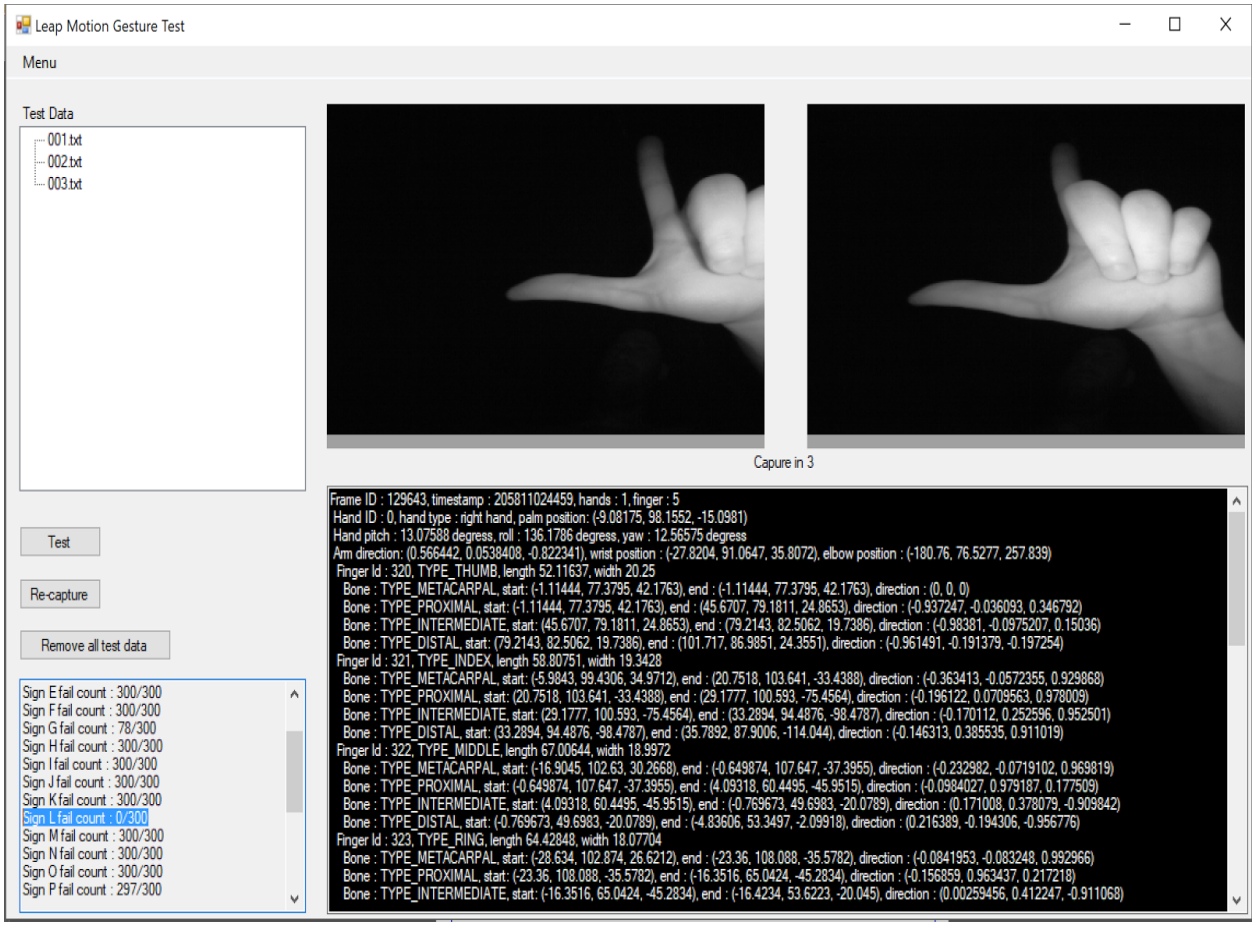

Leap Motion Gesture Test
Menu
Test Data
001.txt
002.txt
003.txt
Capure in 3
Test
Re-capture
Remove all test data
Sign E fail count : 300/300
Sign F fail count : 300/300
Sign G fail count : 78/300
Sign H fail count : 300/300
Sign I fail count : 300/300
Sign J fail count : 300/300
Sign K fail count : 300/300
Sign L fail count : 0/300
Sign M fail count : 300/300
Sign N fail count : 300/300
Sign O fail count : 300/300
Sign P fail count : 297/300
Frame ID : 129643, timestamp : 205811024459, hands : 1, finger : 5
Hand ID : 0, hand type : right hand, palm position: (-9.08175, 98.1552, -15.0981)
Hand pitch : 13.07588 degress, roll : 136.1786 degress, yaw : 12.56575 degress
Arm direction: (0.566442, 0.0538408, -0.822341), wrist position : (-27.8204, 91.0647, 35.8072), elbow position : (-180.76, 76.5277, 257.839)
Finger Id : 320, TYPE_THUMB, length 52.11637, width 20.25
Bone : TYPE_METACARPAL, start: (-1.11444, 77.3795, 42.1763), end : (-1.11444, 77.3795, 42.1763), direction : (0, 0, 0)
Bone : TYPE_PROXIMAL, start: (-1.11444, 77.3795, 42.1763), end : (45.6707, 79.1811, 24.8653), direction : (-0.937247, -0.036093, 0.346792)
Bone : TYPE_INTERMEDIATE, start: (45.6707, 79.1811, 24.8653), end : (79.2143, 82.5062, 19.7386), direction : (-0.98381, -0.0975207, 0.15036)
Bone : TYPE_DISTAL, start: (79.2143, 82.5062, 19.7386), end : (101.717, 86.9851, 24.3551), direction : (-0.961491, -0.191379, -0.197254)
Finger Id : 321, TYPE_INDEX, length 58.80751, width 19.3428
Bone : TYPE_METACARPAL, start: (-5.9843, 99.4306, 34.9712), end : (20.7518, 103.641, -33.4388), direction : (-0.363413, -0.0572355, 0.929868)
