**Table of Contents**

[Installation](#_Toc132987855)

[Directory Structure](#_Toc132987856)

[Assets](#_Toc132987857)

[Build](#_Toc132987858)

[Psiturk\_Wrapper](#_Toc132987859)

[Platform Types](#_Toc132987860)

[Experiment Types](#_Toc132987861)

[Basic Unity UI Guide](#_Toc132987862)

[Codebase](#_Toc132987863)

[BeginExperiment.cs](#_Toc132987864)

[Participant Code Validation](#_Toc132987865)

[Session Detection](#_Toc132987866)

[Session Auto-population](#_Toc132987867)

[CoroutineExperiment.cs](#_Toc132987868)

[DoMicrophoneTest()](#_Toc132987869)

[DoVideo()](#_Toc132987870)

[PressAnyKey()](#_Toc132987871)

[DeliveryExperiment.cs](#_Toc132987872)

[Constant Variables](#_Toc132987873)

[ConfigureExperiment()](#_Toc132987874)

[Update()](#_Toc132987875)

[Start()](#_Toc132987876)

[ExperimentCoroutine()](#_Toc132987877)

[DoSubSession()](#_Toc132987878)

[DoIntros()](#_Toc132987879)

[DoRecapInstructions()](#_Toc132987880)

[DoFamiliarization()](#_Toc132987881)

[DoTownLearning()](#_Toc132987882)

[DoDeliveries()](#_Toc132987883)

[DoPracticeTrials()](#_Toc132987884)

[DoTrials()](#_Toc132987885)

[DoTypedResponses()](#_Toc132987886)

[DoRecall()](#_Toc132987887)

[PickNextStore()](#_Toc132987888)

[DeliveryItems.cs](#_Toc132987889)

[ReliableRandom()](#_Toc132987890)

[PopStoreName()](#_Toc132987891)

[PopItem()](#_Toc132987892)

[ItemExhausted()](#_Toc132987893)

[FlexibleConfig.cs](#_Toc132987894)

[Static Variables](#_Toc132987895)

[GetSetting()](#_Toc132987896)

[GetSystemConfig()](#_Toc132987897)

[GetExperimentConfig()](#_Toc132987898)

[SaveConfigs()](#_Toc132987899)

[GetOnlineConfig()](#_Toc132987900)

[FpsDisplayer.cs](#_Toc132987901)

[FreiburgSyncbox.cs](#_Toc132987902)

[LanguageSource.cs](#_Toc132987903)

[MessageImageDisplayer.cs](#_Toc132987904)

[Update()](#_Toc132987905)

[DisplayMessage()](#_Toc132987906)

[DisplayMessageTimed()](#_Toc132987907)

[DisplayMessageLRKeypressBold()](#_Toc132987908)

[DisplayMessageTimedKeypressBold()](#_Toc132987909)

[DisplaySlidingScaleMessage()](#_Toc132987910)

[DisplaySlidingScale2Message()](#_Toc132987911)

[Another version of the slider message format.](#_Toc132987912)

[SetCuedRecallMessage()](#_Toc132987913)

[DoProgressDisplay()](#_Toc132987914)

[SetGeneralMessageText()](#_Toc132987915)

[SetGeneralBigMessageText()](#_Toc132987916)

[SetGeneralBiggerMessageText()](#_Toc132987917)

[DoTextBoldTimedOrButton()](#_Toc132987918)

[PlayerMovement.cs](#_Toc132987919)

[StoreComponent.cs](#_Toc132987920)

[GetStoreName()](#_Toc132987921)

[Start()](#_Toc132987922)

[DrawSigns()](#_Toc132987923)

[IsVisible()](#_Toc132987924)

[PlayerInDeliveryPosition()](#_Toc132987925)

[PopItem()](#_Toc132987926)

[VideoSelector.cs](#_Toc132987927)

[OnEnable()](#_Toc132987928)

[VideoType](#_Toc132987929)

[SetVideo()](#_Toc132987930)

[Running Courier Online](#_Toc132987931)

[Setting up](#_Toc132987932)

[Creating a build](#_Toc132987933)

[Uploading to server](#_Toc132987934)

[Launching Courier Online experiment](#_Toc132987935)

This is in-depth documentation on the Courier [repository](https://github.com/pennmem/Courier), guiding through the codebase. This guide also assumes that the reader has enough familiarity with both C# and Unity classes. If you are not familiar with/want to know more about C# or Unity, it is highly recommended that you read through the official Unity documentation. Without further due, let’s dive right in.

