Application of Virtual Reality in Experimental Economics

Aaska Shah
Kerala Brendon
Nolan Slade
Vyome Kishore

April 2019

Contents

1 Objective

The project aims to conduct controlled laboratory experiments in order to gain insight into how people make decisions when physically impaired in some way. As a result of the inherent, notably ethical, difficulties presented by actually impairing an experimental participant in the real world, a digital environment serves as a compromise. Our component of the project will be to develop a virtual reality-based environment to be used as a setting for such experiments. This advancement in the project will offer participants a more immersive environment than previously, where they completed the experiment using a two-dimensional computer monitor and game controller. Ideally, in a more immersive environment, researchers would observe more natural decision making from participants.

1.1 Experiment Overview

At a high level, the controlled experiments involve a singular participant repetitively completing a basic task and subsequently being given a reward, all within a virtual environment. The duration of the experiment is separated into discrete blocks of time called *days*. Each day, the participant may be faced with one or more impairments, and may be presented with the option to alleviate them using a *treatment*, available at a cost. Requirements for each of these components are now described.

1.2 Task

The basic task must be a simple, effort-based operation quantifiable over the lifetime of the experiment. As such, the 'effort' component must be clearly defined: what constitutes a participant exerting greater effort?

1.3 Reward

For every completed iteration, the participant should be rewarded based on their performance. Following the above, a participant exerting greater effort will finish the experiment with a higher amount of total compensation. The reward serves as motivation for the participant to keep repeating the task, and also as a means of establishing an opportunity cost to paying for and subsequently receiving a treatment.

1.4 Environment

The virtual reality environment should be developed for the HTC Vive. More specifically, the environment must fit within the HTC Vive-equipped Interview Room located in the McMaster Decision Science Laboratory. The dimensions of the interview room are approximately 10' long by 9' wide. In the ideal case,

the shape of the virtual world should closely match the interview room in order to maximize participant immersion.

Additionally, the virtual environment should include measures to reduce claustrophobia, as well as motion-induced nausea.

1.5 Days

The simulation should allow for a configurable temporal structure comprised of days, each with a desired length and parameter set. The days should be used to determine when the participant becomes impaired and when treatments become available, either free or paid. For a particular day, the participant should be aware of what day number it is, their current earnings, the time remaining in a day, whether they have become impaired and if treatment is available.

1.6 Impairments

The participant should become impaired on the days configured by the researcher. The intensity and type of the impairment will be configured by the researcher.

1.7 Treatment

Treatments should be offered on days determined by the researcher. The participant will be informed of the availability of treatment before the day begins. The treatment will decrease their impairment immediately by an amount determined by the researcher. The location at which to receive treatments should be clearly distinguishable and easily found. In addition, key information pertaining to the treatments must be communicated concisely; for example, the cost required to receive the treatment.

If the user chooses to pay for the treatment, their earned money should decrease by the price determined by the researcher and the treatment should be administered. Conversely, if the user chooses to wait for the treatment, they must not be able to complete the basic task for the duration configured by the researcher. At the end of the waiting period, the treatment should be administered.

1.8 Configuration File

The researcher should be able to use a configuration file that outlines the functionality of the simulations in terms of days, impairments, treatments, and any other simulation variables. This is a critical component; an application that is not highly customizable will not be useful for a long period of time in the McMaster Decision Science Laboratory. Our goal is to provide a simulation environment that can be used to perform many experiments over a long lifetime.

2 Our Implementation

Goal: Develop a configurable virtual reality environment as a setting for controlled laboratory investigations. Created for the HTC Vive virtual reality platform with the Unity Game Engine.

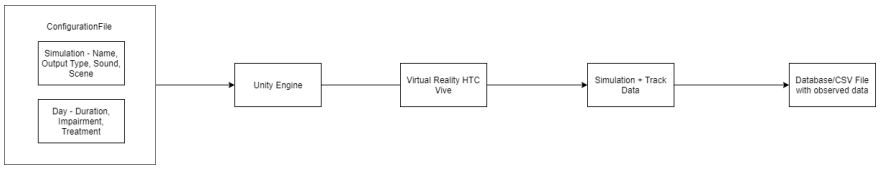

Figure 1: Input to Output Flowchart

2.1 People

Participant - The subject who is participating in the study by using the virtual reality simulation.
Researcher - The member of the McDSL who is conducting the experiments.

