Florida International University
School of Computing and Information Sciences

Software Engineering Focus

Final Deliverable

Project Title: WEB-VR: Towards Virtual and Augmented Reality for the WEB 1.0

Team Members: Hamilton Chevez, Pachev Joseph, Daniel Khawand, Bernardo Pla, Daniel Rivero

Product Owner(s): Francisco Ortega

Mentor(s): Francisco Ortega

Instructor: Masoud Sadjadi

Abstract

This document presents the information necessary to gain a good understanding of "Web-VR: Towards Virtual and Augmented Reality for the Web 1.0," a research and implementation based project that is overseen by Dr. Francisco Ortega, and all of 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 three separate teams in the design, implementation, and research process:* ***WebVR Education, WebVR Gaming, and WebVR Input.*** *Each team has its own section in this document.*

Contents

INTRODUCTION

This following section describes the status of the original system for the WEBVR project. Also in this section, the new system is described within the context of this project.

Current System

Prior to the start of this project, there is no previous software system implemented. The WEBVR Education Team, WEBVR Gaming Team, and WEBVR Input Device Team relies on the use of research papers to obtain starting points for the project.

Purpose of New System

For the WEBVR Education Team, the purpose of this system is to visualize computer science topics through the use of augmented and virtual reality technologies. The topics include but not limited to, abstract data structures and algorithms. The goal of creating this system is to aid in increasing the retention rate of Computer Science students.

For the WEBVR Gaming Team, the purpose of project is to incorporate programming concepts like: variable assignment, method calling, boolean logic, conditional statements and loops into a virtual reality video game. The goal is to introduce these programming concepts in an implicit but engaging way. Using virtual reality the player is encouraged to to uses these programming concepts in a hands on way to complete each level. In doing so, this will reinforce or introduce the player to these programming concepts.

For the WEBVR Input Device Team, the purpose of the new system is to convert the information derived from the research papers into tangible code in the Rust Language. This is handled in the form of a library designed to handle different input devices. The device library represents a supported device as an object with 6-tuples. The tuples represent the following: Connection, Input, State, Output, Resolution, and Events. Upon implementation of the device library, the system can integrate with different applications and support a variety of input devices, including virtual reality devices.

USER STORIES

The following section provides the detailed user stories that were implemented in this first iteration of the WEBVR 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

WEBVR Education Team

User Story ID #681 Selection Sort Algorithm Scene

Description:

- As a user, I would like to see data be sorted using Selection Sort so I can understand the algorithm better.

Acceptance Criteria:

- The sorting pace is slow enough to not lose distraction.

User Story ID #691 Stack Interface Scene

Description:

- As a user that uses this application, I would like to see a stack data structure be implemented visually, so I can understand better the stack interface.

Acceptance Criteria

- System illustrates Pop action.
- System illustrates Push action.
- System illustrates LiFo property.

User Story ID #684 View Binary Search Tree Scene

Description:

- As a User, I would like to see how a Binary Search Tree gets created so I can comprehend the insertion of new elements.

Acceptance Criteria:

- The animation is not rushed.
- User's camera is moved along with the time frames of the scene.

User Story #679 View Merge Sort Scene

Description:

- As a user I would like to see data be sorted using Merge Sort so I can see how the data gets split, sorted, and merged.

Acceptance Criteria

- The user can see an explanation of the sorting algorithm.
- The animation is not rushed.

WEBVR Gaming Team

User Story Name: #620 Movement Controls

Description:

- As a user who plays the game, I would like to have an easy time moving around, with familiar controls, so that I can focus on immersing myself in the game.

Acceptance Criteria:

- Movement controls should be familiar to that of a first-person game.
- Controls need to be mapped to the HTC Vive.
- Must run in Unity

User Story Name: #672 Head Tracking

Description:

- As a user who plays the game, I want to be able to look around in this virtual world, like how I would look around in reality, so that I can further immerse myself into the game.

Acceptance Criteria:

- Head tracking motion should feel natural.
- Requires a virtual reality headset.
- Must be integrated with Unity

User Story Name: #673 Module collision detection

Description:

- As a user who plays this game, I want the robot to be able to detect modules that are placed on its programming board so that I can give commands to the robot.

Acceptance Criteria:

- Functionality of the board should be animated
- The board should only detect collisions with objects that are tagged ‘Module’

User Story Name: #675 Robot Controller

Description:

- As a developer who programs this game, I want the controller for the robot to be the interface between what the user does when programming the robot with the actions provided to the robot by the back-end.

Acceptance Criteria:

- Controller should be programmed using the MonoBehavior interface from unity.
- Controller should be able to interface between front-end assets to the state machine.

User Story Name: #678 Locked Gate

Description:

- As a user who plays this game, I want each door object to have properties about its color, state, and function to open so that my programmable robot can manipulate the properties and call functions.

Acceptance Criteria:

- Some doors require a key parameter for their open function.

User Story Name: #676 Transition Visualization and Logic

Description:

- As a user who plays this game, I want to connect modules on the programming board so that I can logically transition from one function to the next to issue commands to the programmable robot.

Acceptance Criteria:

- Transitions should be made by click on one module then dragging to the next
- Outside libraries can be used.

User Story Name: #719 Level 1

Description:

- As a user who plays this game I want level one to enforce the programming concepts of method calling and variable declaration and parameter passing so that I can understand how to interact with the programmable robot and it with it’s environment.

Acceptance Criteria:

- User must program the robot in the correct way to complete the level
- User must look around his/her environment to find modules that give the programmable robot functionality
- Puzzles for this level should be designed for persons with little to no programming experience.

User Story Name: #720 Level 1 - Realization

Description:

- As a developer who develops this game, I want to finalize and realize level one. Level one should introduce programming concepts of function calling and the logical sequence of events in programming to expose a user to basic programming concepts.

Acceptance Criteria:
- Should not be difficult to complete
- The main object is for the user to program the robot to pick up a key to open a door

User Story Name: #745 Statemachine

Description:
- As a user who plays this game, I want to connect a set of logical events together based on logic I have deduced so that I can execute commands to the robot. I want this to be designed as a state-machine so that I can transition between logical events in a nice way.

Acceptance Criteria:
- Each state must represent a logical event in the world
  - I.e. moveForward, moveBackwards
- Each transition must represent the sequence of logical events
- Guards must be implemented so that certain transitions can be taken based on some condition

WEBVR Input Device Team

User Story Name: #692 Design Connection For Mouse

Description:
- As a developer I want to connect a mouse to a computer and have universal way of representing its connection and receiving output.

Acceptance Criteria:

- Input can be read by device that it is connected to
- Connection is established and represented as a tuple
- Data output is sent in a universal JSON format

User Story Name: #722 Input Device Events - Mouse

Description:

- As a developer, I want to create mouse input device events, so that I can handle states and update device tuple accordingly.

Acceptance Criteria:

- Handle Mouse Events.
- Demo is accepted by product owner

User Story Name: #752 Video Game Controller

Description:

- As a developer, I would like a modular interface to connect a video game controller to any computer and have it be usable.

Acceptance Criteria:

- Game controller can be read from device
- Game controller is able to accept input and turn it into output for developers to use
- Game controller format is outputted in JSON to allow developers to easily read and modify

User Story Name: #758 Generic Empty Device Implementation

Description:

- As a Developer, I want to create a class in rust to handle generic or "unsupported" devices, so that the library can support as many devices as possible.

Acceptance Criteria:

- Empty Device fulfills the 7-tuple device requirement
- Demo is accepted by the Product owner.

User Story Name: #777 Web Application for Implemented Devices

Description:

- Description: As a developer, I want to create a Web Application so that a graphical output of using the Rust library can be displayed

Acceptance Criteria:

- Web Application can successfully communicate with the Rust device library
- Application can track updates using the Rust library
- Demo approved by product owner

User Story Name: #783 Debug Device Library for Implemented Devices

Description:

- As a Developer, I want to perform testing on Device Library so that any errors found can be fixed and to improve previous features.

Acceptance Criteria:

- Reports on bugs found in system are presented.
- Bugs found are fixed
- Work is accepted by product owner

User Story Name: #788 Selection Sort using WebVR API

Description:

- As a user, I want to observe a selection sort on a web page with a VR environment.

Acceptance Criteria:

1. VR Capability
2. Illustrates each iteration of the Selection Sort.

User Story Name: #779 WebVR integration to 3d scenes

Description:

- As a user, I would like to see the WebVR representation of the Three.js dynamic structures so that I can get the learning experience through the VR device.

Acceptance Criteria

- User may switch to VR mode at any time in a scene
- The VR and web page view must be identical

User Story Name: #767 Stack Interface Scene

Description:

- As a user, I want a dynamic Stack Interface Scene where I can push and pop data items onto a stack in a WebVR or WebGL environment.

Acceptance Criteria:

1. Illustrates LIFO property
2. Visualization of data

User Story Name: #725 Implement Hash-Table Interface Scene

Description:

- As a student, I would want a fully functional Hash-Table Interface Scene in WebGL or WebVR so that I could understand Hash-Tables better.

Acceptance Criteria:

1. Collision Handling Illustrated
2. 3D Visualization

User Story Name: #690 Implement Queue Interface Scene

Description:

- As a student that use this application, I would like to see a queue data structure be implemented visually, so I can understand better the queue interface.

Acceptance Criteria:

1. System notifies user of Enqueue action
2. System notifies user of Dequeue action
3. System illustrates FiFo property.

Pending User Stories

This section focuses on the user stories that were not implemented during this iteration of the WEBVR project.

The pending stories are as follows:

- #680 Quick-Sort Algorithm Scene
- #682 Depth First Traversal
- #683 Breadth First Traversal
- #685 User Defined Data for Model
- #686 Doubly Linked List Insertion
- #689 Balance Binary Trees
- #763 Touch Device Implementation
- #778 Integrate Web Application with Servo Browser

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:

WEBVR Education Team:

- Software Resources:
  - A-Frame WebVR Javascript Framework
  - WebVR Api
  - Three.js WebGL Library
  - Cannon.js WebGL Library
  - Chromium Web Browser - Nightly Build
  - FireFox Developer Edition Browser
  - Visual Studio Code IDE
  - NPM 5.0.3
  - Gulp
- Hardware Resources
  - Google Cardboard-Style headset
  - Nexus 6P Android Phone
  - Oculus Rift VR Headset
  - Macbook Pro
    - Intel i7 Kaby Lake quad-core Processor
    - 16gb Ram
    - Intel HD Graphics 630

WEBVR Gaming Team:

- Hardware Resources
    - HTC Vive
    - Dell Alienware Laptop
        - Intel Core i7-6700HQ quad-core processor
        - 16GB RAM
        - NVIDIA GeForce GTX 970M graphics card
- Software Resources
    - Unity 3D 5.3.4 or higher
    - Windows 8 or higher
    - C# Visual Studio Development Environment
    - SteamVR plugin for Unity

WEBVR Input Device Team:

- Hardware Resources
    - Computer Resources
        - Lenovo Z50-75
            - AMD FX processor
            - 8GB of RAM
        - Macbook Air
            - Intel Core i7
            - 8GB of RAM
    - Input Devices
        - Logitech M510 wireless mouse
        - Sony Playstation 4 Controller
        - Xbox 360 Controller
        - Leap Motion controller
        - Acer 19-inch Touchscreen Monitor
- Software Resources
    - Ubuntu 16.04
    - Rust nightly build
    - Cargo nightly build
    - Visual Studio Code v1.14.1
    - VIM Editor
    - Servo Browser nightly build

Sprints Plan

The following is a summary of the planning meetings involved for this iteration of the WEBVR project.

Sprint 1

Sprint Planning Meeting Minutes: May 26th, 2017

Attendees: Hamilton Chevez, Bernardo Pla, Daniel Khawand, Daniel Rivero, Pachev Joseph
Start time: 19:30
End time: 20:15

After discussion, the velocity of the team were estimated to be 142

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

- #719 Level 1 -24 pts
- #692 Design Connection For Mouse - 16 pts
- #721 Add Connection interface for USB Mouse- 8 pts
- #696 Setup Servo Browser- 18 points
- #681 Create Selection Sort Algorithm - 8 pts
- #691 Create Stack Interface Scene - 10 pts
- #687 WebVR Scene User Movement - 8 pts
- #690 Implement Queue Interface Scene - 10 pts
- #725 Implement Hash-Table Interface Scene - 10 pts
- #713 Oculus Rift Testing/Setup - 4 pts
- #722 Input device events - mouse - 16 pts
- #726 Native APIs - 10 pts

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

- Daniel Rivero
    - #719 Level 1
        - Very willing
- Hamilton Chevez
    - #691 Create Stack Interface Scene

        - Very WIlling
    - #681 Create Selection Sort Algorithm
        - Very Willing
- Pachev Joseph
    - #721 Add connection Interface for USB Mouse
        - Very willing
    - #692 Design Connection For Mouse
        - Very willing
    - #696 Setup Servo browser
        - Willing
- Bernardo Pla
    - #722 Input Device Events - Mouse
        - Very willing
    - #726 Native APIs
        - Very willing
    - #669 Rust Programming Language
        - Very willing

- Daniel Khawand
    - #687 WebVR Scene User Movement
        - Very Willing
    - #690 Implement Queue Interface Scene
        - Very Willing
    - #713 Oculus Rift Testing/Setup
        - Very Willing
    - #725 Implement Hash-Table Interface Scene
        - Willing

Sprint 2

Sprint Planning Meeting Minutes: June 11, 2017

Attendees: Hamilton Chevez, Bernardo Pla, Daniel Khawand, Daniel Rivero, Pachev Joseph
Start time: 17:15
End time: 18:45

After discussion, the velocity of the team was estimated to be 94% or 134 points. For this coming sprint, we expect to accomplish 200 points, for an average expected velocity of 167 points.