Bone : TYPE_PROXIMAL, start: (20.7518, 103.641, -33.4388), end : (29.1777, 100.593, -75.4564), direction : (-0.196122, 0.0709563, 0.978009)
Bone : TYPE_INTERMEDIATE, start: (29.1777, 100.593, -75.4564), end : (33.2894, 94.4876, -98.4787), direction : (-0.170112, 0.252596, 0.952501)
Bone : TYPE_DISTAL, start: (33.2894, 94.4876, -98.4787), end : (35.7892, 87.9006, -114.044), direction : (-0.146313, 0.385535, 0.911019)
Finger Id : 322, TYPE_MIDDLE, length 67.00644, width 18.9972
Bone : TYPE_METACARPAL, start: (-16.9045, 102.63, 30.2668), end : (-0.649874, 107.647, -37.3955), direction : (-0.232982, -0.0719102, 0.969819)
Bone : TYPE_PROXIMAL, start: (-0.649874, 107.647, -37.3955), end : (4.09318, 60.4495, -45.9515), direction : (-0.0984027, 0.979187, 0.177509)
Bone : TYPE_INTERMEDIATE, start: (4.09318, 60.4495, -45.9515), end : (-0.769673, 49.6983, -20.0789), direction : (0.171008, 0.378079, -0.909842)
Bone : TYPE_DISTAL, start: (-0.769673, 49.6983, -20.0789), end : (-4.83606, 53.3497, -2.09918), direction : (0.216389, -0.194306, -0.956776)
Finger Id : 323, TYPE_RING, length 64.42848, width 18.07704
Bone : TYPE_METACARPAL, start: (-28.634, 102.874, 26.6212), end : (-23.36, 108.088, -35.5782), direction : (-0.0841953, -0.083248, 0.992966)
Bone : TYPE_PROXIMAL, start: (-23.36, 108.088, -35.5782), end : (-16.3516, 65.0424, -45.2834), direction : (-0.156859, 0.963437, 0.217218)
Bone : TYPE_INTERMEDIATE, start: (-16.3516, 65.0424, -45.2834), end : (-16.4234, 53.6223, -20.045), direction : (0.00259456, 0.412247, -0.911068)

