DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
THE UNIVERSITY OF TEXAS AT ARLINGTON

ARCHITECTURE DESIGN SPECIFICATION
CSE 4316: SENIOR DESIGN I
FALL 2019

oVRworked
Cookums

KASI CROSS
GEETESH KALAKOTI
JOHN LIVESAY
QUINTON TOMPKINS
KEVIN TUNG

Revision History

Revision	Date	Author(s)	Description
0.1	12.01.2019	QT	document creation
1.0	12.07.2019	KC, GK, JL, QT, KT	initial draft

CONTENTS

List of Figures

List of Tables

1 INTRODUCTION

Cookums is a virtual reality food truck simulation game built with Unity. The game will consist of 3 main features. The ability to cook and interact with food, the ability to receive orders from customers, and the ability to deliver the orders to customers and receive a score based on performance and correctness.

Unity games are built with a component based structure. Rather than outline how Unity organizes classes, this document will outline the overall gameplay system's major components and interactions with the engine, along with OpenVR and the VR kit's role in the overlying system architecture.

Gameplay will have an emphasis on using your hands to gather ingredients, prepare them into complete foods, and then serving them in a fast paced, exciting environment.

2 System Overview

The system is comprised of three distinct layers: the hardware layer, the API layer, and the engine layer.

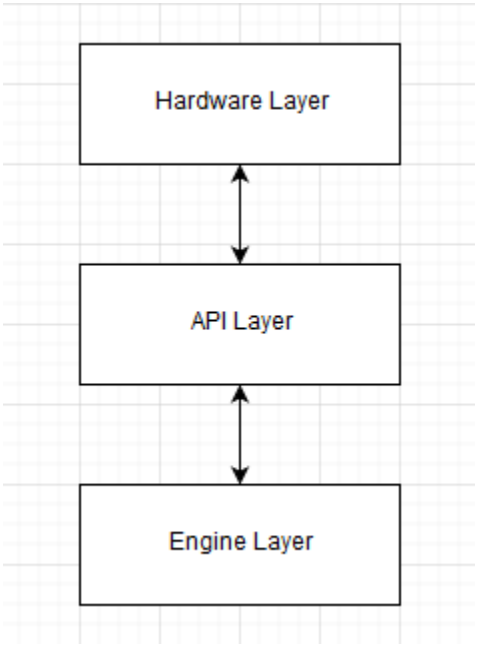

Figure 1: A simple architectural layer diagram

2.1 Hardware Layer

The hardware layer includes the virtual reality kit that serves as the one and only source of input and output for the user, making it critical to the project. At minimum, the kit is composed of a left and right hand controller and a head mounted display (HMD). The kit will supply the rest of the system with information about the user's keypresses along with hand and head transforms relative to the play area. In return, the hands may receive haptic feedback and the HMD will have continuous imagery provided by the system. No business logic is present in this layer; it is entirely an I/O peripheral.

2.2 API Layer

The API layer consists of the OpenVR system's API and runtime. It is responsible for serving as an intermediary of data between the hardware layer and the engine layer, bridging the gap between hardware and software. This layer provides support to a multitude of virtual reality kits and abstracts each individual kit's I/O into a unified interface that works for all.

2.3 Engine Layer

The engine layer includes the entirety of gameplay elements, and handles several critical components. Unity handles lighting, rendering, physics and collision detection. The remaining subsystems handle other gameplay elements unique to Cookums. Unity coordinates interaction between the player controller, entities, and gameplay direction through ordering, difficulty, and customer AI agents. The entities are grouped collectively and labelled as the entity director subsystem, and the SteamVR Interaction System works to create a seamless connection for the player controller to work with other systems.