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

- #752 Video Game Controller- 24 pts
- #754 Touch input - 24pts
- #758 Generic Empty Device Implementation - 24 pts
- #762 Touch Device Implementation - 24 pts
- #765 Design Website Theme - 4pts
- #762 Selenium Testing Tool - 4 pts
- #684 Binary Search Tree Scene - 12 pts
- #637 [Bot Design] Module Collision- 24 pts
- #675 Robot Controller -24 pts
- #690 Queue Interface Scene - 4 pts
- #725 Implement Hash-Table Interface Scene - 4 pts
- #767 Dynamic Stack Interface Scene - 24 pts
- #768 Visualization Research - 4 pts

- Pachev Joseph
  - #752 Video Game Controller
    - Very willing
  - #754 Touch Input

        - Very willing

- Bernardo Pla
    - #758 Generic Empty Device Implementation
        - Very willing
    - #763 Touch Device Implementation
        - Very willing

- Daniel Rivero
    - #675 Robot Controller
        - Very Willing
    - #637 [Bot Design] Module Collision
        - Very Willing
- Hamilton Chevez
    - #684 Binary Search Tree Scene
        - Very Willing
    - #762 Selenium Testing Tool
        - Very Willing
    - #765 Website Design Theme
        - Very Willing

- Daniel Khawand
    - #690 Implement Queue Interface Scene
        - Very Willing
    - #767 Dynamic Stack Interface Scene
        - Very Willing
    - #725 Implement Hash-Table Interface Scene
        - Very Willing

Sprint 3

Sprint Planning Meeting Minutes: June 25, 2017

Attendees: Hamilton Chevez, Pachev Joseph, Bernardo Pla, Daniel Rivero, Daniel Khawand
Start time: 10:30 PM
End time: 11:30 PM

After discussion, the velocity of the team was estimated to be 100% or Z points. For this coming sprint, we expect to accomplish A points, for an average expected velocity of B points.

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

- #684 Binary Search Tree Scene - 12 points
- #679 Merge Sort Scene - 8 Points
- #754 Touch input - 24pts
- #696 Setup Servo Browser- 24pts
- #777 Web Application for Implemented Devices - 24pts
- #778 Integrate Web Application with Servo Browser - 24pts
- #779 WebGL Demo Function Implementation - 24 pts
- #780 Research on WebVR - 8 pts
- #676 Transition Visualization & Logic -24pts
- #720 Complete realization of level 1 - 24pts

- Pachev Joseph
  - #754 Touch Input
    - Very Willing
  - # 696 Setup Servo Browser
    - Very Willing
- Bernardo Pla
  - #777 Web Application for Implemented Devices
    - Very Willing
  - #778 Integrate Web Application with Servo Browser
    - Very Willing

- Daniel Rivero

-
  - #676 Transition Visualization & Logic
    - Very Willing
  - #720 Complete realization of level 1
    - Very Willing
- Hamiton Chevez
  - #684 Binary Search Tree Scene
    - Very Willing
  - #679 Merge Sort Scene
    - Very WIlling

- Daniel Khawand
  - #779 WebGL Demo Function Implementation
    - Very Willing
  - #780 Research on WebVR
    - Very Willing

Sprint 4

Sprint Planning Meeting Minutes: July 8th, 2017

Attendees: Hamilton Chevez, Pachev Joseph, Bernardo Pla, Daniel Rivero, Daniel Khawand
Start time: 4:10 PM
End time: 5:00 PM

After discussion, the velocity of the team was estimated to be 100% or Z points. For this coming sprint, we expect to accomplish A points, for an average expected velocity of B points.

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

- #783 Debug Device Library for Implemented Devices - 24 pts
- #784 VIP Documentation - 24 pts
- #785 VIP Documentation - 12pts
- #787 Debug WebVR Scene Usage - 10 pts
- #786 Research Documentation - 12 pts
- #674 Robot Inventory - 16pts
- #677 User Inventory - 16pts
- #788 Selection Sort Using WebVR API - 24 pts
- #789 VIP Final Documenation - 12 pts

- Pachev Joseph
    -
        - Very Willing
- Bernardo Pla
    - #783 Debug Device Library for Implemented Devices
        - Very Willing
    - #784 VIP Documentation
        - Very Willing

- Daniel Rivero
    - #674 Robot Inventory
        - Very Willing
    - #677 User Inventory

        - Very Willing
- Hamiton Chevez
    - #785 VIP Documentation
        - Very Willing
    - #787 Debug WebVR Scene Usage
        - Very Willing
    - #786 Research Documentation
        - Very Willing

- Daniel Khawand
    - #788 Selection Sort Using WebVR Api
        - Very Willing
    - #789 VIP Final Documentation
        - Very Willing

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

WEBVR Education Team:

For A-Frame:

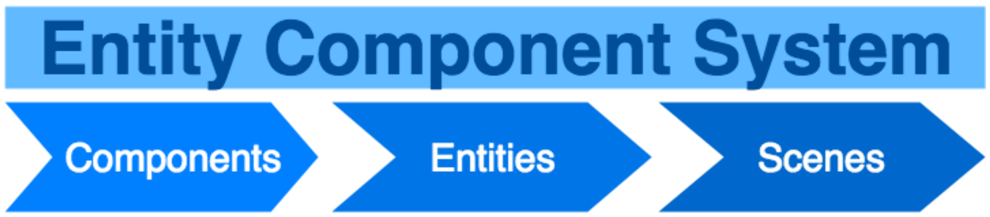


Figure 1: Entity Component System for WEBVR Education Team

The A-Frame framework uses an Entity Component System architecture for their visualizations. The Scene object plays the role of the System, it takes care of setting up the WebGL renderer, the canvas for entities, default camera and light behavior, and connect the UI to the WebVR api. Each Scene object is composed of one or more entity objects. Entities are the primary unit of work for visualizations. They are placeholder objects for which components plug into. Components are reusable and modular objects that can be used to modify the appearance, behavior, and/or functionality of entities. A-Frame is further extendable by the community of user generated components. For future uses of the system, if there is a requirement for an advanced feature or a need for a specific behavior that doesn't exist yet, it can be implemented by creating a custom component to meet that need.

For WebVR API and 3D Scenes (not A-Frame):

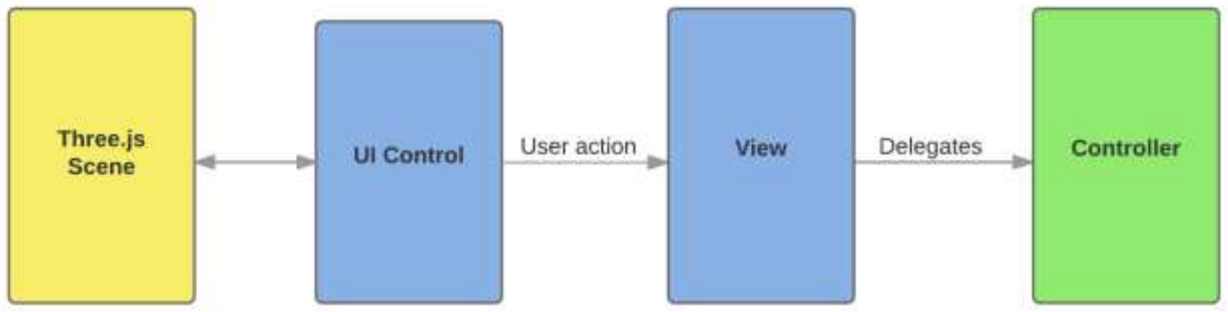


Figure 2: Model-View-Controller for 3D scenes without A-Frame

The **MVC (Model-View-Controller)** architecture was used for the Abstract Data Structure Visualization scenes because each visualization/scene can be treated as a view, the *Threejs* objects as models, and the Javascript code that renders the scene and performs operations on the objects as the controller(s). Eventually, the controller(s) will be improved upon in future releases to include more user input from various sources.

The **models** for these scenes are whatever 3D Objects are rendered to represent the data for the user. The **views** are the various interfaces for the users. The controllers will have to call Javascript functions that will input user data, generate and render the appropriate objects from that data and for the respective view, and then alter the view.

The **controllers** are currently very primitive for the VR scenes but have been mostly implemented for the *threejs* and *cannonjs* scenes. The Cannon scene has a demo controller that the user can reference in their current view to change the state of the data structure that they are currently visualizing.

WEBVR Game Team:

We decided to structure the system following a classic **Model - View - Controller(MVC)** as the primary architecture for the game. This architecture is appropriate for our system because we can treat the unity scene as a view, game objects as models and the scripts given to the game objects controllers. This allows us to structure the project in a modular and scalable way enabling us to separate the application's data, logic and user interface.

System and Subsystem Decomposition

WEBVR Game Team:

Models

- ***States:*** This model represents a classical state from a state machine. It has two properties, an **Id** that uniquely identifies the state and a **Type** which represents the function given to the robot to issue an action.
- ***Transitions:*** This model represents a classic transition from a state machine. It has three properties, an **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:*** This model represents the gameobject that holds the properties of the state model. Each module is unique and is tagged with the appropriate function used to issue commands to the robot.
- ***Robot:*** This model is the main programmable robot that the player interacts with. It has built in sliders for the speed and collision detection.
- ***Board:*** This model is where a state machine is built by attaching modules together.
- ***Tether:*** A very simple model that is a visual representation of a transition between two states. These models are instantiated whenever the player attaches two modules together.

Views

- ***Unity Scene:*** The main view of the project. This is the scene that the player sees. It is in this view that each model gets instantiated in.

Controllers

- ***Robot Controller:*** Controls the robot. Has built in states that monitor what action the robot is doing in realtime. Has knowledge of what states and transitions have been made to create a state machine. Has knowledge of every module in the game and the execution code for each module.
- ***State Machine Controller:*** This controller is for the state machine created by the robot controller. The state machine can build itself based on a list of states and transitions. The state machine has a transition function that goes to the next state in the state machine.

- ➢ ***Board Controller:*** This controller handles everything that has to do with recognizing a module placed on or removed from the board.This controller converts module data into state to transition data and sends it to the robot controller
- ➢ ***Vive Controller:*** This controller is a series of input commands for the HTC Vive controller that add functionality to the game. This controller is responsible for opening and closing the board of the robot. It has the logic to bring up the main menu of the game and interact with game objects in the scene.

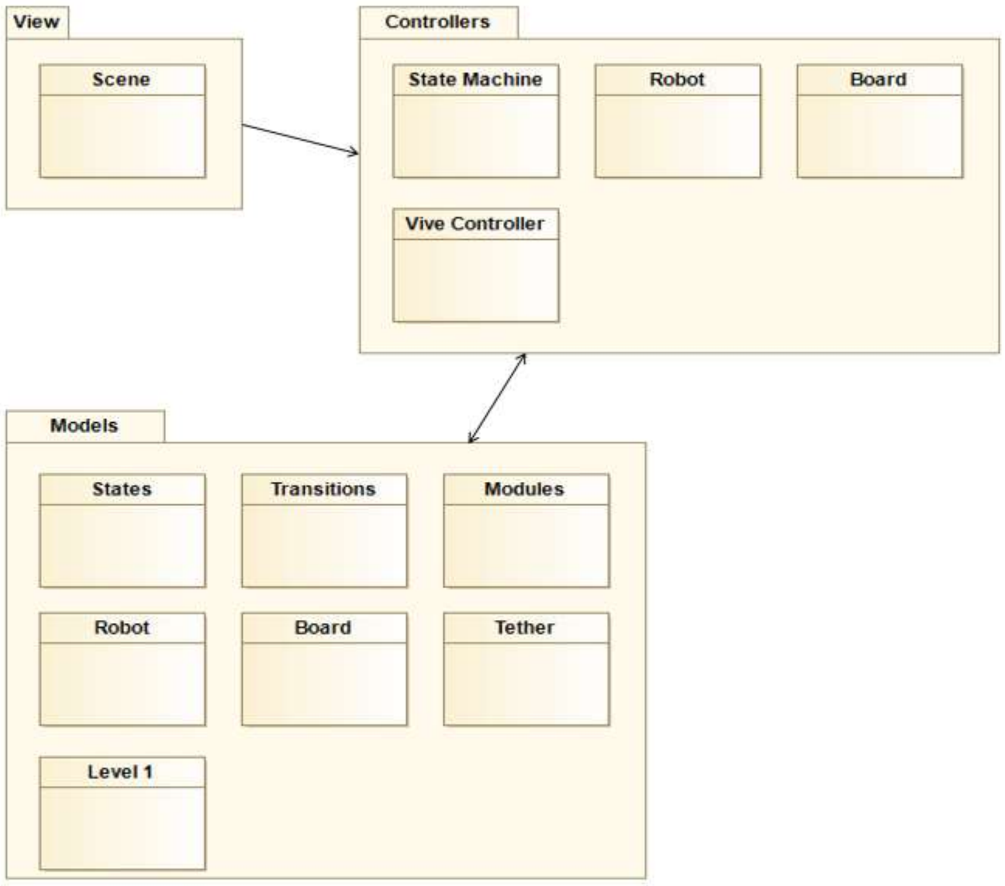


Figure 3: Depicts the WebVR Game System and Subsystem Decomposition

WEBVR Input Device Team:

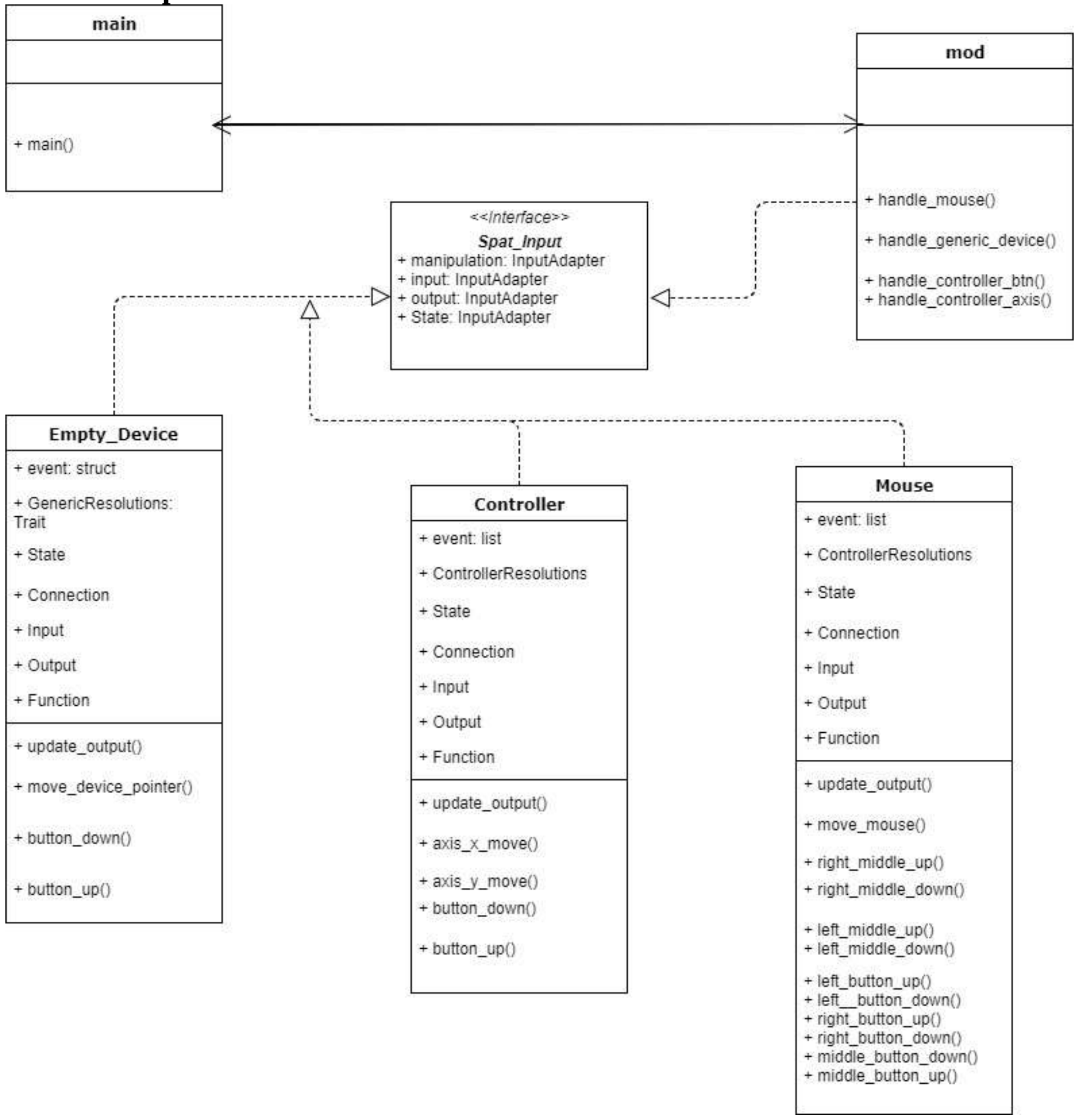


Figure 4: System Class Diagram - This diagram represents the interactions with respect to the input device library.

In the device library, the system is organized with a simple directory structure. This structure is initialized by the Cargo package manager for Rust. For this instance, Cargo creates a source folder in which all the code can be stored. The directory is organized as follows:

- webvr-input: This is the root folder. All the code, configuration files, and Readme documents live in here.
    - Cargo.toml is the configuration file used when compiling the library.
- src: This is the folder in which all of the source code lives. Within the root of this folder, the main class and spatial input.
    - devices: This folder contains the source code for all supported devices

It should be noted that if code does not live within the source folder, Cargo will ignore it when it is compiled and run.

Deployment Diagram

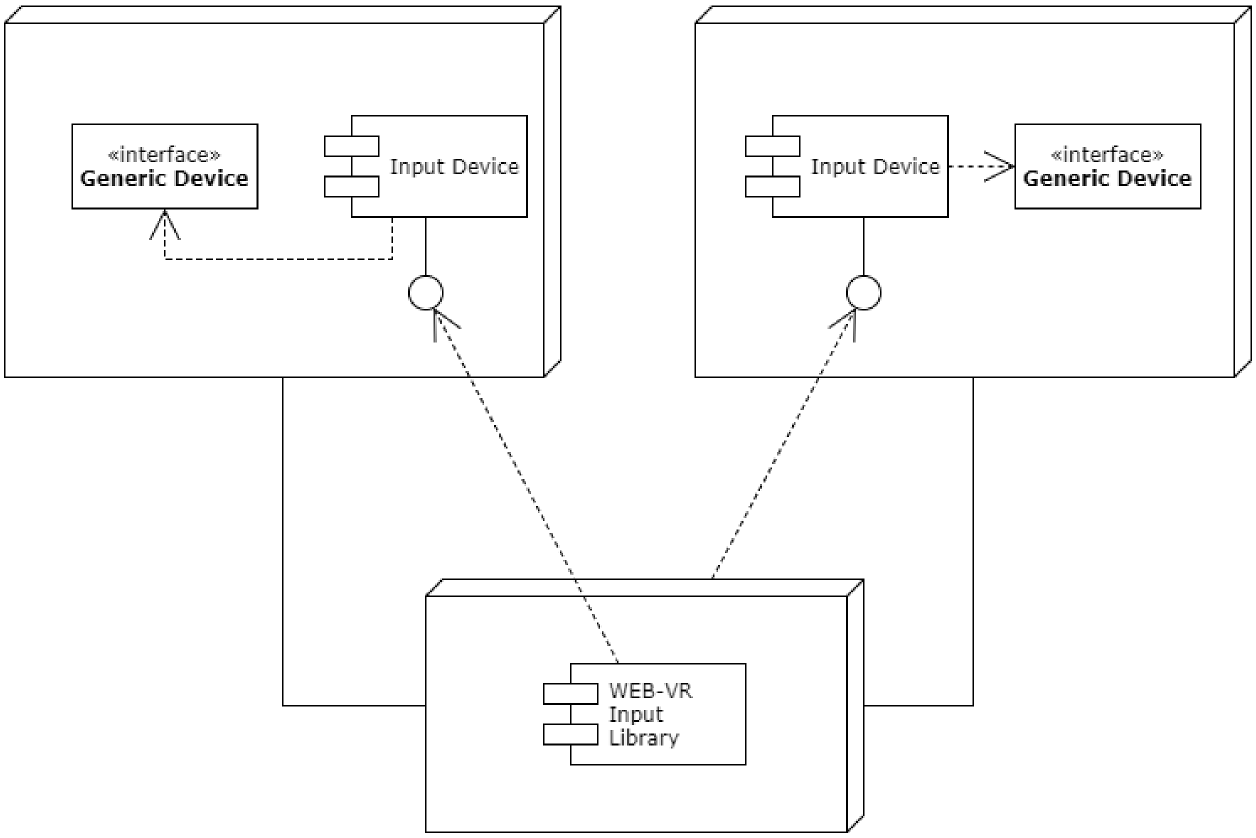


Figure 5: Deployment diagram for WEB-VR Input Library

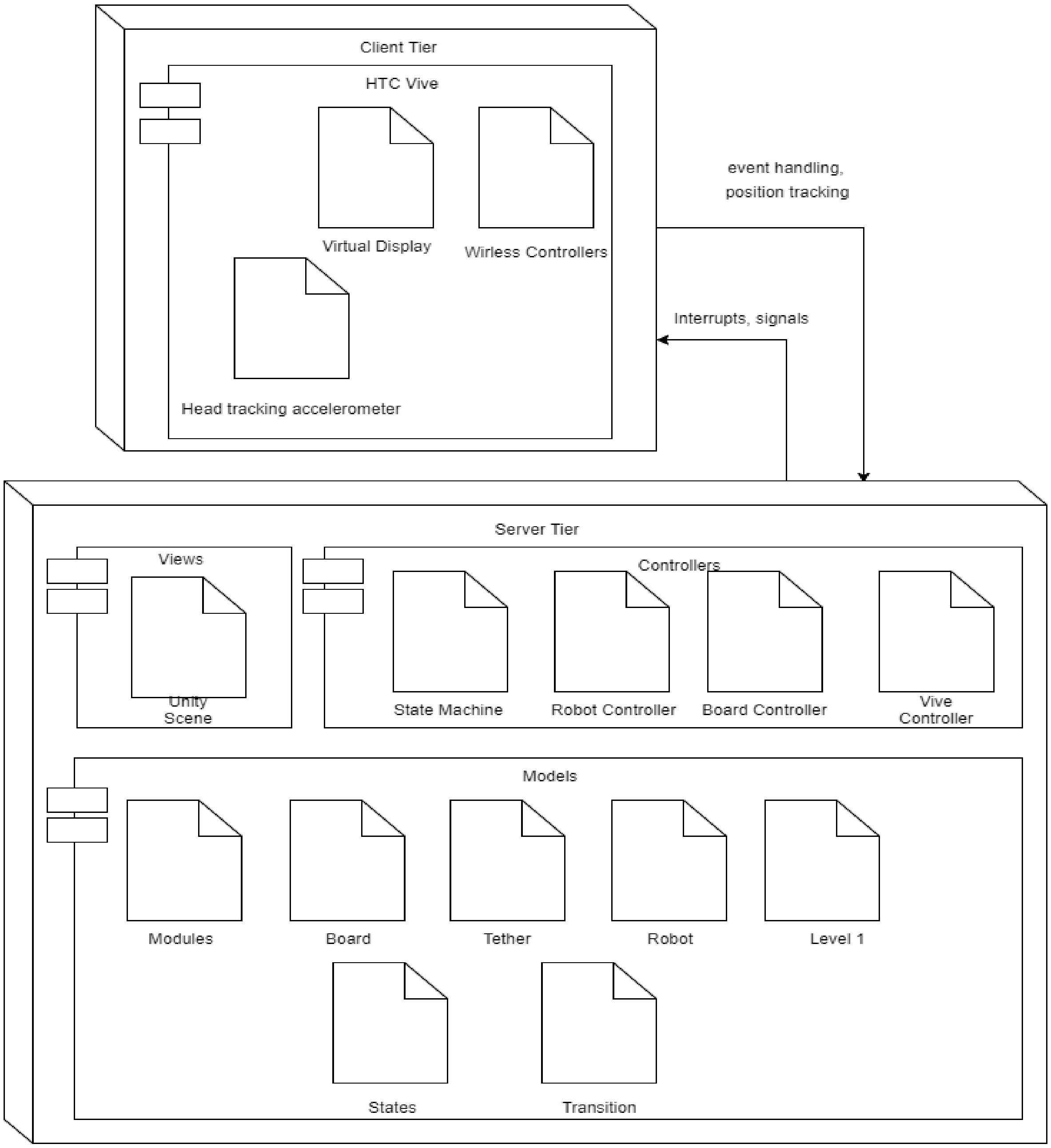


Figure 6: Deployment Diagram for WebVR Game

Design Patterns

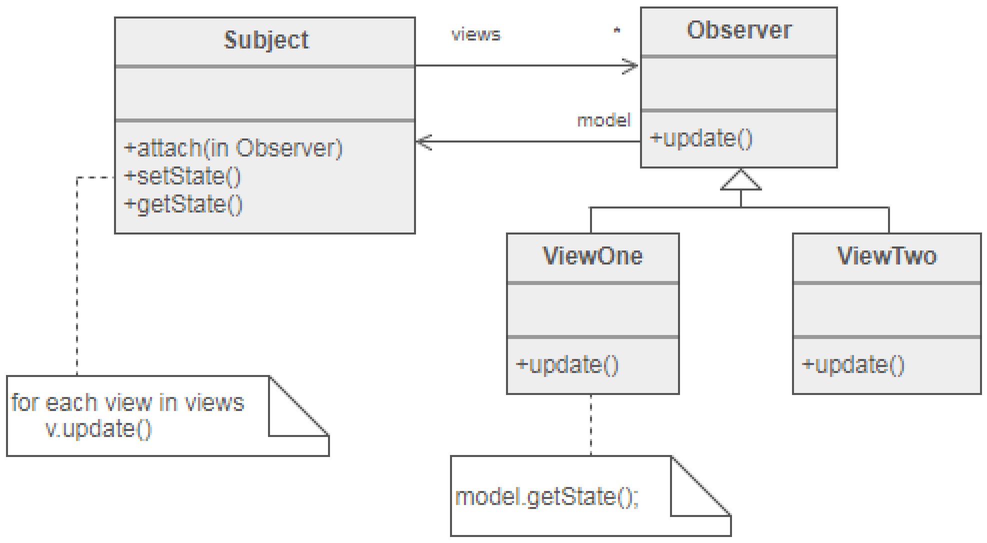


Figure 7: Observer Pattern Used for WEB-VR Input

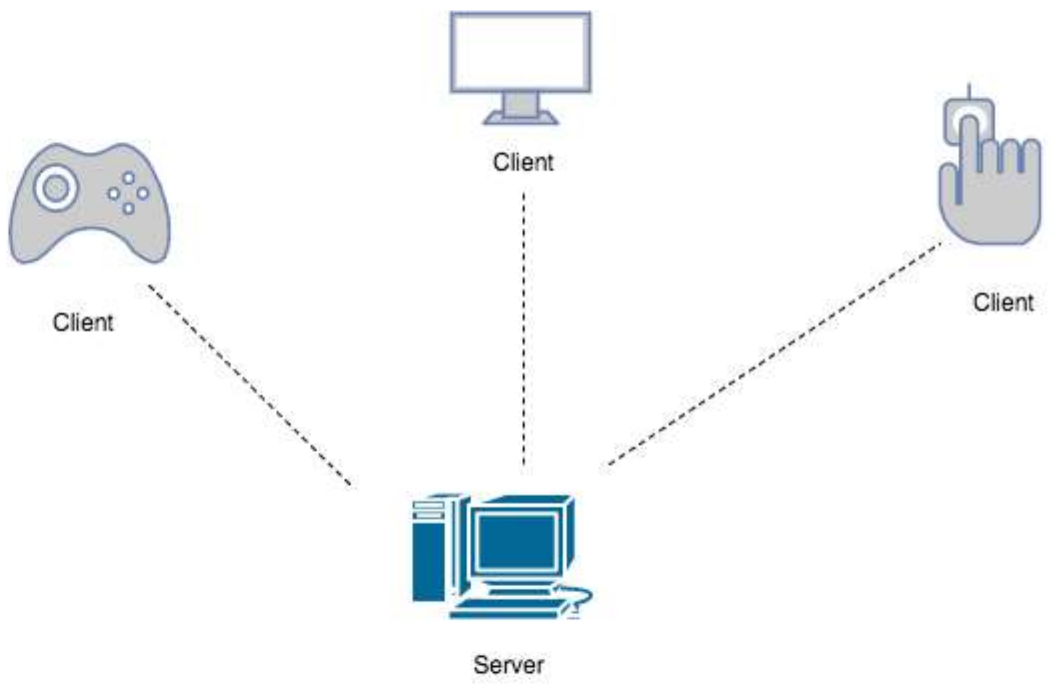


Figure 8: Client-Server pattern used for WEB-VR Input

SYSTEM VALIDATION

WebVR CS Education

Functional Tests:

Test Case ID:	WEBVR-EDU-Func-001
Description:	Verify that TextWrap component gets attached to a generic entity.
Pre-condition	The Draw dependency of TextWrap has been initialized and a TextWrap component has been attached to a generic entity. Draw has been set to “background: orange”
Expected Results:	Upon rendering the generic entity, it should be colored orange.
Actual Results:	The entity is colored orange.
Status:	Pass

Test Case ID:	WEBVR-EDU-Func-002

Description	Verify that a-animation component is moving the generic entity to a specific location.

Pre-condition:	Two a-animation components are attached to an entity with different start times, and positions.
	
	Position1 = “4 4 4” delay=”1000”
	Position2 = “5 5 5” delay = “2000”

Expected Results:	The entity is moved to position (4, 4, 4) from (0, 0, 0) after 1 second. The entity is moved to position (5, 5, 5) from (4, 4, 4) after two seconds
Actual Result;	The entity moved to the specific coordinates at the specified times.
Status:	Pass

Unit Tests:

Test Case ID	WEBVR-EDU-U001
Purpose:	To verify that the default font of Arial gets used.
Test Setup:	A new Text component object is added to an entity object. The string for the text field of the Text component is “Hello World”
Test Input:	addBehavior(text) is called on the entity object, which calls updateFont()
Expected Output:	“Hello World” gets rendered in arial.
Status:	Pass

Test Case ID:	WEBVR-EDU-U002

Purpose:	To verify that the a-tube component has been registered with A-Frame
Test Setup:	An entity object has been created. var newEntity = new Entity();
Test Input:	The function initComponent(a-tube) has been invoked on newEntity.
Expected Output:	initComponent() returns a Component object that reflects the behavior of a-tube default settings.
Status:	Pass

WebVR Game Team

Test Case ID:	WEBVR-GAME-U000
Purpose:	To test if state to state transition was correctly created

Test Setup:	Create dummy states:
	State start = new State(“start”);
	State state1 = new State(“moveForward”);
	State state2 = new State(“turnLeft”);
	State end = new State(“end”);
	
	Set start and end date:
	setInitial(start);
	setFinal(end);
	
	One transition between these declared states;
	Transition tran = new Transition(state1,state2);

Test Input:	The state machine’s next() function is called the output of this function will traverse the state machine taking the first transition found from this state until there are no more transitions to take. A toString() function is used to provide output of the transition.

Expected Output:	"currentState: (0000, start), currentTransition: (0000, start, moveForward)\n" "currentState:(0001, moveForward), currentTransition: (0001, moveForward, turnLeft)\n" "currentState(0002, turnLeft), currentTransition:(002, turnLeft, end)\n" "currentState(1111,end), currentTransition: null\n"
Status:	Pass

Test Case ID:	WEBVR-GAME-U001
Purpose:	To test the visual tether between two game modules
Test Setup:	This test has to be done in a separate unity project. 1.Create two cube objects 2. Create a cylinder object 3. Create an object tag named "module" 4. Assign the "module" tag to the newly created cubes 5. Import the Tether.cs file into the new unity project 6. Assign the Tether.cs script to the cylinder object
Test Input:	1. Add cube1 and cube2 to the public variables To and From of the Tether.cs script 2. Run the unity project
Expected Output:	The cylinder will now position itself between the two cubes and will resize itself in realtime as you move the cubes in any dimension.
Status:	Pass

WebVR Input Device Team

Test Case ID	WEBVR-INPUT-IN001-Mouse-Sunny
Description	The developer will launch the demo class and record a left click, a right click, and a center click on different points of the screen.
Pre-Condition	The main class in the rust library is running. A mouse must be connected to the machine
Expected Results	When the mouse is connected, a left click will print “Left button down.” A right click will print “Right button down”. A center click will print “Middle button down”. The output of these will be recorded on a terminal.
Actual Result	The terminal read “Left button down” on a left click event. It read “Right button down” on a right click event. It read “Middle button down” on a center click event. The output was recorded on a terminal thus proving the events were being handled as expected
Status	Pass

TestCaseID	WEBVR-INPUT-IN002-Controller-Sunny
Description	The developer will launch the demo class and only provide Inputs for left analog and the ‘A’ button
Pre-Condition	The main class in the rust library is running. A controller must be connected to the machine and is fully operational
Expected Results	Once binary is running, only the left analog and ‘A’ button will provide output to developer.

Actual Results	The developer only sees the result of ‘Button A down’ and ‘LeftY moved’
Status	Pass

TestCaseID	WEBVR-INPUT-IN003-GenericDevice-Sunny
Description	The demo window is launched. The device recorded is an unrecognized device as it is not part of the rust device developer library. The user can interact with the device. The developer can process the generic inputs
Pre-Condition	Main rust class is running the demo window. The device connected to the machine is not supported
Expected Results	Terminal reads “Button down” when an interaction is placed on the empty device. When released, terminal should read “button up”
Actual Results	When any button was pressed, the terminal displayed “Button Down.” When any button was released, the terminal displayed “Button Up”
Status	Pass

GLOSSARY

- WEBVR Education Team - The group that primarily worked on using virtual reality frameworks to teach Computer Science
- WEBVR Gaming Team - The group that primarily worked on using virtual reality for gaming
- WEBVR Input Team - The group that primarily worked on using Rust to create a library for input devices
- Rust - A relatively new programming language used for systems development. This language relies on concurrency and memory safety.
- Cargo - A package manager used to build and run Rust programs. It also handles the installation of dependencies.
- AFrame - A WebVR framework that specializes in creating immersive experiences in Virtual Reality
- Unity - A game development platform specializing in Virtual Reality and Augmented Reality.
- HTC Vive - A virtual reality device made by HTC

APPENDIX

Appendix A - UML Diagrams

1. WebVR CS Education

A. System Use Case

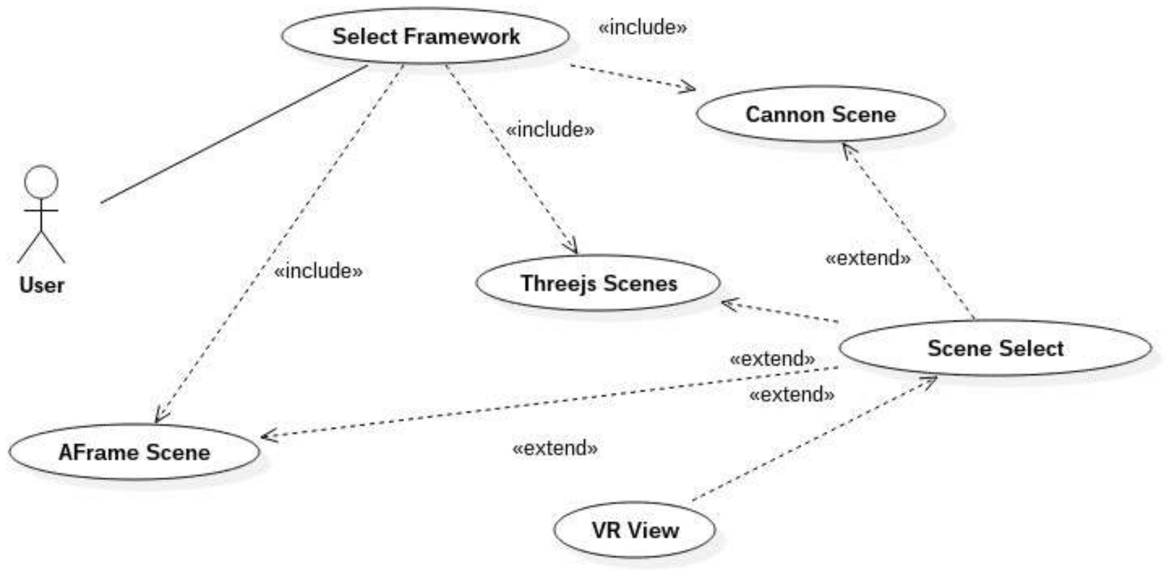
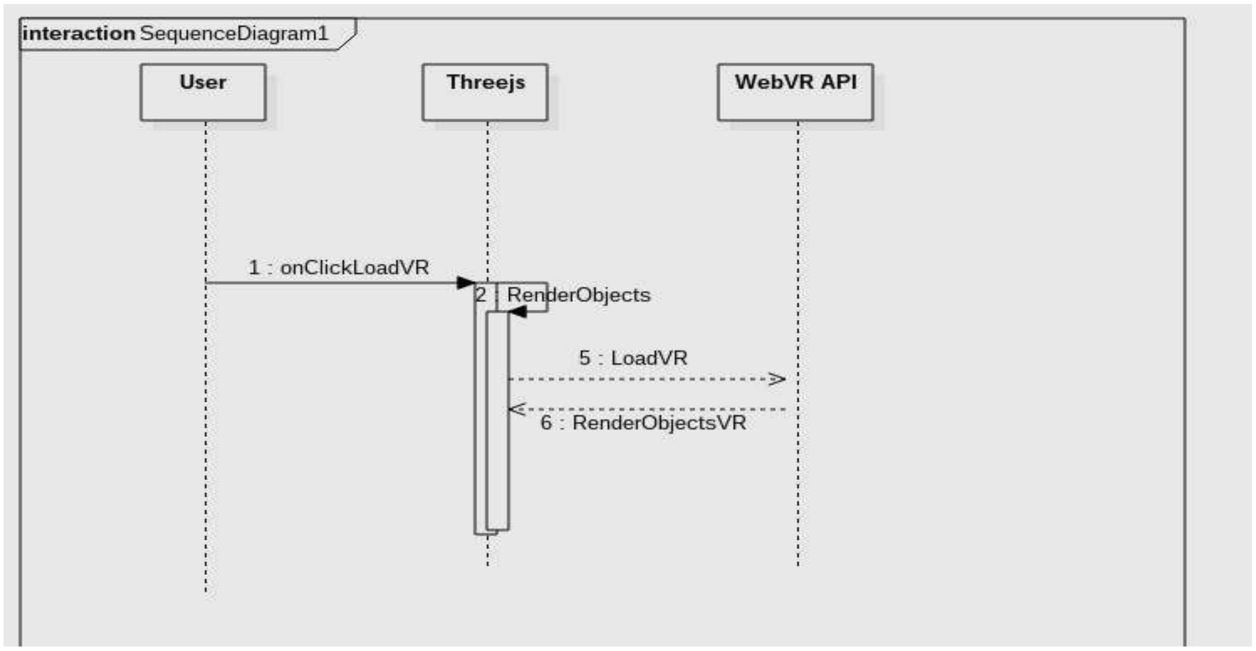