2.2 Environment

The virtual reality environment is developed using the Unity game engine for the HTC Vive. The environment must fit within the HTC Vive-equipped Interview Room located in the McDSL Lab. The measurements of the lab are 10' long and 9' wide. The measurements in the virtual world are multiplied by a programmer-specified scaling factor to achieve this.

By default, the virtual room has windows that show a view of a forest to reduce the potential for claustrophobia. These windows can be covered if the participant indicates that they are prone to experiencing motion sickness. Similarly, if the participant indicates they are both claustrophobic and sensitive to motion sickness, the windows are removed entirely and replaced with a door (such that the participant imagines that there is a way out of the room) and modern art.

2.3 Task

The implemented task is moving buckets of bouncy balls from a source pipe to a destination tub. A successful delivery of a ball from the source to destination earns the participant a number lab dollars configured by the researcher ($1.00 by default). The source pipe instantiates a bouncy ball every 0.1 seconds. The tub is used to collect the balls and calculate how many are delivered (as well as the subsequent payout to the participant).

The bucket can fit approximately 65 balls. It can be grabbed by the participant using the trigger buttons on the HTC Vive hand controller to simulate

gripping. Gravity is simulated to be slightly lighter than the real world and therefore the participant may spill balls from the bucket if it is tipped.

2.4 Days

The simulation is split into a set number of days, each with its own length and parameters. This temporal structure is configured by the researcher.

The days are used to determine when the participant becomes impaired and when treatments become available, either free or paid. For a particular day, the participant is aware of what day number it is, the time remaining in a day, whether they have become impaired and if treatment is available.

2.5 Lab Money

The participant earns lab money for every ball successfully delivered from the source to the sink. Treatments for impairments cost lab money, with an amount determined through the evaluation of a cost function. The cost function is configured by the researcher. The participant's goal is to maximize their earnings of lab money throughout the lifetime of the experiment. They are aware of how much money they have earned and the cost of treatment at the time it becomes available.

2.6 Impairments

The participant will become impaired on the days configured by the researcher. The intensity and type of the impairment is configured by the researcher. Multiple impairments can be utilized on a single day.

Physical Shake: Inhibits the participant's dexterity by implementing haptic feedback on the controllers. If impaired in this way, the participant's maximum carrying capacity will be decreased; that is, balls will simply fly out of the bucket if they try to carry too many. *Vision Impairment:* Inhibits the participant's sight by using fog. With this impairment active, the participant will find it more difficult to complete the task with accuracy.

2.7 Treatments

Treatments are offered on days determined by the researcher. The participant is informed of the availability of treatment during the day transition. The treatment will decrease their impairment immediately by an amount determined by the researcher. The treatments will be offered on a table marked by a red cross symbol.

When treatment is available, 2 pill bottles will be placed on the table. One will be indicated with the price of payed treatment and the other with the time to wait for free treatment (configured by the researcher). If the user picks up the pay bottle, then treatment is administered and their lab dollars decrease by the appropriate amount. If the participant picks up the wait bottle, then

the bucket is taken away from them and they are unable to complete the basic task for the communicated amount of wait time. After the waiting period has expired, the treatment is administered and their lab dollars remain the same.

The cost of the treatment is calculated using:

$$C = (c - bT + aT^2)$$

where each of the variables C, c, b, and a are determined by the researcher.

2.8 Configuration File

The researcher may modify the contents of a configuration file in order to define the temporal structure of a given experiment as well as auxiliary parameters. The full details of this file are described later in this document, in the User Manual.

```
Simulation
    Name:Sample_Config
    Output:.txt,database
    Description:Sample file.
    Sound:disabled        # If not specified, enabled by default
    Scene:forest          # If not specified, forest by default
Day
    Duration:5:00         # Minutes:Seconds
    Impairment
        Type:Visual/Fog   # Type/Subtype
        Factor:50%        # Percentage of Maximum
    Treatment
        Wait:15           # Seconds
        Cost
            C:100         # Starting cost
            a:default     # By default 1/Omega
            b:default     # By default day number
            c:default     # Omega (by default length of day)
Day
    Duration:5:00
    Impairment
        Type:Physical/Shake
        Factor:70%
    Impairment
        Type:Physical/Speed_Impairment
        Factor:50%
    Treatment
        Wait:10
        Certainty:80%     # If not specified, 100% by default
        Cost
            C:80
            a:0.05
            b:3
            c:120
```

Figure 2: Example Configuration File

2.9 Communicating to Participant

The participant is informed about the time remaining in a day, the day number and lab dollars earned through a display located above the source and destination.