3 Subsystem Definitions & Data Flow

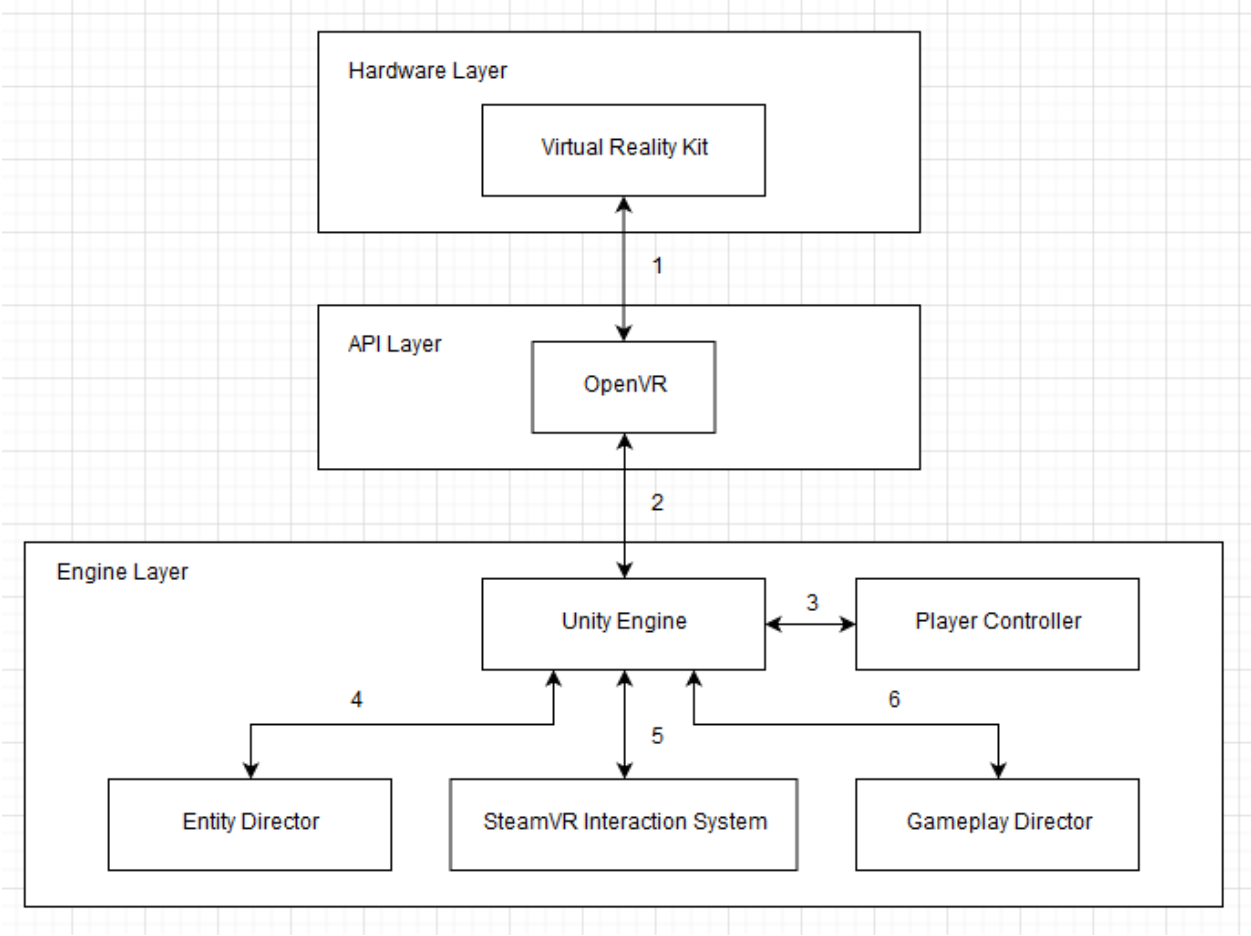

Figure 2: A simple data flow diagram

4 Hardware Layer Subsystems

The Hardware Layer contains the Virtual Reality Kit. This layer will manage the player's interactions and observations of the system, communicating to the API layer about the player's actions and receiving input for the player to see and feel.

4.1 Virtual Reality Kit

The Virtual Reality Kit communicates with the player via visual and physical stimulus. It will send and recieve data to the player. It will send data about the player's movement that is collected via the headset and the controllers. It will receive data for the player to see through the headset and feel through vibrations in the controllers.

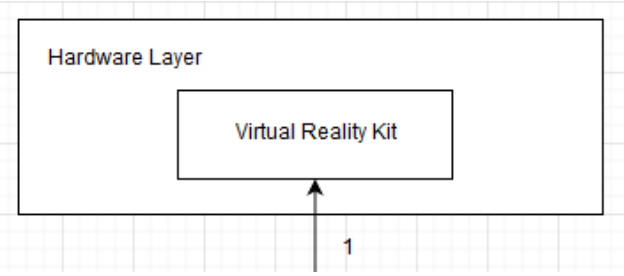

Figure 3: The Virtual Reality Kit Subsystem.

4.1.1 Assumptions

The player wil be able to see the game through the headset and feel vibrations from the game through the controllers. The movement of headset and controller when in range of sensors should be detected and should send movement information correctly to the API layer.

4.1.2 Responsibilities

The Virtual Reality Kit will include positional sensors, a headset, and two controllers. The sensors will know the location and direction of the player's hands and head based off the hardware location and position. The headset will display visual information to the player. The controllers will vibrate when told to by lower layers. The controllers also have buttons for the player to use to capture and issue events to the API layer.