Figure 9: System Use Case for WEBVR Education Team



B. **WebVR Integration Sequence Diagram**
Figure 10: Integration Sequence Diagram for WEBVR Education

C. WebVR-Scenes Use Case Diagram

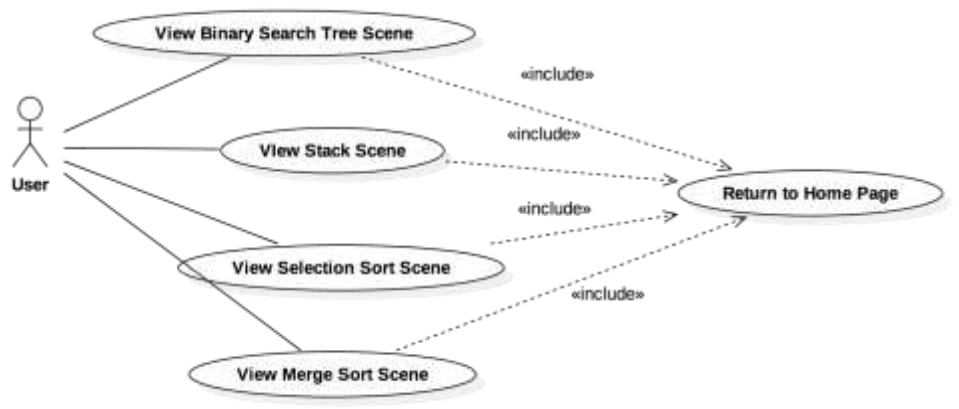


Figure 11: WEBVR Education Scenes Use Case Diagram

D. Web-VR System Class

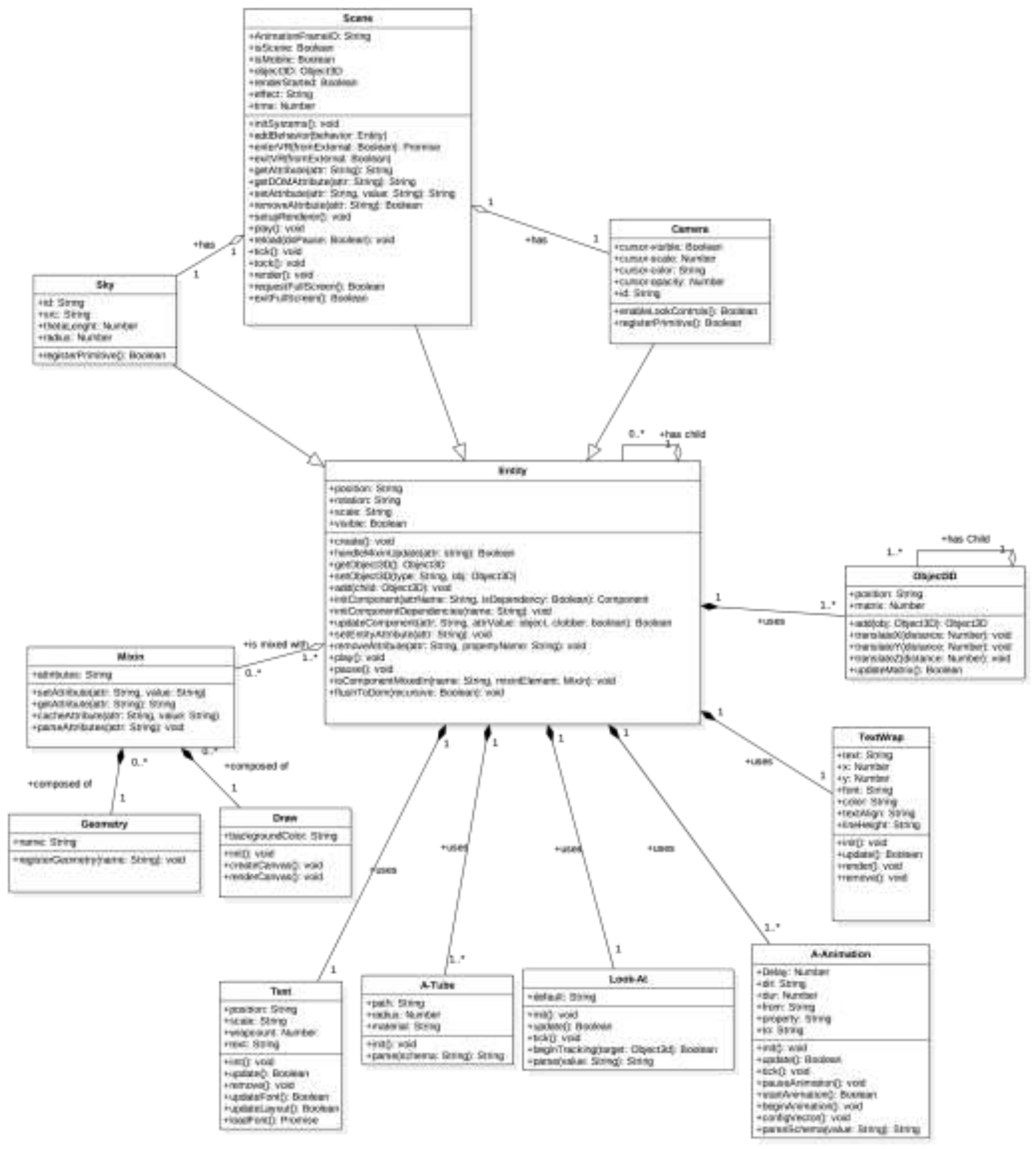


Figure 12: Class Diagram for WEBVR Education Team. Includes A-Frame

2. WebVR Gaming

A. Using a module use case diagram

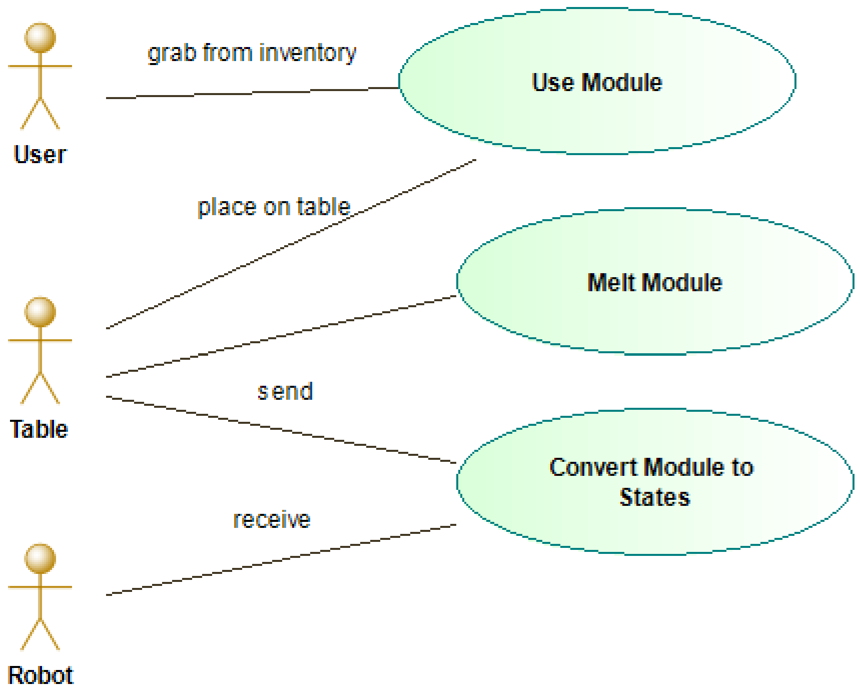


Figure 13: Use case diagram for WEBVR Gaming Team. Displays how signals are sent and received per object in game

B. Detecting a module Sequence Diagram

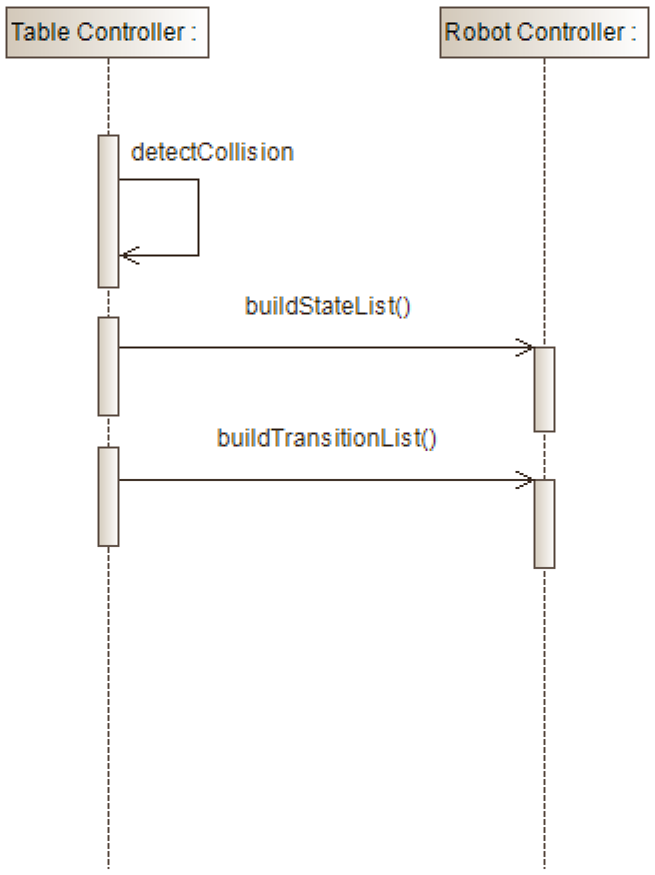
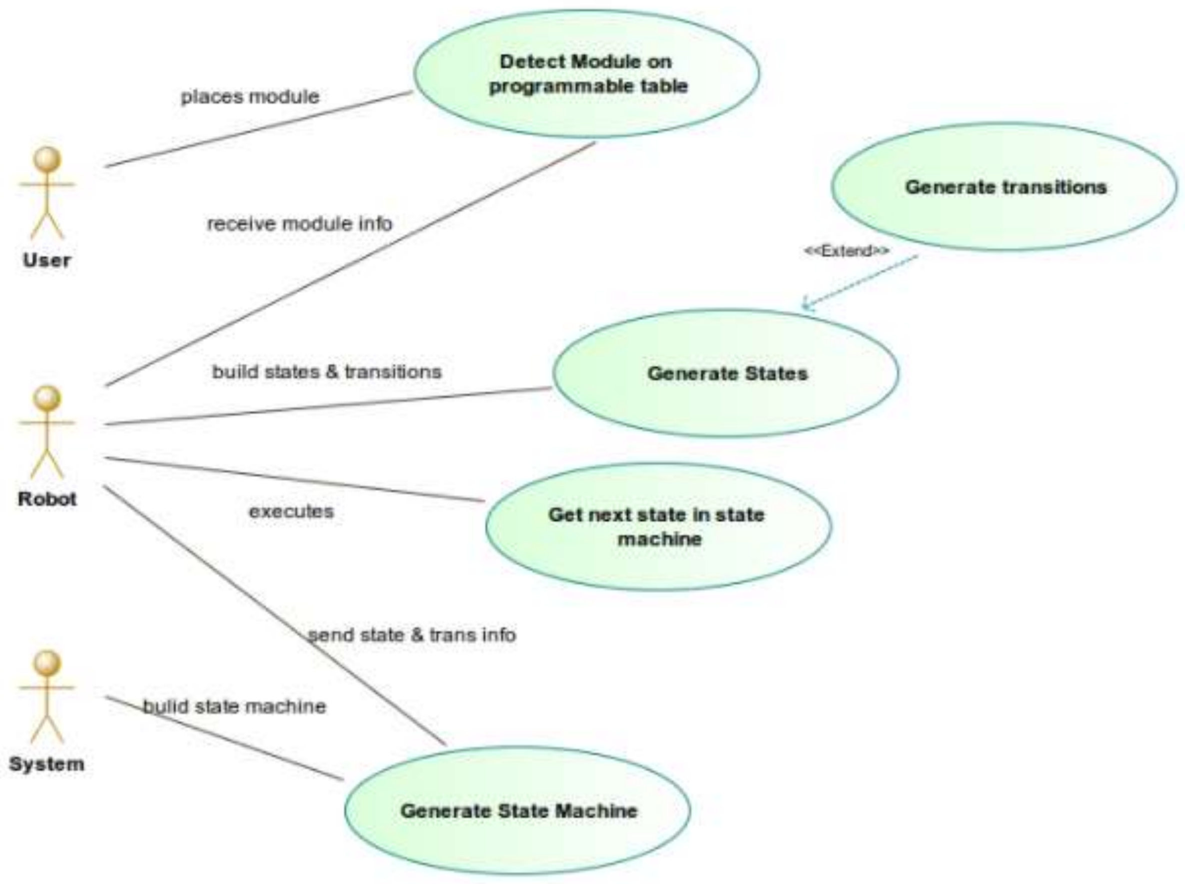


Figure 14: Use Case Diagram for Module Detection for WEBVR Gaming Team



C. Issuing a robot command use case diagram

Figure 15: Robot command use case diagram.