Figure 3: The wall UI is duplicated on both sides of the room

The participant is informed that a day is being changed by floating text attached to the camera (i.e. it is always in their centre field of view). The participant is also informed if they are becoming impaired or if a treatment has become available in this manner. Instructions, present throughout different parts of the simulation, are also presented in such a fashion.

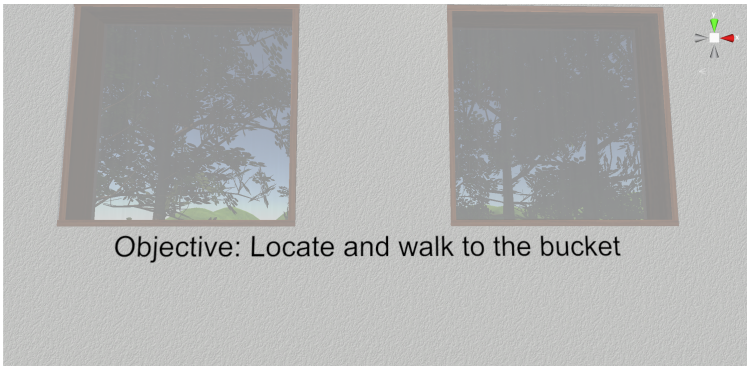

Figure 4: A sample instruction message

2.10 Output

The data from the simulation is persisted into a csv file. This includes the data collected in the previous experiment plus virtual reality-related positional data. Examples of persisted fields include headset position, controller positions, current day, current levels of impairment, and costs of available treatments, among others.

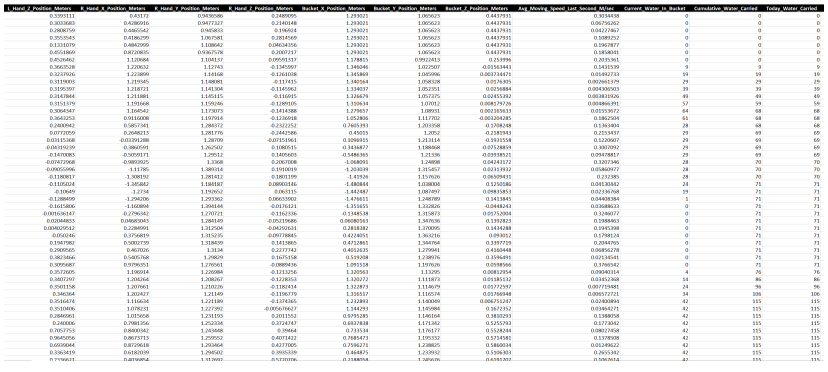

Figure 5: Some of the data collected in the csv file

3 Application Breakdown: Scene Components

3.1 VR Equipment Interface

SteamVR: SteamVR plugin developed by Valve Software, which allows the Unity engine to interface with the HTC Vive, or more generally, most popular VR equipment sets.

CameraRig: Works in conjunction with the aforementioned SteamVR component. This set of objects includes a headset tracker, which tracks position relative to the centre of the room in meters (*Camera*), as well as controller trackers that work in a similar fashion (*Controller (left/right)*). The CameraRig object is translated and rotated according to the transform provided by the *Camera* object, such that the participant is able to walk naturally across the scene.

Virtual_Hand_Left/Right: Since the *Controller (left/right)* object transforms are required in order to know the position and rotational values of the Vive's physical controllers, they themselves cannot be translated within the scene like the CameraRig. In order to render the hands in the correct location within the scene, these two game objects are translated instead. These two hands are the ones that the participant sees during the lifetime of the simulation.

3.2 Core Task Functionality

Container_Base: The container game object represents the bucket the participant uses to carry bouncy balls from one side of the room to the other. It contains 5 rectangular prism objects that make up the walls of the bucket: (*Container_Wall_Front/Back/Right/Left*). It also contains an invisible collider object, *Water_Droplet_Counter*, which is used to keep track of how many balls are currently inside of the bucket.

SimManager: Arguably the most important game object in the application - this object manages the state of the simulation, and keeps track of critical variables throughout its lifetime. Many other game objects reference the Simulation Manager in order to carry out their respective tasks.

PillManager: Manages treatment functionality including, but not limited to rendering visible components, and determining current pay and wait-time costs.