Fig 2. Test UI

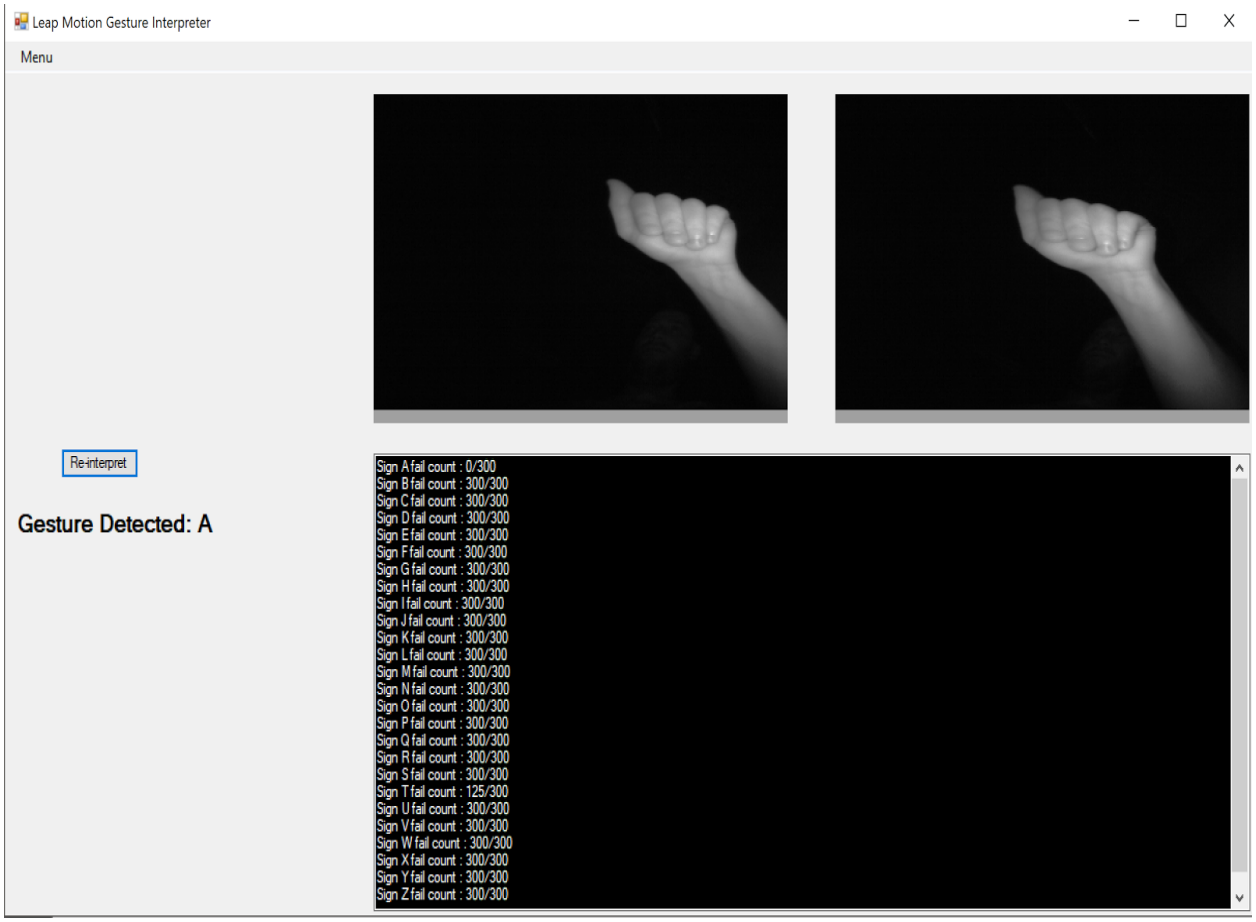


Fig 3. Interpretation UI

Appendix C - Sprint Review Reports

Sprint 1 Review Meeting Minutes
Attendees: Cristian Cabrera, Hamilton Chevez, Nicolette Celli, Filip Klepsa, Francisco Lozada, Lukas Borges
Start time: September 18, 6:00PM
End time: 6:30PM

After a show and tell presentation, the implementation of the following user stories were accepted by the product owners: All.