D. Building state machine sequence diagram

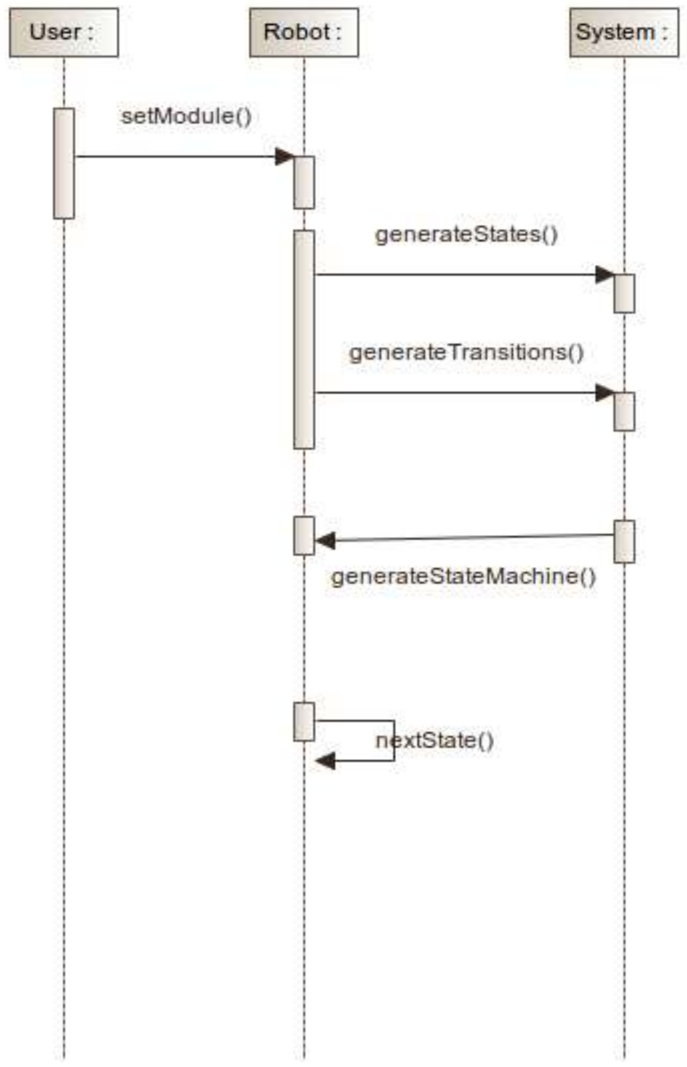


Figure 16: Sequence Diagram for constructing state machine

E. Robot Controller UML Diagram

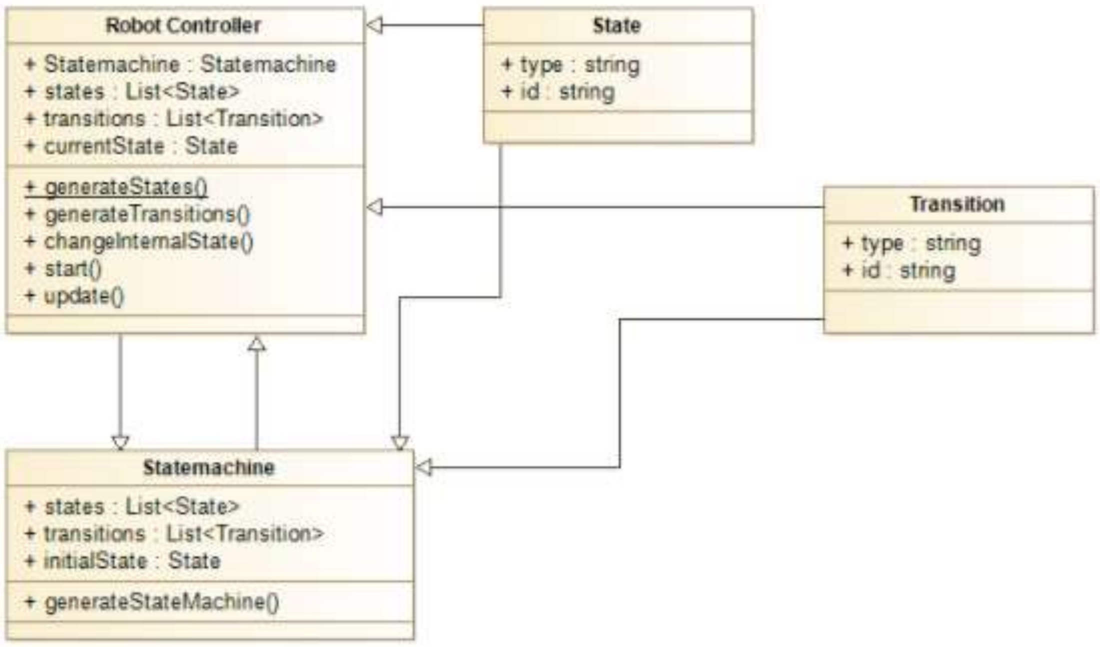


Figure 17: Robot Controller class diagram

F. Board Controller UML Diagram

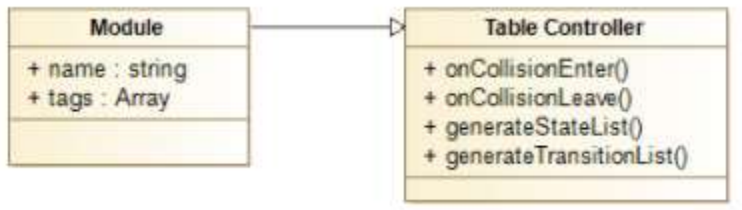


Figure 18: Board Controller class diagram

3. WebVR Input

A. Use-Case Diagram Tracking Life of Mouse connection tracking.

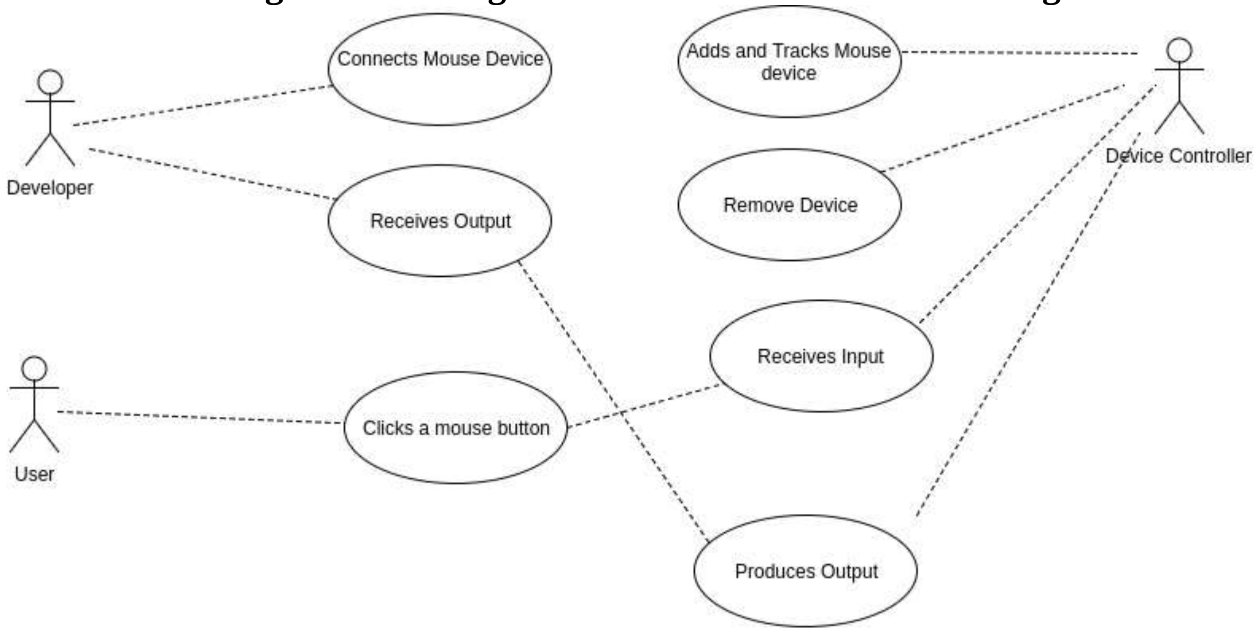


Figure 19: Mouse use case diagram for WEBVR Input

B. Sequence Diagram for Input Device Tracking

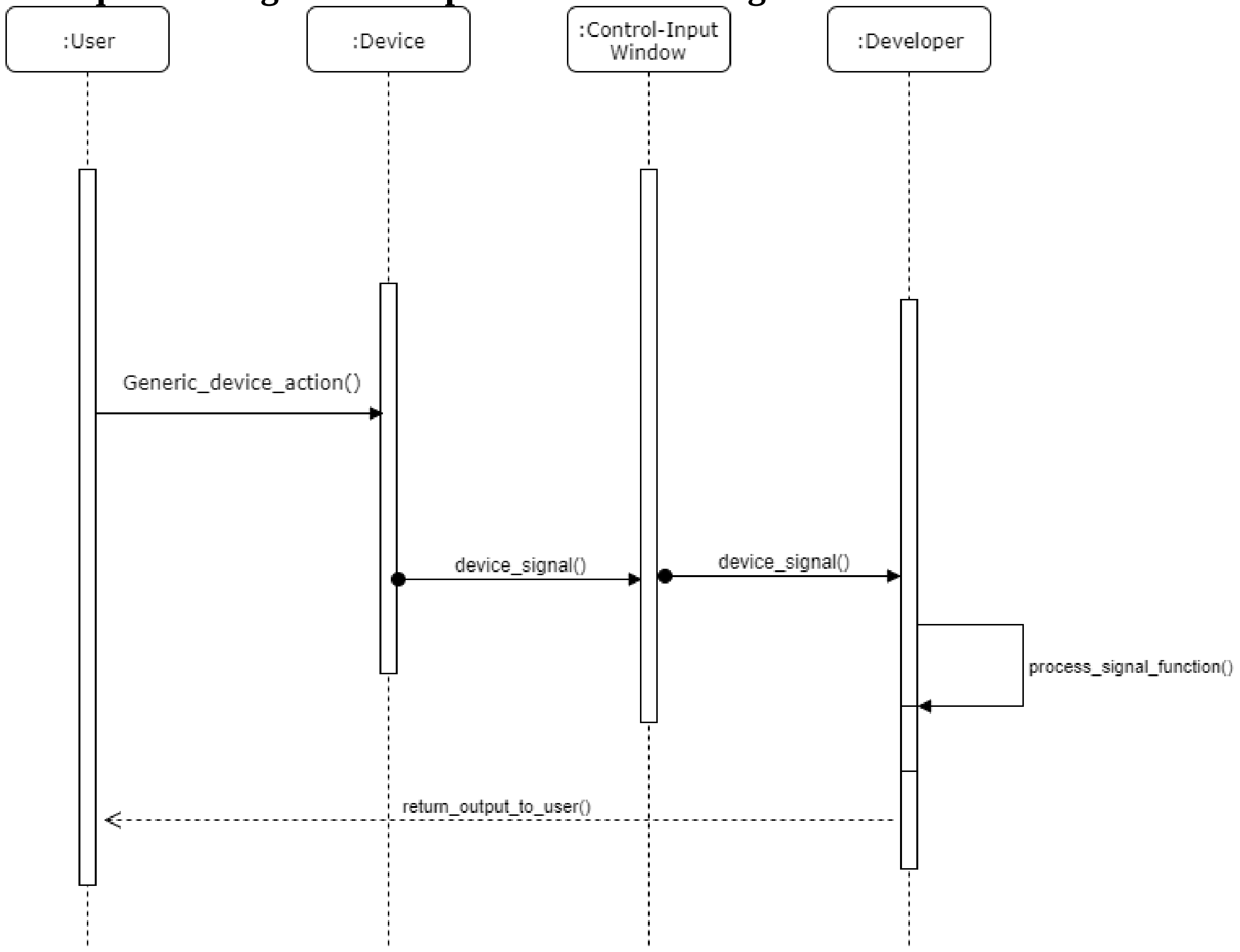


Figure 20: Sequence diagram for input device signal processing

C. System Class Diagram for Input Library

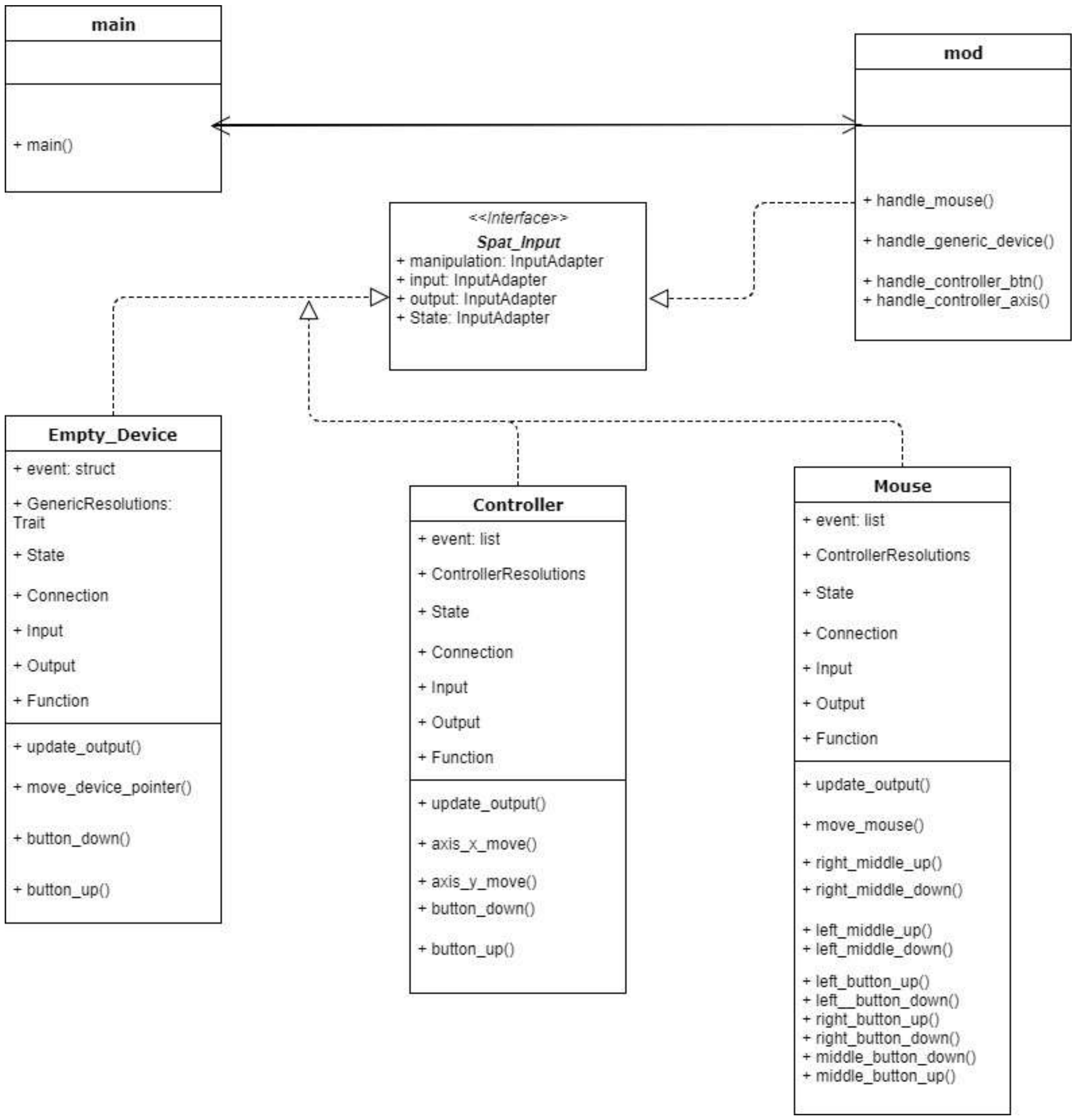
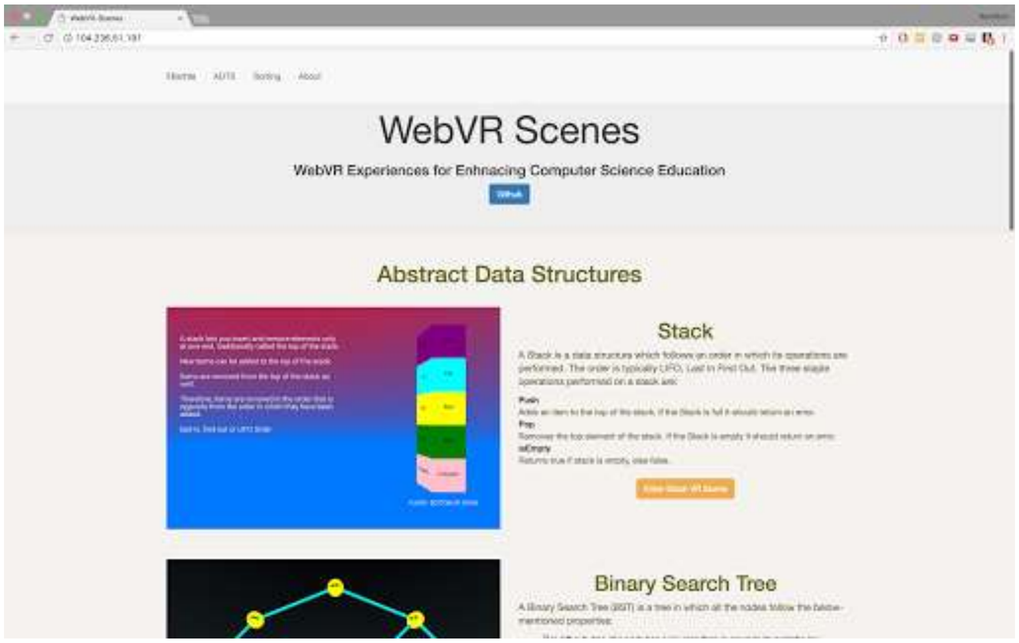