FlowManager: Responsible for starting and stopping the instantiation of bouncy balls from the source pipe. This component is required as there are certain times when instantiation, *flow*, should not be enabled. These include when the participant is standing too far away, or when they are transitioning between days.

FlowLimiter: An invisible cube object with a collider to determine if the headset is close enough to the source for the flow to be activated. The reasoning behind this object's inclusion is to discourage cheating, where a participant would try to reach the source pipe without actually walking to it.

DestinationLimiter: Similar to the FlowLimiter, except instead of preventing flow from being activated, it prevents the participant from being payed unless they are standing close enough to the destination tub.

DayZeroSpeedCounter: Currently not in use, but functional: measures the participant's average speed during day 0 tutorial. It is designed to provide the most accurate measurement possible, in that it only keeps track of speed and elapsed time when the participant is actually walking in between the source and destination.

TargetDrainage: Facilitates participant reward payout when it makes contact with a bouncy ball (assuming the participant is standing close enough). Regardless of whether a reward is actually payed out, this game object is also responsible for destroying individual ball objects once a collision occurs.

Drainage: Exhibits similar behaviour to the TargetDrainage object, except it does not trigger a reward payout.

GameInfoMultiDay/_Dest: Wall interfaces that present important information to the user throughout the lifetime of the simulation, including the current day, the remaining day time, their current wage, and earnings.

3.3 Tutorials & Instructions

InstructionMan: Manages the displaying of instructions throughout the tutorial stage of the simulation.

BucketMarkerTrigger: This game object is an invisible cube with a collider to determine if the participant is standing close to the bucket, signifying they have completed that portion of the tutorial.

BucketMarker: This game object is a red exclamation point used to notify the participant of the bucket's location.

BucketPickedUpTrigger: Used to detect when the bucket is actually picked up by the participant - consists of two rectangular prism-shaped colliders positioned immediately above and below the bucket, such that any movement would cause a collision.

FarSinkMarker: This game object is a red exclamation point used to notify the participant of the destination tub.

FarSinkMarkerTrigger: Similar in functionality to the *BucketMarkerTrigger*. Used to advance to the next tutorial step as soon as the participant is standing close enough to the destination tub.

CameraRig/Camera/Canvas/InstructionPanel: This canvas is attached to the Camera object such that it appears in view at all times and moves with the headset. It contains *InstructionTxt*, which will display instructions to the participant when required.

CameraRig/Camera/Canvas/TransitionPanel: This canvas is attached to the camera so that it appears in the view at all times and moves with the headset. It contains *TransitionCountdown* which will display the number of seconds until a new day begins.

3.4 Receiving Treatment

TreatmentInformationCanvas: This canvas is located just above the treatment station cabinet. It contains two other panels: *WaitRemainingPanel* for displaying the wait-style treatment cost and *PayPanel* for displaying the pay-style treatment cost. *TreatmentInformation* is for displaying the title.

Pay/WaitPills: These game objects are simple prescription pill bottles that

the participant may interact with to receive treatment. There is a bottle for both pay and wait styles of treatment.

Pay/WaitPedestal: These objects are colour-coded to match the UIs holding the pay and wait-time costs. The bottles for pay and wait-style treatment will be placed atop their respectively-coloured pedestals such that the participant is able to clearly distinguish the functionalities of the bottles.

WaitPlatform: This is the platform that the bucket will rest on until the wait-time has elapsed, should the user choose to wait for free treatment. The platform is only rendered if the participant chooses this particular style of treatment. When the bucket is placed on this platform, it cannot be picked up until the wait-time has expired.

3.5 Other Features

Source_Dispense_Pipe: This is a simple pipe model placed above the source sink, which serves as a point to instantiate bouncy balls. No functionality is attached to this game object.

Door: A prefab door and two paintings that are only rendered if the participant indicates they are highly sensitive to both motion sickness and claustrophobia.

AudioManager: Manages the playing of an array of sound effects for a variety of events throughout the lifetime of the simulation.

WaterAudioMan: Similar to the standard AudioManager, except responsible for different sound effects, notably the dispensing of bouncy balls. Since a given audio manager game object can only play one sound effect at a time, it's critical to have at least two included in the scene such that no two sound effects will conflict with each other. For example, a countdown sound effect could be played at the same time that the participant is filling their bucket with bouncy balls.

Source_Tub: A prefab tub over which the bouncy balls are instantiated.

Destination_Tub: A prefab tub for the subject to pour balls into.