# Installation

There are two things to download: [Unity Hub](https://unity.com/download) (including the actual Unity program) and [Courier repository](https://github.com/pennmem/Courier).

Detailed steps on Unity Hub installation.

1. Download Unity Hub
2. Create an account
3. Click Activate New License Button
4. Select “Unity Plus or Pro” and put in the serial number

**E4-YNQR-25UA-B53X-3YQR-9BDJ**

1. Follow the rest of the steps

Unity license needs to be renewed every year. For the most up-to-date license information, refer to this [link](https://upenn.box.com/s/45telibdbv64oco6dshwecezprcip37i) in box.

The current Courier repository is developed on Unity version 2021.3.2f1. Make sure to install “Mac Build Support (IL2CPP)” and “WebGL Build Support.” And this concludes the installation for the Unity side.

As for the Courier repository, you will need to additionally download the “Town Constructor 3” folder from the box link [here](https://upenn.box.com/s/1s1ba0u6tf2hktigw0z7onxciduswpl1). Once downloaded, place the folder under the Assets folder and reload the Unity application.

# Directory Structure

There are many folders/files inside the courier repo. Don’t get frightened. Here are the ones that you will be working with the most.

## Assets

any item that you use in your Unity project to create your game or app, which includes audio, video, 3D models, scripts, etc.

* Audio
  + StoreAudioEnglish: holds all the item recordings for all the stores (e.g., bagels.wav for Bakery)
  + StoreAudioGerman: German translation of StoreAudioEnglish assets
  + beephigh.wav, beeplow.wav
* ElememInterface
* FreiburgSyncboxInterface
* Images
  + Recap Instructions for various task types (exported from ppt slides)
* NiclsInterface
* Plugins
  + DLLs (e.g., Accord.dll, MathNet.Numeric.dll, etc.)
  + PsiturkPlugin.jslib
* RamulatorInterface
* Scenes
  + MainMenu.unity
  + MainGame.unity
* Scripts: most of the necessary codes are located here
* StreamingAssets: assets for webGL builds
* Town Constructor 3: courier town models
* UnityEPL
* Video

## Build

Unity build system for webGL. For more description, refer to the [official documentation](https://docs.unity3d.com/Manual/webgl-building.html).

## Psiturk\_Wrapper

Contains materials needed to run Courier on the Psiturk server. DO NOT modify the script unless you know exactly what you are doing. The most up-to-date files are in CMLPsiturk server (maint@cmlpsiturk.compmemlab.org:~/Value\_Courier)

* config.txt: config for Psiturk server
* static: assets needed for webGL builds.
  + ./js/Unity/build: symlink to the actual build folder
* templates: HTML templates for the Psiturk server, which include the ad, consent, survey, completion page, and all others.

# Platform Types

There are two types of platforms that Courier can be run on: PC and WebGL. PC platform builds are usually used in hospital settings and scalp lab settings. WebGL platform builds are designed to be run on the online server, mainly Psiturk.

To change the platform, click File/Build Settings…/ on the left top corner (for the Mac version of Unity).

Graphical user interface

Description automatically generated

Here, you can see multiple platforms available, but the two that we are most interested in are Windows, Mac, Linux, and WebGL options.

Click on the desired platform, and the bottom right button prompt will change to “Switch Platform.” Click that button, and Unity will start compiling for the new platform.

If you are building for WebGL, please keep the options the same as displayed on the left. For some reason, the game will not build with the “Faster runtime” option for IL2CPP Code Generation.

You can also add/remove the scenes that will be part of your final build.

From a programming perspective, we need to be careful about what to add to each platform. The reason why this is important, for one example, is that for webGL, we don’t want any system IO operations to take place, as it is not possible in the first place due to browser security restrictions. To ensure this separation and maintain a single codebase, we will use **conditional** **compilation**. With this functionality, we can simply divide up the code into parts by using #if statements. For more detail, refer to this [link](https://docs.unity3d.com/Manual/PlatformDependentCompilation.html). You will see this conditional compilation throughout the scripts.

# Experiment Types

There are three types of experiments existing in the Courier repo: NICLS, EFR, and Value Courier. For more design details on each of the experiments, refer to the design documents in box.

From a programming perspective, we specify them as Boolean variables in two scripts: *BeginExperiment.cs* and *DeliveryExperiment.cs*. Every time you want to switch the experiment type, make sure to set the Boolean values accordingly. These values are always initialized at the beginning of the class object. For convenience, it might be a good idea to make these config variables, but for now, they should be set manually.