Figure 21: System Class Diagram for Input Device Library

Appendix B - User Interface Design

I. WebVR CS Education



WebVR Scenes

WebVR Experiences for Enhancing Computer Science Education

Abstract Data Structures

Stack

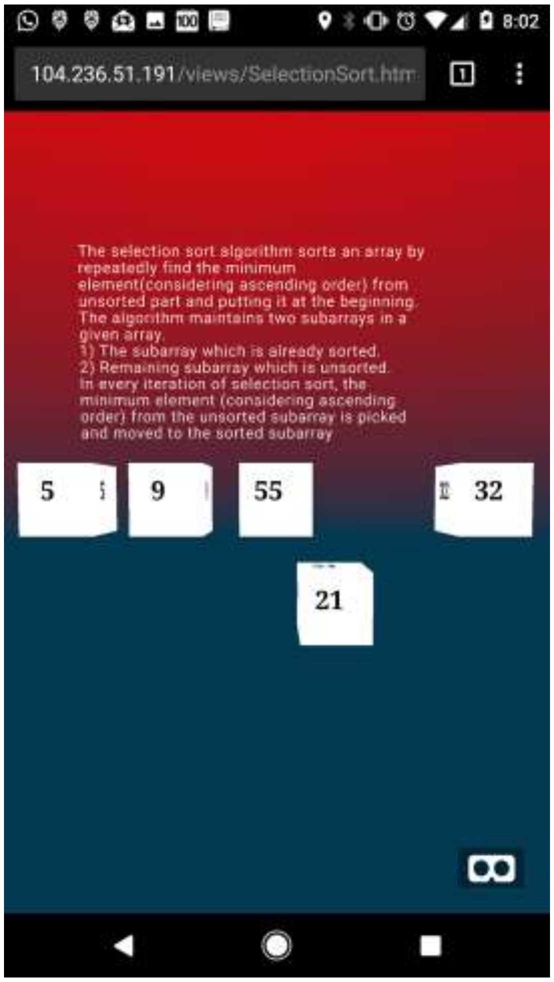
A Stack is a data structure which follows an order in which its operations are performed. The order is typically LIFO, Last In First Out. The three staple operations performed on a stack are:

Push
Adds an item to the top of the stack. If the Stack is full it should return an error.
Pop
Removes the top element of the stack. If the Stack is empty it should return an error.
isEmpty
Returns true if stack is empty, else false.

Binary Search Tree

A Binary Search Tree (BST) is a tree in which all the nodes follow the below-mentioned properties:

Figure 22: WebVR Scenes - Education Home Page



The selection sort algorithm sorts an array by repeatedly find the minimum element(considering ascending order) from unsorted part and putting it at the beginning. The algorithm maintains two subarrays in a given array.
1) The subarray which is already sorted.
2) Remaining subarray which is unsorted.
In every iteration of selection sort, the minimum element (considering ascending order) from the unsorted subarray is picked and moved to the sorted subarray

Figure 23: WebVR Scenes : Education - Selection Sort Scene

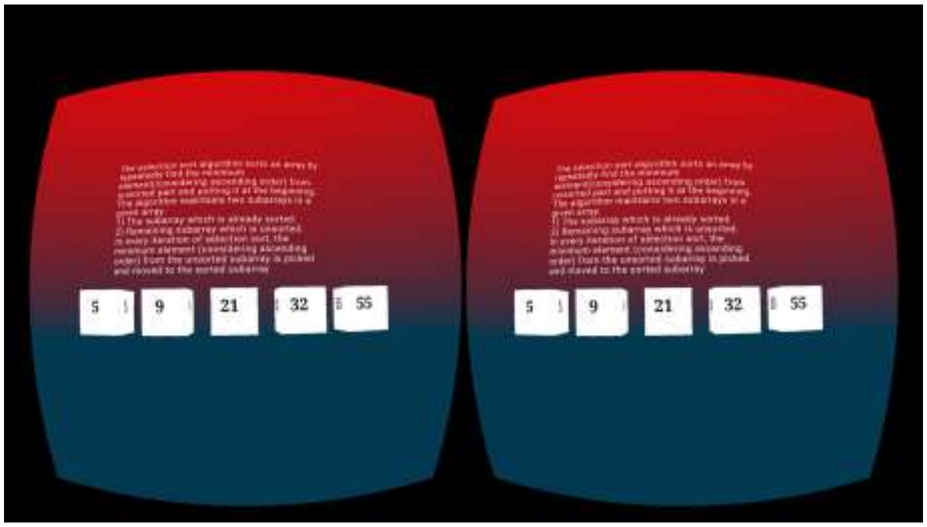

The selection sort algorithm sorts an array by repeatedly find the minimum element(considering ascending order) from unsorted part and putting it at the beginning. The algorithm maintains two subarrays in a given array.
1) The subarray which is already sorted.
2) Remaining subarray which is unsorted.
In every iteration of selection sort, the minimum element (considering ascending order) from the unsorted subarray is picked and moved to the sorted subarray.

Figure 23: WebVR Scenes : Education - Selection Sort Scene - VR Mode 1

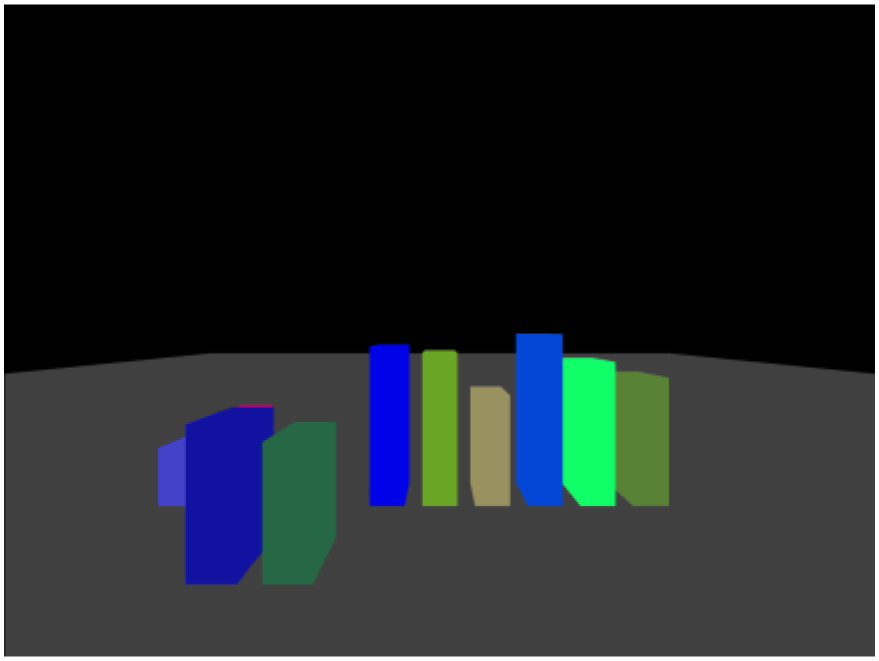


Figure 23: WebVR Scenes : Education - Selection Sort Scene - VR Mode 2

II. WebVR Gaming

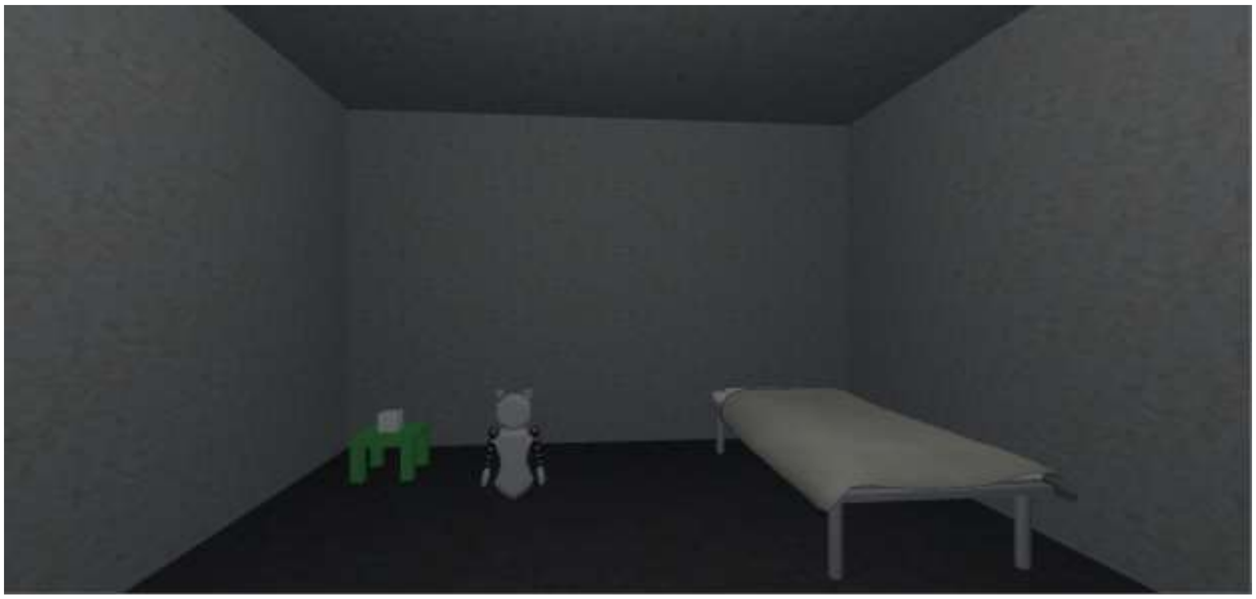
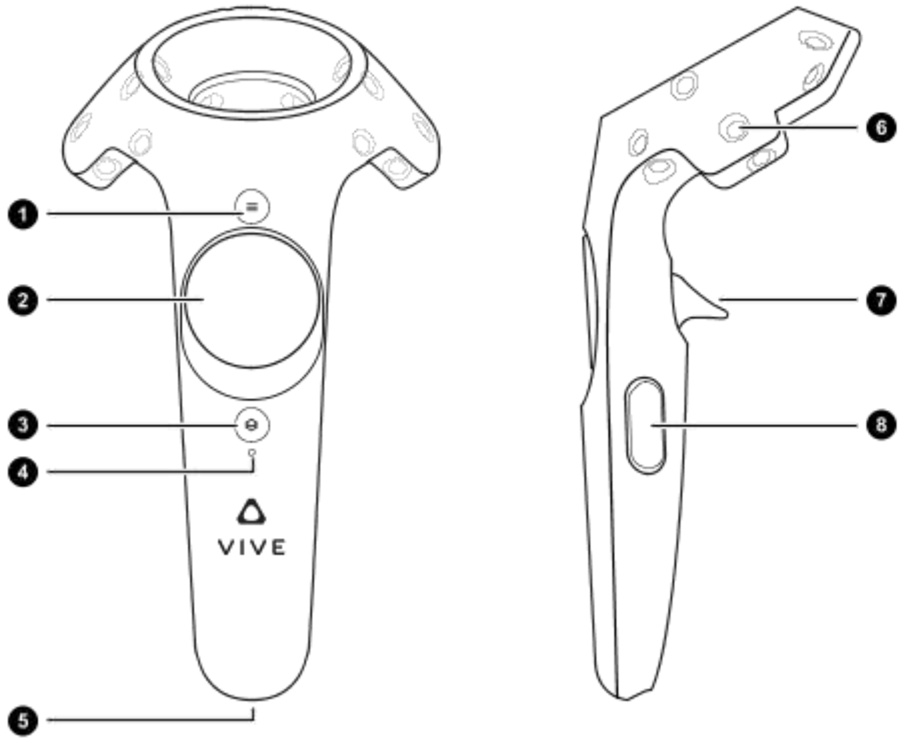