- User Story <Enter the user story number and title>
- User Story#670 Learn Leap API
- User Story#671 Learn Real Sense API
- User Story#672 Build a mini Leap enabled program
- User Story#685 Learn Vuforia API
- User Story#686 Augmented Reality Templace Scene
- User Story#688 Learn Unity- Nicolette
- User Story#689 Learn Blender
- User Story#690 Learn C# Syntax and Semantics
- User Story#699 Learn UE4
- User Story#700 Learn CodeVR's innerworkings and codebase
- User Story#698 learn jsoncpp
- User Story#692 Learn Qt Framework for C++
- User Story#707 Learn Unity- Kevin

The following ones were rejected and moved back to the product backlog to be assigned to a future sprint at a future Spring Planning meeting.

- User Story#697 Improve the AST parser

Sprint 2 Review Meeting Minutes
Attendees: Cristian Cabrera, Hamilton Chevez, Nicolette Celli, Filip Klepsa, Francisco Lozada, Lukas Borges, Kevin Delamo
Start time: October 29, 5:30PM
End time: 5:45PM

After a show and tell presentation, the implementation of the following user stories were accepted by the product owners: All.

- #666 Logical OR Gate in AR
- #667 Logical AND Gate in AR
- #687 Add Keyboard/Mouse Control
- #693 Add an Indicator
- #698 Learn jsoncpp
- #710 Design - Gesture Process
- #713 Research - Template Matching
- #711 Translate Point Class to C++
- #712 Translate PointMap Class to C++
- #714 Translate Geometry Class to C++
- #715 Translate Gesture Class to C++

The following ones were rejected and moved back to the product backlog to be assigned to a future sprint at a future Sprint Planning meeting.

- #697 Improve the AST parser
- #708 Options Menu to Change Game Settings
- #709 Option to Change Input Device
    - How this should be reflected on the user story definition in Mingle:
        - The user stories should be broken up into smaller user stories.

Sprint 3 Review Meeting Minutes
Attendees: Cristian Cabrera, Hamilton Chevez, Nicolette Celli, Filip Klepsa, Francisco Lozada, Lukas Borges, Kevin Delamo
Start time: October 13, 4:30PM

End time: 5:00PM

After a show and tell presentation, the implementation of the following user stories were accepted by the product owners: All.

- #673 Logic XOR Gate in AR
- #720 Logical NOT gate in AR
- #716 Translate the Recognizer class to C++
- #717 Translate the CircGesture Class to C++
- #730 Translate DirectionalEvents Class to C++
- #718 Translate the CircGR Class to C++
- #719 Translate the CircClassifier Class to C++
- #697 Improve the AST parser
- #704 Create an in-game console widget
- #708 Options Menu to Change Game Settings
- #709 Option to Change Input Device
- #721 Improve the parser
- #724 Change the Key's Material
- #725 Improve – Output of JSON file
- #726 Build – Database of ASL Alphabet
- #728 Create new labels and materials for Items
- #729 Integrate python server to UE4
- #731 Have Robot Communicate
- #732 Create Button for Completing Level
- #733 Prevent Robot from Going Through Walls
- #734 Create Lightning Door

The following ones were rejected and moved back to the product backlog to be assigned to a future sprint at a future Sprint Planning meeting.

- #722 Add a Main Menu
- #723 Add Instructions for Controls
- #727 Evaluate – Machine Learning Algorithms using Weka & Constructed database

Sprint 4 Review Meeting Minutes
Attendees: Cristian Cabrera, Hamilton Chevez, Nicolette Celli, Filip Klepsa, Francisco Lozada, Lukas Borges, Kevin Delamo
Start time: 4:30 PM
End time: 5:30 PM

After a show and tell presentation, the implementation of the following user stories were accepted by the product owners: All.