4.1.3 Virtual Reality Kit Interfaces

Table 2: Virtual Reality Kit Interfaces

ID	Description	Inputs	Outputs
#1	OpenVR Connection	Visual feedback Haptic feedback	User position User inputs

5 API Layer Subsystems

5.1 OpenVR

The API layer consists of just the OpenVR subsytem. The OpenVR subsytem communicates with the virtual reality kit subsystem and the Unity Engine subsystem, bridging the gap between hardware and software in a unified interface for all virtual reality kits. This was chosen in order to reduce the amount of work necessary for the project.

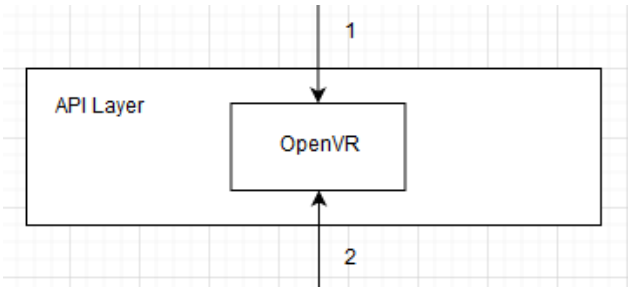

Figure 4: The OpenVR Subsystem.

5.1.1 Assumptions

The Virtual Reality Kit must be connected either by cable or WiFi adapter to allow data flow. The user has the OpenVR runtime installed.

5.1.2 Responsibilities

OpenVR's main role is to relay information from the Hardware Layer to the Engine Layer. The player's head movement translations, and left-hand/right-hand movement translations from the Virtual Reality Kit is recorded and passed down to the API Layer, where OpenVR then translates that data to the Engine Layer, enabling the Engine Layer to make the same translations inside Unity.

5.1.3 OpenVR Interfaces

Table 3: OpenVR interfaces

ID	Description	Inputs	Outputs
#1	Hardware layer dataflow with data specific to VR kit	User head position User hand position Button presses	Visual feedback Haptic feedback
#2	Engine layer dataflow with data abstracted to engine	Abstracted visual feedback Abstracted haptic feedback	Abstracted head, hand position Abstracted button presses

6 Engine Layer Subsystems

6.1 Unity Engine

The Unity Engine subsystem facillitates communication and coordinates the other components of the game. It also handles physics, objects, lighting, rendering, collision, and other basic modern game engine capabilities. As a note, SRVIS is shorthand for the SteamVR Interaction System.

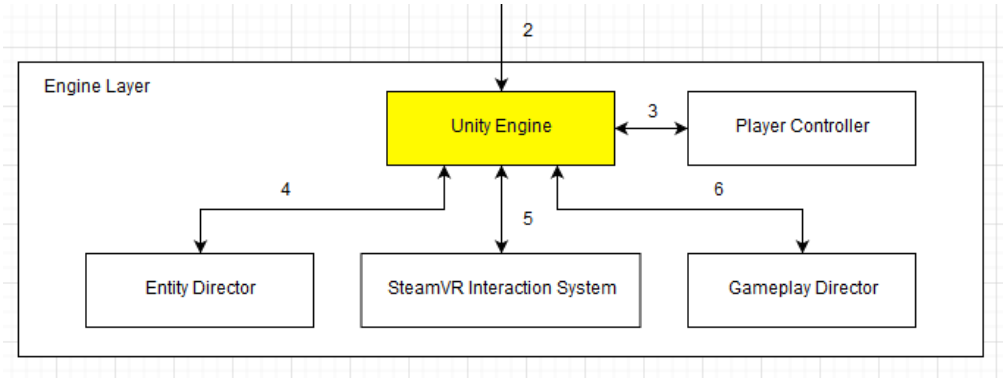

Figure 5: The Unity Engine Subsystem.

6.1.1 Assumptions

The user's operating system supports the executables exportable by Unity.

6.1.2 Responsibilities

The Unity Engine controls every other subsystem for gameplay, keeping track of delta time between frames, the gameobject node hierarchy, and the actual application process.

The primary purpose includes rendering each frame, and calling the necessary functions of each subsystem to update the game state and visual appearance. The other subsystem functions can be triggered because of a frame update, a threshold being reached (such as the passage of time), an entity entering a collision area with some form of trigger, or because some button press was made by the user. From there, the function within the subsystem will update the game state and may necessiate a different subsystem's function, which gets Unity to fire off the the function.