Figure 24: No HUD Game experience



1. Toggle button-layout tooltip
2. Cartesian Grid (x,y) (Pressing a unique region will have different functionality whenever the tooltip layout is visible)
    a. (0,n) - Toggle Robot Table
    b. (n,0) - Run SM configuration
    c. (0,-n) - Compile SM configuration
    d. (-n,0) - No input

   Note: These are only for the right controller. The left controller does not have any input mapping
3. HTC Vive pause menu
4. Battery Light indicator
5. Charging port
6. IR tracking receiver
7. Teleport Movement controls
8. Grab objects in the environment

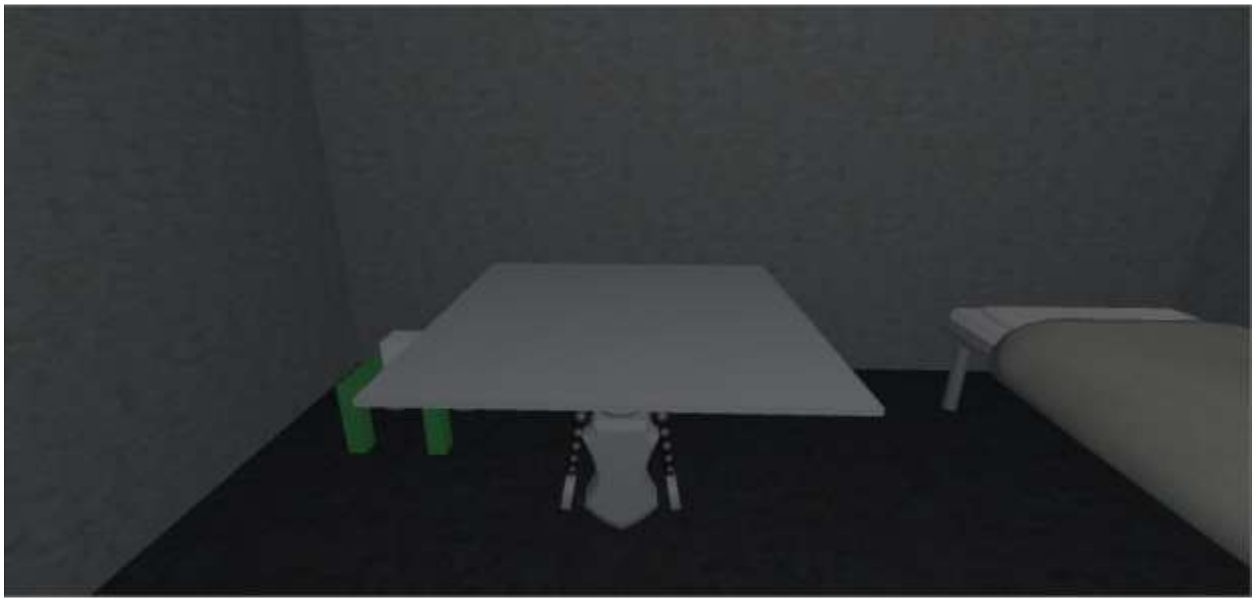


Figure 25: Board toggled on

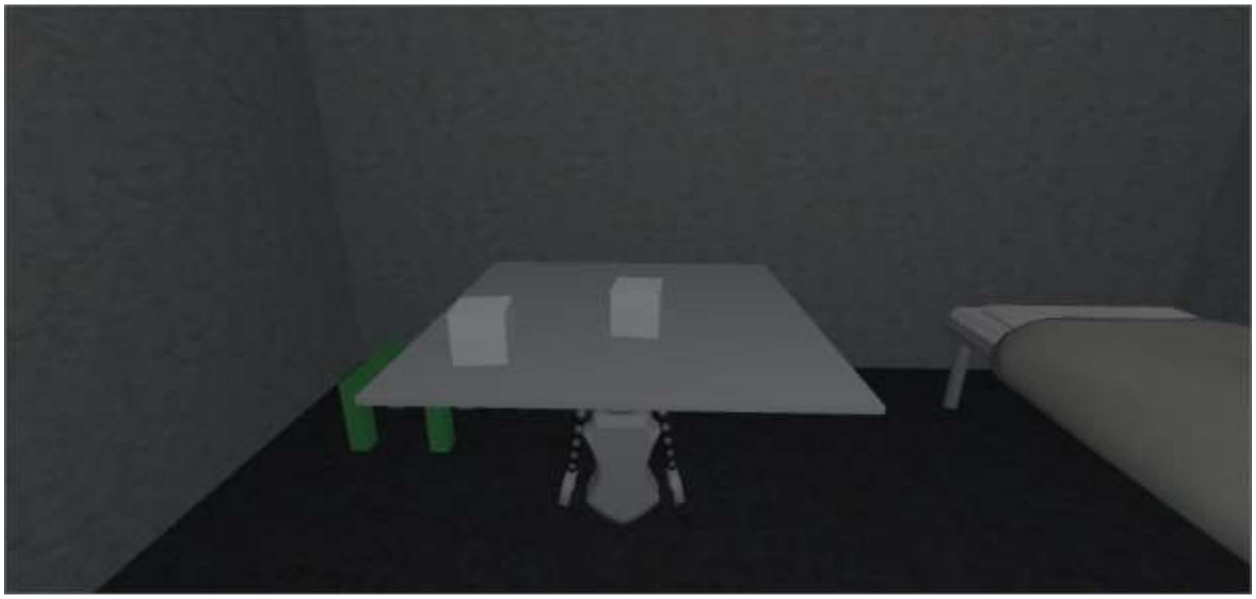


Figure 26: Two modules placed onto the open board

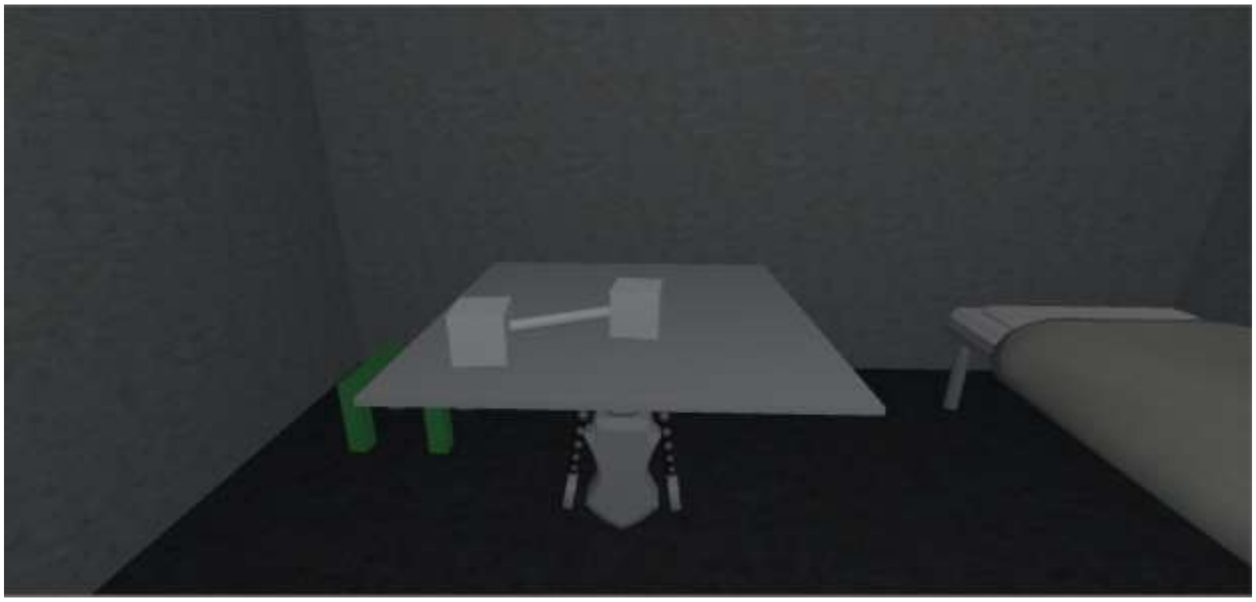


Figure 27: Two modules connected together to form a state to state transition.

Appendix C - Sprint Review Reports

Sprint Review Meeting Minutes: May 26th, 2017

Attendees: Hamilton Chevez, Bernardo Pla, Daniel Khawand, Daniel Rivero, Pachev Joseph, Francisco Ortega (Product Owner)
Start time: 19:25
End time: 19:30

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

- As this was the first sprint, most of the user stories revolved around setting up and doing research. There were no user stories that warranted a demo

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.

- None

Sprint Review Meeting Minutes: June 9th, 2017

Attendees: Hamilton Chevez, Bernardo Pla, Daniel Khawand, Daniel Rivero, Pachev Joseph, Francisco Ortega (Product Owner)
Start time: 17:30
End time: 18:40

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

- Daniel Rivero's demo was accepted by the product owner.
- Hamilton Chevez and Daniel Khawand presented a demo
    - Daniel Khawand presented a scene in which elements were added by a click event.
    - Hamilton Chevez presented graphical scenes of stack and sort data structures.
    - Both were accepted by the product owner.
- Pachev Joseph and Bernardo Pla presented a demo of the rust library for the mouse
    - This was accepted by the product owner

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.

- None

Sprint Review Meeting Minutes: June 23rd, 2017

Attendees: Daniel Khawand, Pachev Joseph, Daniel Rivero, Bernardo Pla,
Francisco Ortega (Product Owner)
Start time: 5:45 PM

End time: 6:06 PM

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

- All of the demos were accepted by the product owner.
    - Daniel Rivero presented the grabbing of objects and movement in a Unity WebVR environment. He also showed the animation of the robot to be programmed.
    - Pachev Joseph demonstrated remote-controller input as designated by a developer.
    - Bernardo Pla presented the generic input device implementation.
    - Daniel Khawand demonstrated the dynamic scene changes for varying data structure visualizations in WebGL to be later implemented for WebVR.

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.

- None

Note:
Hamilton Chevez was unable to attend the Sprint Review Meeting due to a medical condition. Hamilton informed Dr. Ortega of his condition, and he was granted a leave of absence for the meeting.

Sprint Review Meeting Minutes: July 8th, 2017

Attendees: Daniel Khawand, Pachev Joseph, Daniel Rivero, Bernardo Pla, Hamilton Chevez
Francisco Ortega (Product Owner)
Start time: 4:10 PM
End time: 5:00 PM

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

- All of the demos were accepted by the product owner.

- Hamilton & Daniel Khawand shared with Francisco their web link for displaying data structures in VR
- Pachev and Bernardo shared with Francisco their web link for generalizing input devices through rust
- Daniel Rivero shared with Francisco a demo showcasing the finalized level and the linking of modules as transitions.

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.
None

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

1. **WebVR Education**

   Installation

   1. Install NPM on your system
      a. Link: https://www.npmjs.com/get-npm
   2. Clone our Github repository locally.
      a. Link: https://github.com/FIU-SCIS-Senior-Projects/WEB-VR-Towards-Virtual-and-Augmented-Reality-for-the-WEB-1.0/tree/master
   3. From the command prompt/terminal, navigate to the root directory of the github repository. Change directory to /webvr-scenes/.
   4. Install all project dependencies by executing “npm run install”.
   5. To run the project locally, execute the command “npm run serv” and it will launch the project on a local server that can be accessed via a browser on localhost:8000

   Wishlist

   1. Add dynamic content to the scenes.
   2. Human interaction via touch controls.
   3. Finish Algorithms and ADTs User Story scenes.

2. **WebVR Gaming**

   Installation

   6. Download the latest version of Blender
      a. Link: https://www.blender.org/download/
   7. Download the latest version of Unity
      a. Link: https://store.unity.com/download?ref=personal
      b. Note: Downloading Unity gives you the option to install Microsoft Visual Studios (Check this option if not previously installed)
   8. Download latest project revision from GitHub
   9. Run Unity and open the project in it the respective project directory

   Wishlist

   1. More levels
   2. Integrate Conditionals statements

a. Conditionals can be implemented by adding guards to the state machine

3. **WEBVR Input Device Team:**

Installation:

1. Navigate to the Rust Website: https://www.rust-lang.org/en-US/
2. Download the latest version of Rust (at the time of this document, the latest stable version is 1.18.0)
3. Run the installer file.
   a. A command prompt appears allowing you to customize your installation. You can use the system defaults for your installation
4. Rust will let you know when the installation completes and you can press any key to close the installer
   a. You can verify the installation by running “rustc --version” and if it returns the version, you have installed Rust
   b. You can verify the installation for Cargo by running “cargo --version” and if it returns a value, you have installed Cargo

Running the library as an executable:

1. Download the library
2. Navigate to the library directory using a command prompt
3. Run the command “cargo run”
4. Interact with the library by interacting with the pop-up window

Wishlist:

1. Integrate library with web applications
2. Add more devices to support
3. Find ways to convert library to Dynamic Link Library or Shared Object
4. Publish library to Cargo repository to be used by Rust community

REFERENCES

A-Frame Github Repository:
https://github.com/aframevr/aframe
A-Frame Home Site
https://aframe.io/
Rust Programming Site:
https://www.rust-lang.org/en-US/