# Basic Unity UI Guide

Graphical user interface, application

Description automatically generated

There are four compartments to the Unity editor screen: Hierarchy (top left), main screen (middle), Inspector (top right), and Project/Console (bottom).

Hierarchy section shows all the GameObjects that exist on the main screen. You can expand the object by clicking the downward arrow on the left. Try to click each object and understand how they are linked to the main menu screen. They are pretty self-evident!

You will also notice that on the right panel, the Inspector section is now filled with information (Rect Transform, Image, Input Field, etc). Feel free to adjust each variable to get a clear understanding. If you are still lost, please refer to the official documentation.

**A picture containing table

Description automatically generated**

One thing to keep in mind when creating/manipulating any game object in the game. Please press that R button to scale the object with respect to the display. If you miss this, then your object will be out of scale when displayed on different displays.

Lastly, whenever you make changes to the scripts, Unity will auto-compile your changes and display warnings and errors if there are any.

# Codebase

As mentioned above in the Directory Structure section, “Scripts” folder holds (almost) all the necessary scripts. Let’s break them down one by one.

## BeginExperiment.cs

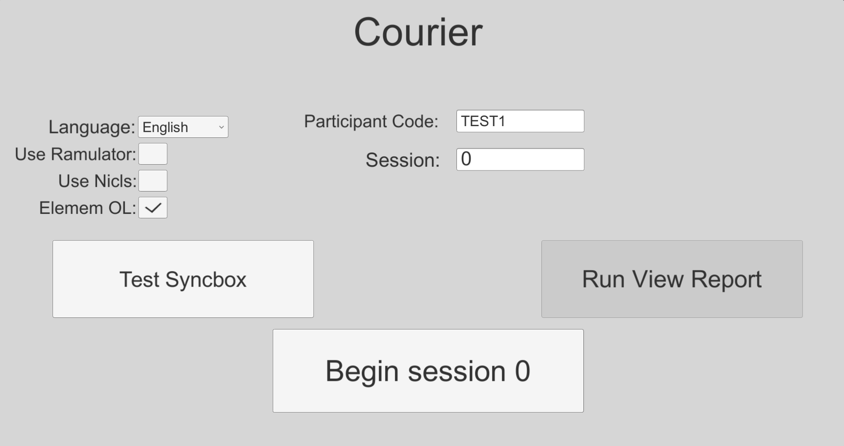
This script is linked to MainMenu.unity scene.

Graphical user interface, text, application

Description automatically generatedThis script is straightforward, as it passes down whatever participant code, session, and toggle options you give in to DeliveryExperiment.cs (which we will discuss in the next part).

Each toggle option is linked to the Boolean value, which then get passed to a function called ConfigureExperiment().

After filling out all the necessary blanks, the screen will now look like this.



If you press “Begin session 0”, it will trigger the script to run DoBeginExperiment() function, which will lead us to the actual gameplay.

Few features to note.

### Participant Code Validation

Currently, it only checks whether there is an input to the participant code. However, you can uncomment the codes under the function IsValidParticipantName(), which checks for the “correct format” of the subject ID (e.g., R1XXXJ). This functionality has been disabled due to the following reasons. 1) NICLS courier is a scalp experiment where it uses LTP as a prefix for the subject code. 2) Value courier has been run under a class setting, so we needed to turn this participant code check off.

### Session Detection

EFRCourier has two versions: ReadOnly and OpenLoop, and sessions numbers do not carry over to different versions. For example, if you have already run EFRCourierReadOnly session 0 and want to run EFRCourierOpenLoop session next, then you need to label the session as 0, not 1. As EFRCourier is mainly run in hospital settings, it is important to cut down as much time as possible to acquire more session data. Therefore, if you type in 0 for the session number, the script will automatically look for the session folders and see if this participant has run either version of EFRCourier prior to this session. If so, then it will pass in a Boolean value of false for *isFirstSession* variable, which will turn off the instruction video in the beginning. (only available on PC platform)

### Session Auto-population

Based on the participant code, NextSessionNumber() function will determine the next session number by browsing through the default participant data directory. (only available on PC platform)

## CoroutineExperiment.cs

A coroutine is the backbone of Unity architecture, which allows you to spread tasks across several frames. In Unity, a coroutine is a method that can pause execution and return control to Unity but then continue where it left off on the following frame. Here, it contains the basic IEnumerator functions such as DoMicrophoneTest, DoVideo, PressAnyKey, etc.