4 Application Breakdown: Script Components

4.1 Interactivity

CameraBehaviour.cs: Locates the physical headset object (inside SteamVR CameraRig), then, on every frame, scales its transform positions by the Unity-Vive scale constant (SimManager.UNITY_VIVE_SCALE) to accurately map the headset's position to the virtual environment.

HandTracker.cs: This script is attached to each virtual hand game object within the scene. Its job is to take the corresponding physical controller's transform (inside SteamVR CameraRig), then, as in CameraBehaviour, scale the transform values so that virtual controller is placed properly within the scene. Additionally, once the virtual controller's transform has been determined on every frame, the script modifies the transform using user-defined floating point rotation and translation values so that the models of the hands appear to be in a natural position.
In addition to positioning a virtual hand, HandTracker also handles haptic feedback for its respective controller, according to the strength of an active impairment. The intensity of the haptic feedback is directly proportional to the strength of such an impairment.

Hand.cs: Open source Steam VR hand script to assist with interactability of hands with other assets in the scene. Modified to work with our assets and functionalities.

4.2 Immersion

AudioManager.cs: Offers a public method to play an array of sound effects, including ball instantiation, medicine consumption, day start, day end, simulation end, as well as countdowns. Each one of these sound effects is attached within the editor onto an unassigned public AudioClip variable. Multiple AudioManagers can be placed within the scene in case of potential conflicts; for example, a ball could spawn at the same time a countdown is taking place. The script also includes methods to mute all sounds, as well as stop the current sound. All supported sound effects are defined in the AudioManager.*SoundType* enumerated type. Each AudioManager is designed to be attached to an empty game object within the scene.

4.3 Experiment Setup

DayConfiguration.cs: Encapsulates basic information for each day of the simulation. Specifically, their unique number identifiers (int), durations in seconds (float), impairments, treatment options, and optionally, how much to pay the participant for each delivered ball (float) on the given day. In addition to two constructors, this script also offers accessors for each component.

ConfigParser.cs:: A class that is responsible for parsing the data from the input configuration file and setting experiment variables. It contains methods that divide up the parsing by *Simulation* and *Days*. The *Simulation* part of the parser allows global variables directly related to the look and feel of the experiment to be set. Conversely, the *Day* parser sets the structure of the experiment on a day-by-day basis (duration, impairments, treatments, etc).

SimManager.cs : *establishSimulationParameters ()*: Creates the configuration parser object to read the configuration file for the simulation. On success of all parameters being set, it returns true, otherwise returns false. Instructions, sound, and tutorial scores are all set in this method.

SimManager.cs : *getCurrentDayConfiguration ()*: An accessor created to allow other classes to access the day configuration for the current day as defined by the SimManager.

SimManager.cs : *Update ()*: A method that is called on every frame. Responsible for the main event-driven loop that runs the simulation. It keeps track of the state of the game and transitions between them accordingly. It also manages critical variables that may be persisted into an output file. Essentially, it is the heart of the simulation.

4.4 Task Design

4.4.1 Task Framework

FlowManager.cs: This manages the flow of the bouncy balls from the source pipe. It offers methods to either stop or start the instantiation of bouncy balls. As well, it offers a method to clean the scene; that is, remove all balls from the scene and stop their instantiation.

DrainageBehaviour.cs: Logic for the destination tub: the participant is payed depending on how many balls have been successfully captured inside the tub, and also on whether or not the drainage object is flagged as being a target. Also handles the destruction of balls once a collision occurs.

FlowLimiter.cs: Prevents the tap from flowing except when the participant is standing close enough. The main idea behind this script's implementation was to prevent cheating in that players are forced to physically cross the room rather than reach from one side to the other while standing relatively still. The script is meant to be attached to an invisible game object with a collider enabled (the collider must have *isTrigger* set to true). Collisions will then be detected between this invisible game object and the participant's headset; no collisions will be detected with the participant's virtual hands. The script is also designed to prevent the tap from flowing if the participant is waiting for treatment, or if

the simulation is not in the *RUNNING* state.

DestinationLimiter.cs: Similar to FlowLimiter, except instead of preventing balls from being instantiated, this script prevents the participant from being payed unless they are standing close enough to the destination sink. This script is also meant to be used on an invisible game object with a collider attached (and *isTrigger* = true).

4.4.2 Impairment

Impairment.cs: Encapsulates the two components of an impairment: the type (defined in the Treatment.*ImpairmentType* enum), and strength (float, percentage expressed from 0.0 to 1.0). Offers public methods to retrieve and set each one of these components.