6.1.3 Unity Engine Interfaces

Table 4: Unity Engine interfaces

ID	Description	Inputs	Outputs
#2	User I/O to upper layers	OpenVR input bindings OpenVR positional data	Rendered frames Haptic feedback data
#3	Player orientation data in world	Info for SVRIS User game orientation data	User real orientation data
#4	Updating of world entites	Entity properties	Delta time elapsed Manipulation by SVRIS
#5	VR Interaction data	Interactable properties Visual highlight flags VR Physics approximation	Collision data User inputs User orientation
#6	Order, level, and AI direction	Order statuses Customer statuses	Entity submission events Delta time elapsed

6.2 Player Controller

The player controller subsystem is responsible for the player's position and orientation in the world space, as well as movement.

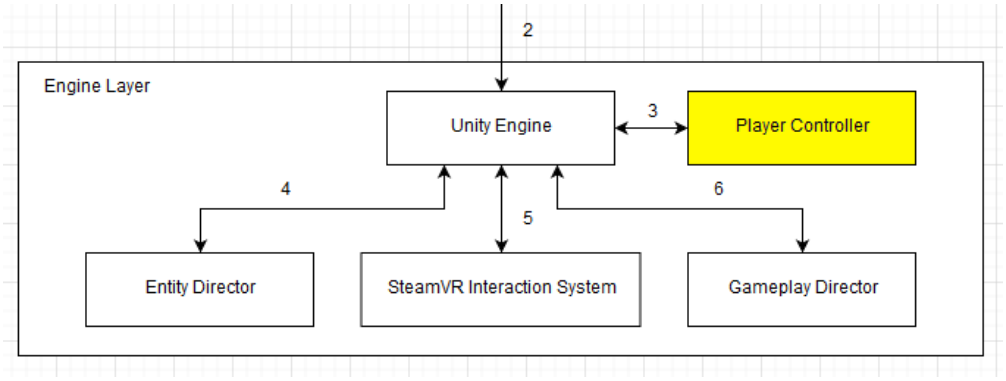

Figure 6: The Player Controller Subsystem.

6.2.1 Assumptions

Takes in information that the Unity Engine obtains from the API layer that is correctly calibrated.

6.2.2 Responsibilities

The Player Controller is responsible for syncing the player's movement and interaction with objects with the state of the objects within the Unity Engine. This includes where the player is in the world, what the player's hands are interacting with, and making sure that the player isn't interacting with objects in ways they are not supposed to such as clipping through walls.

6.2.3 Player Controller Interfaces

Table 5: Player controller interfaces

ID	Description	Inputs	Outputs
#3	Player orientation data in world	User real orientation data	Info for SVRIS User game orientation

6.3 Entity Director

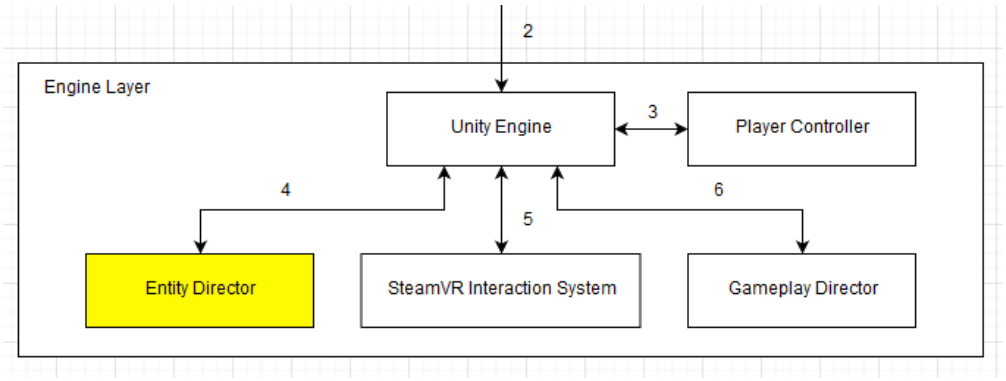

Figure 7: The Entity Director Subsystem.

6.3.1 Assumptions

There is enough processing power to handle entities in the game world seamlessly.

6.3.2 Responsibilities

The Entity Director keeps track of every Unity gameobject that is instantiated (either by default or player interaction) and it's data values. For food objects, it must keep track of the food score of that object, if that object is being cooked, and if that object is able to snap onto another object. Furthermore, it must keep track whether or not the player is currently interacting with an object, or if object is interacting with another object via collider boxes. Then, it will call necessary functions demanded by the entities.

6.3.3 Entity Director Interfaces

Table 6: Entity director interfaces

ID	Description	Inputs	Outputs
#4	Updating of world entites	Delta time elapsed Manipulation by SVRIS	Entity properties after update

6.4 SteamVR Interaction System

The SteamVR Interaction System (SRVIS) handles much of the virtual reality based interactions. Without it, the game would still work in a 3D physical sense, but this system allows for our hands to become manipulators of the VR world.