### DoMicrophoneTest()

Records the microphone testing audio and saves out the recording under the session data directory. Only works on PC platform build.

### DoVideo()

Starts the video.

### PressAnyKey()

Waits for any key to be pressed. Once Unity detects any key press, it moves on to the next IEnumerator call. Behaves just like a while function.

## DeliveryExperiment.cs

This script is the highlight of the Courier codebase. For a basic scripting guide, refer [here](https://docs.unity3d.com/Manual/ScriptingSection.html). We will touch upon most of the functions here, but not all of them due to the sheer amount of code.

### Constant Variables

We will assign constant variables (e.g., free recall duration, recall message display duration, etc) at the beginning. It is imperative that you keep these default variables in place. These variables are assigned after careful experimental design choices, and they shouldn’t be changed unless otherwise directed. For debugging purposes, comment out the original values and input the desired values.

### ConfigureExperiment()

As the name of the function implies, this function configures the experiment accordingly based on the inputs that we give in. It should be called before starting the actual experiment.

### Update()

Update function is called every frame of the experiment. If you want something to happen in every frame, this is the place where you want to write code.

### Start()

Start is called when the GameObject begins to exist (either when the Scene is loaded or the GameObject is instantiated). Here, we start the experiment coroutine.

### ExperimentCoroutine()

This function holds the game logic for all three experiment types. The basic logic is as follows, but do note that it may vary based on the experiment type:

* Send BeginNewSession message to the appropriate system (Ramulator/Elemem)
* Save current configuration settings to the participant data directory
* Environment Initialization (refer to DeliveryItems.cs and StoreComponent.cs scripts)
* Intros
  + video/ recap instruction slides
* Town Learning
* Practice Trials
* Trials
  + navigation
  + free/cued recall
  + final free/store recall
* End

### DoSubSession()

Although it is called SubSession, this is a session-level function in reality. Here, the function will run X trials and finish the session with final recall tasks. For NICLS Courier, there will be two sub-sessions with a break in between (hence the name of SubSession instead of regular Session).

### DoIntros()

Displays introduction videos based on the experiment types. Refer to VideoSelector.cs script for more detail on how to assign videos to each experiment.

### DoRecapInstructions()

In addition to intros, this function, when called, displays a series of instruction slides. Participants can browse through the slides for as long as they want.

### DoFamiliarization()

Displays images of all the stores for familiarization purposes. Currently not in use and replaced with town learning and practice trial phases.

### DoTownLearning()

Starts a town learning phase, where participants are asked to navigate around the town indicated by the prompt on the left corner of the screen. Depending on the experiment type, a pointing task is presented before every delivery ( DoPointingTask() ).

### DoDeliveries()

This is where actual navigation & item presentation happens, and it is crucial that you understand the logic behind it. Please go over the code closely and read through the comments. Here is the simplified logic for your convenience.

For each delivery, we will pick the next store to visit (refer to PickNextStore() function for more detail). Depending on the experiment type, a pointing task is presented before the actual navigation to the next store. Upon arrival at the next store (refer to DeliveryZone.cs on how to detect arrival), display both the audio and the text of the item on the screen.

### DoPracticeTrials()

Runs through practice trials. Unlike actual trials, depending on the experiment type, it will display instruction videos in between the phases. Design is straightforward.

### DoTrials()

Runs through the actual trials. Calls DoDeliveries() and DoRecall() functions to initiate delivery and recall phases for X number of trials specified in config.

### DoTypedResponses()

Records the typed responses. This was designed to replace audio recordings for recording recall responses. It is mainly used for the online version of the task.

### DoRecall()

Handles the recall phase. Calls DoFreeRecall(), DoCuedRecall(), and DoValueRecall() functions. These recall functions are self-explanatory, so we will skip the details here.

### PickNextStore()

Given the list of stores, determines the next store to visit based on the criteria described [here](https://upenn.box.com/s/35ktgnz5rqi9tvn7w56r7ta27zvumcgg).

## DeliveryItems.cs

Weird design choice here, but this script is where we populate the store names based on the participant code. Also, this script handles item-related events, such as saving out all\_items.txt, all\_stores.txt, and remaining items after the end of the session.

### ReliableRandom()

Similar thing to setting a seed for a random generator. In this case, to ensure that subjects visit the same town layout, we set the random seed based on the participant code.

### PopStoreName()

Sequentially returns the store name from randomly shuffled total store lists. This function is called in StoreComponent.cs when initializing the store objects in town.