- #674 Binary Conversion Activity
- #681 Difficulty Selection for Binary Activity
- #718 Translate CircGR Class to C++
- #719 Translate CircClassifier Class to C++
- #727 Evaluate - Machine Learning Algorithms using Weka & Constructed Database
- #739 Begin To Implement – A Machine Learning Algorithm from Resulting Data
- #722 Add a Main Menu
- #723 Add Instructions for Controls
- #736 Create Impassable Areas
- #735 Create Second Level Model
- #721 Improve the AST parser
- #738 Implement classes for the parser
- #737 Move parser to C+

The following ones were rejected and moved back to the product backlog to be assigned to a future sprint at a future Spring Planning meeting.

- #691 Create GUI to Test CircGR-API

Sprint 5 Review Meeting Minutes
Attendees: Cristian Cabrera, Hamilton Chevez, Nicolette Celli, Filip Klepsa, Francisco Lozada, Lukas Borges, Kevin Delamo
Start time: 4:30 PM

End time: 5:30 PM

After a show and tell presentation, the implementation of the following user stories were accepted by the product owners: All.

- #744 Recreate the parser using JSON for modern C++ Library
- #747 Parse instances of Assign nodes
- #748 Parse instances of FuncDef nodes
- #749 Parse instances of Expression nodes
- #745 Improve the geometries with data fed from the parser
- #750 Conversion Activity Modes: Binary to Hex
- #751 Mobile Devices: Activities Menu
- #742 Create Looping Transitions
- #743 Add Left Module
- #691 Create GUI to Test CircGR-API
- #746 Test & Debug MTCircGR API
- #741 Continue to Implement - A Machine Learning Algorithm from Resulting Data
- #754 Add Right Turn Module
- #755 Create Third Level
- #757 Proceed to Next Level

The following ones were rejected and moved back to the product backlog to be assigned to a future sprint at a future Spring Planning meeting.

- #755 Create Third Level

Sprint 6 Review Meeting Minutes
Attendees: Cristian Cabrera, Hamilton Chevez, Nicolette Celli, Filip Klepsa, Francisco Lozada, Lukas Borges, Kevin Delamo
Start time: 3:00 PM
End time: 4:00 PM

After a show and tell presentation, the implementation of the following user stories were accepted by the product owners: All.

- #756 Finish Implementing - A Machine Learning Algorithm

The following ones were rejected and moved back to the product backlog to be assigned to a future sprint at a future Spring Planning meeting.

-

Appendix D - User Manuals, Installation/Maintenance Document, Shortcomings/Wishlist Document and other documents

- **MANUALS**
  - **1**
  - **2**
  - **3**
  - **4**
  - **LEAP-TRAINER TEAM:**
    - **USER TUTORIAL: RECORDING AND TRAINING GESTURES**

      1. WHEN YOU FIRST OPEN THE PROGRAM YOU WILL FIND YOURSELF ON THE TRAINING FORM.
      -IN THE TOP LEFT CORNER, YOU WILL HAVE A DROPDOWN MENU. THERE YOU WILL BE ABLE TO NAVIGATE TO TEST OR INTERPRETATION FORMS.

      2. TO TRAIN GESTURES FIRST YOU MUST SELECT EITHER ONE OF THE BUILT-IN ENGLISH LETTERS (A IS DEFAULT) OR CLICK ON THE "ADD GESTURE" BUTTON TO CREATE A CUSTOM ONE.
      - YOU CAN SELECT THE GESTURE YOU WISH TO TRAIN BY DOUBLE CLICKING ON THE NAME WITHIN THE GESTURE LIST.
      - AT THIS POINT YOU MUST SPECIFY THE AUTO CAPTURE SETTINGS:
      -TIME OUT= TIME BETWEEN FRAMES (500 MS DEFAULT) I FIND 50 OR 100 TO BE MOST EFFECTIVE HOWEVER, THIS CAN BOG DOWN SLOWER MACHINES.
      - FILE COUNT= NUMBER OF FRAMES YOU WILL CAPTURE IN A SINGLE SESSION. 40 IS DEFAULT TO AVOID HAND FATIGUE HOWEVER, THE INTERPRETATION WILL NOT BE ACCURATE WITHOUT AT LEAST A TOTAL OF APROX 300 FRAMES. MORE IS BETTER!
      - START INDEX= WHERE THE PROGRAM BEGINS COUNTING FROM, I RECOMMEND LEAVING THIS AT 1.
      - IF YOU WISH TO REVIEW THE DATA RECORDED, SIMPLY ENTER THE CONTAINER OF YOUR SELECTED GESTURE AND CLICK ON THE NUMBERED .TXT FILES.