SimManager.cs : *Update ()*: During the transition between days, the method will check if impairments exist in the configuration of the day and will apply them accordingly.

SimManager.cs : *modifyImpairmentFactors (float)*: Iterates over all active impairments for the current day, and decreases their respective strengths by the given factor. For example, if all impairments have a 50% strength, calling modifyImpairmentFactors (0.75) will yield new impairment strengths of 12.5%; or, specifically, strength = original_strength * (1 - factor).

SimManager.cs : *unapplyImpairments ()*: Iterates over all active impairments for the current day (if any), and deactivates them accordingly. Depending on the impairment, the operations to deactive will vary. Functionally, this method is equivalent to calling *modifyImpairmentFactors (1.0)*, that is, remove 100% of the strength for each active impairment.

4.4.3 Offering Treatment

Treatment.cs: Encapsulates all components that make up a given day's treatment options. These components include private float members for C, a, b, c of both the day's pay cost function and its wait time cost function (wait_C, wait_a, wait_b, wait_c and cost_C, cost_a, cost_b, and cost_c, respectively). If a treatment day only offers the choice of paying, then the wait members are set to a static value of Treatment.*NONE*, and vice versa. Other private members include the probability of effectiveness (float 0.0 to 1.0), and effectiveness (also float 0.0 to 1.0). If a treatment's probability of effectiveness is 0.5, and its effectiveness is 0.8, then there exists a 50% chance that the treatment will alleviate 80% of the active impairment(s); conversely, there exists a 50% chance the treatment will have no effect.

This script also offers an assortment of public methods to do things like retrieve the treatment's current cost or its current wait time cost, determine if

the treatment is effective (determined only once and then cached), or to check if the treatment has been obtained already.

PillManager.cs: The class contains logic pertaining to how the participant is able to receive treatment. The bottles only show up on the days that they have the option to receive treatment. The days of treatment, panels are show behind the bottles that show the price or wait time of the treatment. Actions are triggered depending on what the participant chooses to do.

4.4.4 Receiving Treatment

To trigger any action, the participant must pick up the pill bottle they wish to choose. If they choose the pay bottle but do not have enough money to pay for treatment, an error event is triggered and the participant is given auditory feedback. Once a successful transaction has occured, the appropriate panels disappear. For pay-style treatment, all panels disappear and the participant can continue the simulation normally. However, when wait is chosen, only the pay components disappear, and the bucket is placed on a platform on the table. The participant is unable to perform the basic task until the countdown has expired.

4.4.5 Instructions

Instruction.cs: A simple class to encapsulate the two components of any instruction: a string *message*, and a float *displayDuration* (seconds). Instruction objects can be passed into later-described methods in order to display their respective messages for the desired amount of time.

InstructionManager.cs: Offers methods to display instructions on the participant's HUD. When these methods are called, a grey translucent panel is enabled in front of the participant's eyes, and the corresponding instruction text is placed on top. Once an instruction is set and is being displayed, this class is also responsible for disabling the instruction text and panel once the message's display duration has expired. This script is meant to be attached to an empty game object within the scene, and requires instruction text and panel game objects to be attached to it through the Unity editor.

SimManager.cs : *limbo (Instruction [])*: This method is to be used when instructions are required in the midst of the simulation, rather than during the tutorial day, *Day Zero*. The programmer must first define an array of Instruction objects, and then pass those into the *limbo* method. Once the method is called, the simulation will enter the *LIMBO* state. In this state, the user is not able to complete the task or interact with the environment in any way. Instead, the grey translucent instruction panel will be enabled, and each instruction in the array will be displayed sequentially. Once all instructions have been displayed for their respective durations, the exitLimbo() method should be called

so that the simulation will resume in the *RUNNING* state.

GoalMarker.cs: A script that defines the behaviour of a point of interest marker meant to accompany a given instruction. The marker will move smoothly up and down, keeping a constant X and Z position within the scene. It requires the programmer to set the upper and lower Y bounds, as well as how fast it should move (all public float variables). This script is designed to be attached to marker game objects within the scene - if the object is activated, it will always exhibit the behaviour defined in this script.