### PopItem()

For the PC platform, return an item to deliver for a given store by accessing the remaining store/item folders, and update the remaining store/item information afterward. For the WebGL platform, return an item to deliver randomly from the remaining item lists.

### ItemExhausted()

Checks whether there are remaining items available for delivery or not.

## FlexibleConfig.cs

For some experimental settings (e.g., number of deliveries, number of trials, etc), it is better to create a separate config file and read the configuration settings accordingly while running the experiment. The default location for these config files should be under /data/configs/.

### Static Variables

Here are the static config variables that are later used through various scripts.

### GetSetting()

Fetches both experiment and system configs by calling GetExperimentConfig(), GetSystemConfig(), and see if there are any corresponding config variables within config files.

### GetSystemConfig()

Looks for “config.json” file under the usual config directory (or StreamingAssets folder if using WebGL build). Once located, loads the text within the file, which will then be readily readable through TryGetValue() function.

### GetExperimentConfig()

Looks for a specific config file (.json) that matches the experiment type (e.g., NICLS, EFR, and Value Courier) under the usual config directory (or StreamingAssets folder if using WebGL build). Once located, loads the text within the file, which will then be readily readable through TryGetValue() function.

### SaveConfigs()

Records both system and experiment configs on the event log. For PC builds, saves out both config files externally under the session data directory.

### GetOnlineConfig()

Fetches the Courier Online config files from the StreamingAssets folder. Unlike the PC builds, where you can directly read from the files, WebGL builds need to send a web request to “download” the config files. Refer to this [documentation](https://docs.unity3d.com/Manual/webgl-networking.html) for more details on how web requests work on Unity webGL. One caveat is that if you were to debug the webGL builds on Unity editor, web requests don’t work, so you would have to manually read the config files, just like the PC builds.

## FpsDisplayer.cs

This script allows the game screen to have FPS show up on the top left of the screen. This was initially developed to monitor the gaming performance in real-time.

## FreiburgSyncbox.cs

This script is for the Freiburg sync box code, sending pulses for data alignment.

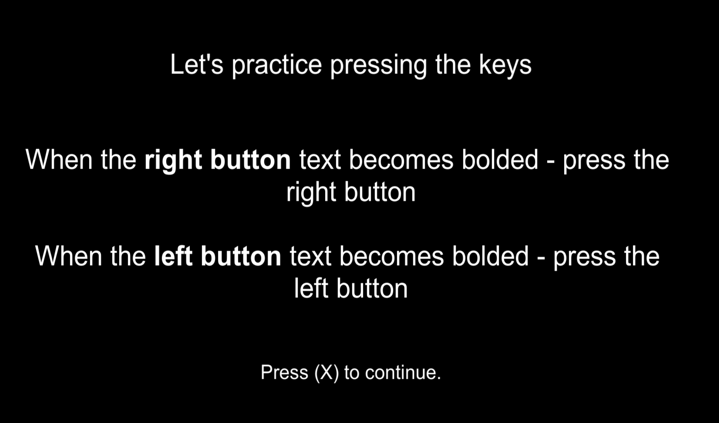
## LanguageSource.cs

All the prompts used in the experiment exist in this script (both English and German). They are saved as dictionary objects, which consist of a prompt title and the actual prompts in a list.

## MessageImageDisplayer.cs

This script handles the “message” portion of the experiment. What this class enables us to do is that we no longer need to create an unnecessary number of new text GameObjects to display messages. Depending on the format/size of the message that you want to display on the screen, you can choose amongst these options: general message, general big message, and general bigger message.

Below are the screenshots of these message formats. You can also check them by selecting them under MessageImageDisplayer in the Hierarchy panel on the left.

**Graphical user interface, application

Description automatically generated general message general big message**

### Update()

If the progress display is active, updates the progress bar fill amount.

### DisplayMessage()

Displays a given message GameObject and waits for the appropriate key/button to be pressed to move on.

*DisplayMessageFunction()*

Displays a given message GameObject just like DisplayMessage(). However, it now can “overlay” another message on top. Refer to DoCuedRecallDisplay() in DeliveryExperiment.cs.

### DisplayMessageTimed()

Displays a given message GameObject for a given amount of time. Can skip with a secret button (“q”).

### DisplayMessageLRKeypressBold()

Displays a given message GameObject and requires pressing a corresponding key/button to move on. When pressing the right key/button, the text will be bolded.