3. Hit "Auto Capture" to activate the Leap Motion Controller.

4. Hit "Start" once you are ready to record.

5. When you have finished recording your gestures, it is now time to train the application. Hit the TRAIN button on the bottom and select the recorded gestures you wish to train.

6. All recorded gestures can be found within the Capture\ directory under its associated name. The data is a Well-formed JSON which can be converted to a CSV for data analysis in excel.

- **User Tutorial: TESTING TRAINED GESTURES**

1. Use the dropdown Menu to navigate to the Testing Form.

2. Hit the "Test" button to activate the Leap Motion Controller.

3. Place your hand over the device Field of View and perform the gesture you wish to test.

4. Hit the "Capture" button. The application will then tell you the Miss Rate of the performed gesture versus the learned recording.

5. You can now hit the Re-Capture button if you wish to test again.

6. Since the test data is saved, you might wish to delete the data post testing. If so, hit the "Remove all test data" button.

- **User Tutorial: INTERPRETATION MODE**

  1. After training and testing the accuracy of your recorded gesture, it is now time to show it off. Use the dropdown Menu to navigate to the Interpreter Form.

  2. The Leap Motion Controller will be already active in this state. Simply perform your gesture and hit the "Interpret" button. The program will then try to decipher the gesture you performed and return its name in the "Gesture Detected:" field.

  3. If you are unsatisfied with the result, try again by hitting "Reinterpret".

- **Installation**
  - **ARCSE Team:**
    - Download the latest version of Blender
      - https://www.blender.org/download/
    - Download Unity v 5.6.1 in the download archive
      - https://unity3d.com/get-unity/download/archive
    - Pull the latest project revision from GitHub into a local repository.
      - Included will be the Vuforia plugin used for the project.
    - Run Unity and open the project from your local repository.
      - The Scenes folder includes the scenes that were finished this iteration.
  - **CodeAdventures Team**
    - Download the latest version of Blender
      - https://www.blender.org/download/
    - Download Unity v 5.6.1 in the download archive
      - https://unity3d.com/get-unity/download/archive
      - Note: Downloading Unity will give you the option to download Microsoft Visual Studios; if not already installed, do so now.
    - Pull the latest project revision from GitHub into a local repository
    - Run Unity and open the project from your local repository

- **CodeVR Team:**
- **LEAP-Trainer Team:**

NOTE: the following guide is written with Visual Studio 2017 and C# programming in mind. I am assuming you will install the controller drivers, visual studio IDE, and the C# .Net Framework 4.0 before proceeding.

- Located in the LeapDeveloperKit_2.3.1+31549_win directory, you will find the **LeapSDK** as well as the **Controller installation executable**.

NOTE: When your application calls a function in the Leap Motion C# API, the C# code calls the matching function defined in LeapC.dll. The Leap Motion libraries are designed to be loaded from the same directory as your application executable. You are expected to distribute the appropriate Leap Motion library with your application. The Leap Motion libraries are located in the \LeapSDK\lib directory.