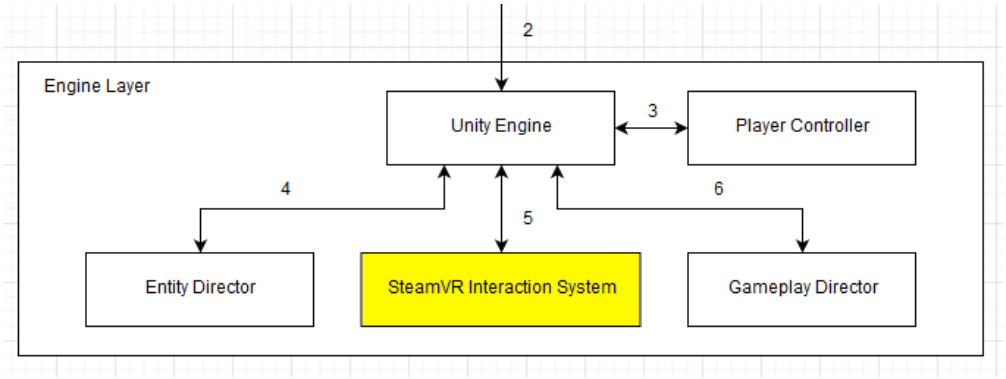

Figure 8: The SteamVR Interaction Subsystem.

6.4.1 Assumptions

The information communicated by the Unity Engine about the player's orientation is correct.

6.4.2 Responsibilities

The SVRIS handles item highlighting, interaction through being picked up, pressed, manipulated, or stacked by the player's input. It tells the Unity engine how to process items, whether to render a highlight indicating something can be picked up, and whether something may be interacted with by a player controller. Collisions with hands and interactable objects prompts a highlight render trigger, which is passed up to the engine to take care of for the object. Hands that are accompanined with user trigger pulling will hold the objects, passing up physics to move the object along with the player's hands.

6.4.3 SVRIS INTERFACES

Table 7: SVRIS interfaces

ID	Description	Inputs	Outputs
#5	VR Interaction data	Collision data User inputs User orientation	Interactable properties Visual highlight flags VR Physics approximation

6.5 GAMEPLAY DIRECTOR

The gameplay director handles actual gameplay objective elements, not necessarily the entire element of gameplay.

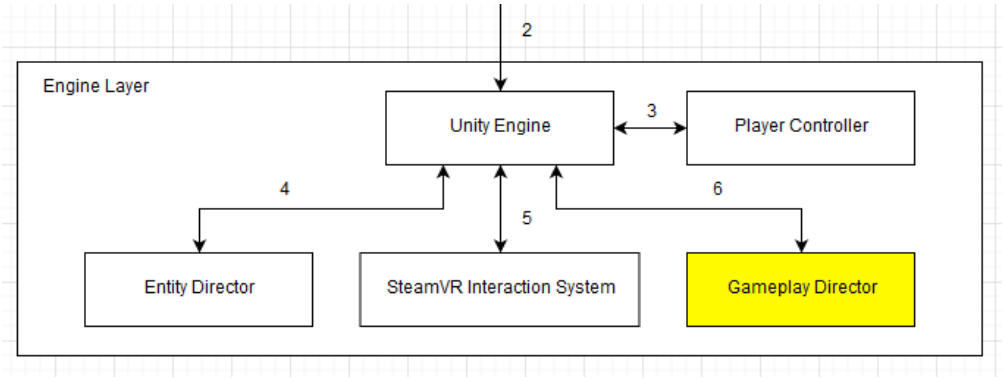

Figure 9: The Gameplay Director Subsystem.

6.5.1 ASSUMPTIONS

The entity director can communicate order submissions to the engine correctly for the gameplay director to process.

6.5.2 RESPONSIBILITIES

The Gameplay Director is responsible for pushing the player through the cycle of receiving the order, showing the player what the order is, giving feedback to the player on how pleased the AI is with their service, and keeping track of making sure what the player is doing gets back to the Unity Engine to update what the engine is sending out for the player to do. It generates new orders based on elapsed time and previous completion of orders, and will update the status of orders and customers to the engine to process.

6.5.3 GAMEPLAY DIRECTOR INTERFACES

Table 8: Gameplay director interfaces

ID	Description	Inputs	Outputs
#6	Order, level, and AI direction	Entity submission events Delta time elapsed	Order statuses Customer statuses

References