### DisplayMessageTimedKeypressBold()

Has the exact functionality as DisplayMessageLRKeypressBold(), but has the display show up for a certain amount of time.

### DisplaySlidingScaleMessage()

Timeline

Description automatically generated

Displays a sliding scale message GameObject (shown on the left). To understand more about slider objects, refer to this [documentation](https://docs.unity3d.com/ScriptReference/UIElements.Slider.html).

### DisplaySlidingScale2Message()

### Another version of the slider message format.

### SetCuedRecallMessage()

Sets up the message GameObject with given input texts.

### DoProgressDisplay()

Turns on the progress display for a set amount of time given. Once the progress GameObject is set to active, Update() will update the progress display until the time ends.

### SetGeneralMessageText()

Sets up the message GameObject based on the text, which will then grab each component within the GameObject (e.g., title, main, and continue text).

### SetGeneralBigMessageText()

Same as SetGeneralMessageText.

### SetGeneralBiggerMessageText()

Same as SetGeneralMessageText.

### DoTextBoldTimedOrButton()

Make the given display text bold for a given set amount of time or until a corresponding key/button is pressed.

## PlayerMovement.cs

This script handles all necessary parts for player movement.

## StoreComponent.cs

This script defines the StoreComponent class.

### GetStoreName()

Returns a store name.

### Start()

Initializes all necessary parts of the store object (e.g., delivery zone, store name, and building signs). Notice that the store name is generated by calling deliveryItems.PopStoreName().

### DrawSigns()

Given the store name, display the corresponding store name on the building sign.

### IsVisible()

Checks whether a given store is visible on the game screen.

### PlayerInDeliveryPosition()

Checks whether a player is in a delivery position or not.

### PopItem()

Returns a random item to display from remaining items for a given store.

## VideoSelector.cs

This script is where you assign various videos. You would need to initialize a video object (UnityEngine.Video.VideoPlayer) first. For more info, refer to this [document](https://docs.unity3d.com/ScriptReference/Video.VideoPlayer.html).

### OnEnable()

Turns on the video that is assigned to videoPlayer.

### VideoType

All video types in an enumeration type.

### SetVideo()

Sets up a video based on a given video type. For the WebGL platform, grab the URL for the videos that are located under StreamingAsset folder.

# Running Courier Online

There are two important things to keep in mind when coding an online version of the experiment.

1. WebGL platform does not support dynamic types.
2. WebGL platform does not allow System I.O operations (obviously).

Other than that, everything should be identical in terms of coding up the experiment. Here are more detailed steps on how to get everything ready for the Courier experiment to run on the online server (currently using Psiturk).

## Setting up

For switching the platform, refer to this [guide](#_Platform_Types). Once you are under the WebGL platform, check whether there are any compile errors appearing on the error console at the bottom. If not, we are ready to edit/build the experiment!

## Creating a build

On Build Setting panel, select “*Clean Build…*” by clicking a downward arrow on “*Build*” button on the bottom right. Name the folder “build”.

## Uploading to server

There are three layers of servers that you will need to scp the build folder through:

Local rhino server (<user\_name>@rhino2.psych.upenn.edu) ->

Maint server ([maint@rhino2.psych.upenn.edu](mailto:maint@rhino2.psych.upenn.edu)) ->

CMLPsiturk server (maint@cmlpsiturk.compmemlab.org)

For server access, ask any member of the System Development team to grant you access.

In the CMLPsiturk server, you will notice two folders: “Value\_Courier” and “courier\_online\_class”. The former is designed to run on MTurk, and the latter is designed to run on Mike’s memory class. The exact location to scp the build folder is “./Psiturk\_Wrapper/static/js/Unity/”. Now, it is ready for the actual launch!

## Launching an experiment

Refer to this [documentation](https://docs.google.com/document/d/1QwO0-tD3Jj0th-hyYwQNGvC5oqLdFDJ6_kmvXGzOlTI/edit) on how to set up the right config for launching Psiturk experiments. The overall workflow is as follows (assuming that it is the first time setting up the experiment),

1. Open a new screen
2. Conda activate “pturk” environment
3. Under Psiturk\_Wrapper directory, type “psiturk” to launch an experimental server
4. Type “server on” to turn on the server
5. Launch experiment
   1. By manually setting the HIT (e.g., “hit create 9 10.00 4”)
   2. By using an automated script (e.g., psiturk\_batcher.sh)
6. For debugging purposes, navigate to a specific URL to try it out yourself (<http://exp1.compmemlab.org:22001/> for Value Courier).