To add Leap Motion support to a new or existing project:

1) Make a **LEAP_SDK** system environment variable to point to your Leap Motion SDK directory. You must restart your IDE after creating this variable, or it will not be recognized. (As a reminder, you can create and change system environment variables from the Windows System Properties dialog.)
    a. Hit windows key, type: "**edit the system environment variables**"
    b. With system properties open, select **Environment Variables**.
    c. In *System variables*, select **New.**
    d. In *Variable name* field, type: "**LEAP_SDK"**
    e. In *Variable value* field, type the full path of the **LeapSDK** directory.
    f. Hit **Ok.**
2) Open or create a new project in Visual Studio of the desired Windows Forms or WPF type.
3) Add a reference to one of the pre-built .NET libraries:
    a. Select **Project** > **Add Reference**.
    b. In the *References* dialog, select **Browse**.
    c. Browse to your Leap Motion SDK folder.
    d. Choose *LeapCSharp.NET4.0.dll.* located in \LeapSDK\lib.
    NOTE: do not add any of the other Leap Motion library files as a reference.
4) Set the target platform to x64:
    a. Open the project properties page with **Project** > *ProjectName* **Properties**.
    b. Select the **Build** page.
    c. Set the **Configuration** to *All Configurations.*

d. Under General, set **Platform target** to *x64*.
NOTE: Do not use the *Any CPU* target.

5) Edit the *Post Build Event* to copy the native libraries to the project's target executable directory.
   a. With the project properties page open, select the **Build Events** page.
   b. Click **Edit Post-build**
   Add: xcopy /yr "$(LEAP_SDK)\lib\x64\Leap.dll" "$(TargetDir)"
   xcopy /yr "$(LEAP_SDK)\lib\x64\LeapCSharp.dll" "$(TargetDir)"

- # Maintenance
  - ### ARCSE Team:
    - The ARCSE project requires no additional maintenance.
  - ### CodeAdventures Team:
    - The CodeAdventures game requires no additional Maintenance.
  - ### CodeVR Team:
    -
  - ### LEAP-Trainer Team:
    - The LEAP-Trainer requires no additional maintenance.
- # Shortcomings
  - ### 1
  - ### 2
  - ### 3
  - ### 4
  - ### LEAP-Trainer Team:
    - The accuracy of the software: The letters "J" and "Z" are completely unclassifiable within a single frame because they require motion to be represented. The accuracy of "M," "N," and "T" are unreliable due to occlusion. "G" and "Q" can be confused with each other, similarly With "A" and "S", due to their resemblance in their representation, which causes similar angles to be generated. If they manage to fall within the

acceptable boundaries, then the LEAP-Trainer will consider it a pass even if it wasn't the users' intended gesture.

- **Wishlist**
  - **ARCSE Team:**
    - Remove the use of the virtual buttons and use the touchscreen of the mobile device.
    - Include an option for typing in responses.
    - Create and connect the application to a restful api for getting more questions and submitting answers.
  - **CodeAdventures Team:**
    - Prevent the Vive controllers from phasing through walls.
    - Add Ability to link multiple modules to one module in order to save world space.
    - Add ability to destroy transitions.
  - **CodeVR Team:**
    -
  - **LEAP-Trainer Team:**
    - Introduce a system that can support multi-framed gestures.
    - Extend software to utilize more than one controller.

References

1

2

3

4

LEAP-Trainer Team:

- Leap Motion, http://www.leapmotion.com (last access: Dec 2017)
- “American sign language – National Association of the Deaf” http://nad.org/issues/american-sign-language (last access: Dec 2017)
- Fok, Kai-Yin, et al. "A real-time asl recognition system using leap motion sensors." Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), 2015 International Conference on. IEEE, 2015.
- Chuan, Ching-Hua, Eric Regina, and Caroline Guardino. "American sign language recognition using leap motion sensor." Machine Learning and Applications (ICMLA), 2014 13th International Conference on. IEEE, 2014.
- Cover, Thomas, and Peter Hart. "Nearest neighbor pattern classification." IEEE transactions on information theory 13.1 (1967): 21-27.
- Leap Motion CSharp API, https://developer.leapmotion.com/documentation/v2/csharp/devguide/Leap_Overview.html (last access: Dec 2017)
- ASL hand-shape chart, http://resources.seattlecentral.edu/faculty/bbernstein/ASL101/handshapechart.gif (last access: Dec 2017)