InstructionTrigger.cs: There are different events that should trigger the next instruction set in a sequence; for example, a certain time duration having elapsed, some threshold being met, or the participant (or some other object) physically moving to a specified location within the scene. This script allows the latter, and is meant to be attached to an invisible game object within the scene (this object must have a collider that has *isTrigger* enabled). In order for the script to work, the following public members must be assigned through the Unity editor: the SimManager, the next Tutorial Step to advance to (use TutorialStep.NULL unless multiple InstructionTrigger objects are being used for one instruction and could try to advance the tutorial step at the same time; in this case, it is advisable to explicitly define which step to advance to such that no duplicate advances are made) and the object to detect collisions with (such as the participant headset, or the container). Optionally, the programmer may also attach a game object to the destroyOnTrigger member - this object will be destroyed whenever the aforementioned collision takes place. This is useful when a Goal Marker is being used in conjunction with an instruction. Once the instruction is no longer needed, such as when the participant moves to a certain location, the Goal Marker is also no longer needed and should be destroyed.

4.4.6 User Interfaces

MultiDayUIUpdate.cs: Responsible for updating the Wall UI components with time remaining, current day, current wage, and earnings across multiple days of the experiment. Its behaviour depends on how many days are being used for the experiment; notably, the layout will display at most four days worth of earnings. If more than four days are used in the experiment configuration, then the script will display earnings for the four most recent days. In order to facilitate this main functionality, the script has public methods, which should be called only by SimManager, to set the current day, total number of days, among other critical fields.

TransitionMessage.cs: When in the transition state, this script will populate text fields to indicate to the participant how much time remains until the next day of the simulation. As well, it will tell the user whether or not the next day is a full health day, or an impaired day.

4.5 Custom Configuration

ConfigKeyword.cs: Sealed class that contains static read-only keywords used by the ConfigParser to read the input file written by the user.

ConfigParser.cs: This class allows the experiment to be structured as per the researcher's preferences through a configuration file. This class reads the configuration file by scanning for keywords, as provided by the ConfigKeyword class. During the parsing process, it first sets the *Simulation* preferences such as whether or not sound is enabled. Next, it sets the structure of the tutorial, *Day Zero*, with the corresponding values of the minimum scores required to move on to the first paid day of the experiment. Finally, the structure of each *Day* is set. Key variables per day include duration, active impairments, treatment options, and payout per ball. When the parser finishes a given day, it creates an object of type *DayConfiguration* to encapsulate all of this information.

4.6 Persistence

ParticipantData.cs: Static class populated after the participant presses the start button on the welcome screen. The participant's name and sensitivity information is gathered from the input form, and kept inside public static variables *name* (string), *nauseaSensitive* (bool), and *claustrophobicSensitive* (bool). These values can be accessed at any point during runtime.

PopulateParticipantData.cs: Facilitates the population of the ParticipantData static class. Each field of the welcome screen input form is mapped to GameObject variable within this script. When the confirm button is pressed, the values of each form element are retrieved, and the ParticipantData variables are assigned accordingly. Once this process is complete, this script also loads the main simulation scene, using a *LoadLevel* method call.

SimPersister.cs: Contains definitions for log file naming, output data directories, as well as output data string formats. When instantiated, this class validates the output directory, and creates a new text file for persistence, which is named using the start time and participant name (if available). Once the output file is created, the constructor calls the *writeIntroduction()* method, which summarizes the participant and application information, and prints CSV column headers. This class offers a public method persist() - it takes a number of important simulation metrics as arguments, and persists them into the output file in CSV format.

5 User Manual: Configuration File Breakdown

This document outlines how to modify the simulation configuration file in order to achieve the desired experiment format. The application expects such a file to be named *sim_config.txt*, and for it to be located in the *Ball_Sim_x.x/ simulation_one_Data/InputData* directory. Failing to follow this naming convention or moving the file out of this location will result in the application being unable to start up properly. The configuration file supports a number of keyword-value pairs that can be easily combined to define the temporal structure of the simulation as well as auxiliary parameters.

5.1 Simulation

This section includes information pertaining to high-level details of the simulation. This keyword is not tabbed; each of the following four keywords are tabbed once.

5.1.1 Name

For user convenience - offers a simple way of differentiating multiple configuration file setups. No restrictions on value.

5.1.2 Description

Similar to the above: offers a way to describe a given simulation configuration for documentation purposes. No restrictions on value.

5.1.3 Instructions

Toggles whether or not instructions will appear throughout the experiment. These include both the *Day Zero* tutorial and pre-treatment instruction sets.

Possible values:

enabled
disabled

5.1.4 Sound

Disables or enables all sound effects within the scene, including pipe flow and countdowns, among others.

Possible values:

enabled
disabled

5.1.5 Sample Simulation Configuration:

```
Simulation
    Name: April 2019 configuration.
    Description: Stronger impairments.
    Instructions: enabled
    Sound: enabled
```

5.2 Tutorial

This keyword refers to the *Day Zero* tutorial preceding the main experiment. During this stage of the simulation, the user is required to earn a certain amount of money before being allowed to move on to the first paid day. Optionally, *Day Zero* can be split into two portions: one unimpaired portion, and one impaired portion. The impaired portion is preceded by an instruction set. The second portion will only take place if the *Impairment* keyword is used when configuring the tutorial. Usage of the *Impairment* keyword is described later in this document. The *Tutorial* keyword is not tabbed, while the *Score*, *ImpairedScore*, and *Impairment* keywords are tabbed once. Sub-keywords for the optional second portion impairment, *Strength* and *Type*, should be tabbed three times.

5.2.1 Score

Sets the amount that the participant will need to earn to pass the tutorial day. If not specified, the default is **$150.00**.

Possible values:

Any decimal number

5.2.2 ImpairedScore

Sets the amount that the participant will need to earn to pass the optional second portion of the tutorial day. If not specified, the default is also **$150.00**.

Possible values:

Any decimal number

5.2.3 Sample tutorial configuration:

```
Tutorial
    Score:175
    ImpairedScore:125
    Impairment
        Type: Physical/Shake
        Strength:50%
```

5.3 Day

Signifies a new day to be included in the experiment. Each day may include the following sub-keywords: *Duration*, *Impairment*, and *Treatment*. The *Day* keyword is not tabbed, and each associated sub-keyword is tabbed once.

5.3.1 Duration

This keyword refers to how long each day will last. This field is **mandatory**.

Possible Values:

minutes:seconds, where both components must be positive integers

5.3.2 BallValue

This keyword allows for the payout per ball to be adjusted. This is not a required field, and by default it is set to 1.

Possible Values:

Any decimal number

5.3.3 Impairment

This keyword will determine which impairments will be in effect during the day. There can be one or more impairments imposed on a single day, and each one requires its own *Impairment* keyword. *Impairment* is tabbed once. Associated sub-keywords, *Type* and *Strength*, are tabbed twice.

Type

Possible values:

Physical/Shake

Visual/Fog

Strength

Possible values:

Any integer between 0-100 followed by a % sign

5.3.4 Treatment

This keyword is used to specify which treatment options will be available to the participant on the given day. The cost of obtaining a treatment follows the functional form of $C(c - bT + aT^2)$, where by default, the values of a, b, and c are 1/day length in minutes, 2, and day length in minutes, respectively. The functional form is consistent across both pay-style treatments (where cost is in dollars), and wait-style treatments (where cost is in seconds). When the *Wait* or *Cost* keywords are included, their respective **C** values are **mandatory**; and, if all three of their respective a, b, and c values are excluded entirely, they will be set to default.

Typically, a treatment alleviates 100% of all active impairments, with 100% certainty. If desired, these two values can be modified using the *Effectiveness* keyword, with sub-keywords *Effect*, and *Probability*, respectively. These two fields are not mandatory and will default to 100% each if excluded.

Treatment is tabbed once while sub-keywords *Effectiveness*, *Wait*, and *Cost* are tabbed twice. *C*, *a*, *b*, *c*, *Effect*, and *Probability* are tabbed three times.

C
Possible values:
Any decimal number

a, b, c
Possible values:
Any decimal number
default

Effect
Possible values:
Any decimal number, followed by a % sign

Probability
Possible values:
Any decimal number, followed by a % sign

5.4 Sample Configuration File

```
Simulation
    Name:Full sample configuration file
    Description:Sample 3-day (plus intro) configuration file
    Instructions:enabled
    Sound:enabled
Tutorial
    Score:180
    ImpairedScore:125
    Impairment
        Type:Physical/Shake
        Strength:50%
Day
    Duration:1:30
Day
    BallValue:1.50
    Duration:1:30
    Impairment
        Type:Physical/Shake
        Strength:70%
Day
    BallValue:1.25
    Duration:2:00
    Impairment
        Type:Visual/Fog
        Strength:75%
    Impairment
        Type:Physical/Shake
        Strength:50%
    Treatment
        Effectiveness
            Probability:50%
            Effect:90%
        Wait
            C:0.5
            a:0.1111
            b:3
            c:1.5
        Cost
            C:80
            a:default
            b:default
            c:default
```