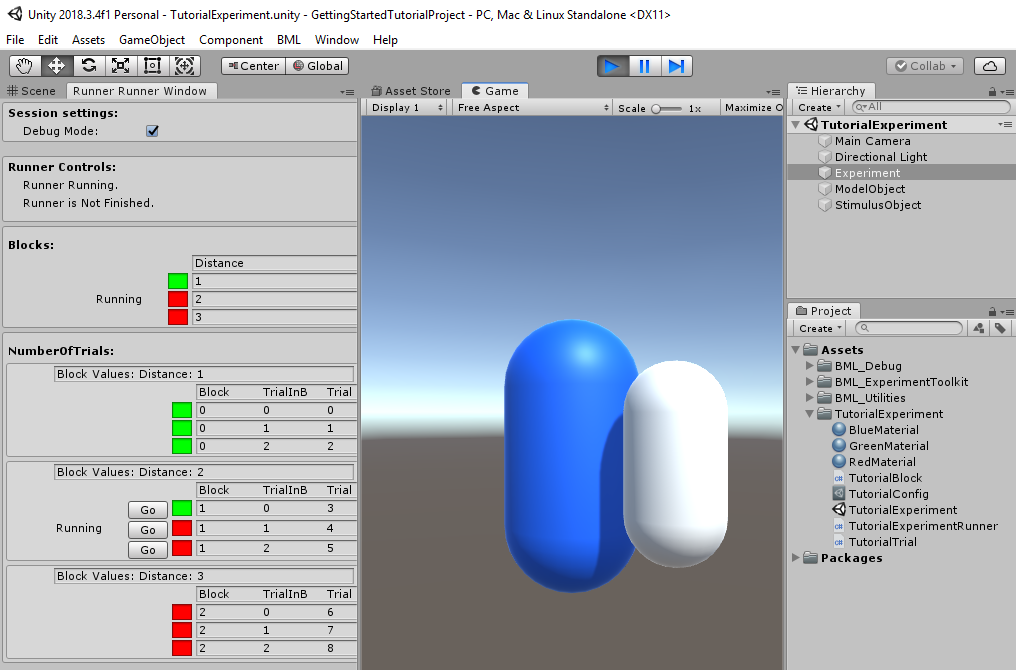
TutorialExperiment: Creating your first experiment using the BML Experiment Toolkit



Contents

[Overview 1](#_Toc6240321)

[Requirements 1](#_Toc6240322)

[Set up a new experiment for the project 1](#_Toc6240323)

[Set up the Unity Scene 2](#_Toc6240324)

[Configuring the experimental design 3](#_Toc6240325)

[Test out our experiment design using basic components 5](#_Toc6240326)

[Define what happens in a trial 6](#_Toc6240327)

[Setting up the Stimuli 8](#_Toc6240328)

[Referencing GameObjects in your experiment 9](#_Toc6240329)

[Participant input 12](#_Toc6240330)

[Outputting Data 14](#_Toc6240331)

[Defining Blocks 15](#_Toc6240332)

[Moving forward 17](#_Toc6240333)

# Overview

This tutorial will guide you through the basic steps to make a simple experiment. For this tutorial our experiment will present the participant with a simple stimulus, and then have the participant match the size of another object to the stimulus. We will vary the stimulus’ size, and distance, and the color of the stimulus. We will also collect participant’s age and assign them an ID. We want to record how close the participant matched the size, and the time it took them.

The finished project from this tutorial can be found in the documentation folder for reference.

Note: This tutorial, for simplicity, does not teach best coding practices. It is designed to show the basic functionality of the experiment framework, and in some instances purposely uses poor practices to increase clarity.

# Requirements

* See Requirements section in main documentation for a more exhaustive list and tips on setting up unity.
* This tutorial was written for BML Experiment Toolkit version 0.6.0. Since it is still under heavy development some names/process may be different in newer versions.
* This is written for Visual Studio 2017, Unity 2019.1
* Have the Visual Studio Unity Tools extension installed.
* [Highly recommended] Using JetBrains Resharper Plugin for Visual Studio. Resharper also has its own Unity Extension for unity-specific help. It makes your coding life way easier and saves a ton of time. It keeps your code clean and suggests changes/problems that may come up.

# Set up a new experiment for the project

1. Create a new Unity Project
2. Go to Edit > Project Settings > Player:
   1. Change the Api Compatibility Level to .Net 4.0
   2. Change the Scripting Runtime Version to .Net 4.x Equivalent
3. Download the latest release of BML\_ExperimentToolkit from NEED LINK
4. Import the .unitypackage in the downloaded folder:
   1. In Unity, go to Assets > Import Package > Custom Package and browse to the .unitypackage.
   2. Click Import all

# Set up the Unity Scene

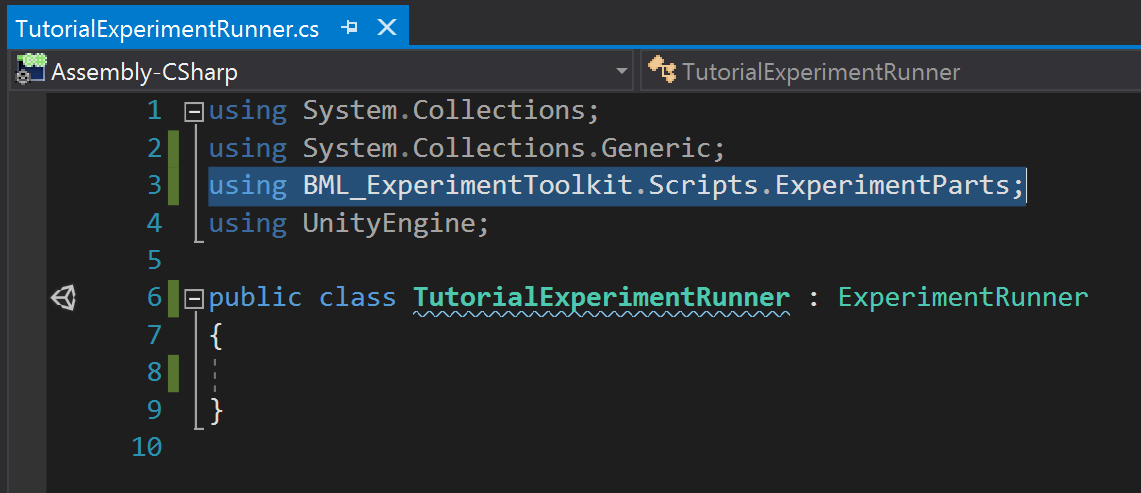
1. Create a folder for your experiment files:
   1. In the project navigator window, right click your assets folder, and select Create > Folder
      1. Name it TutorialExperiment
2. Create a new Unity Scene inside the folder and name it TutorialExperiment
3. Create an Experiment GameObject
   1. In the new scene, create an Empty GameObject, and name it Experiment.
4. This GameObject needs to have a custom ExperimentRunner Script attached to it. We will need to create this script, and then drag it onto our Experiment GameObject. This is the main window of communication between the toolkit and your unity scene.

Create Custom ExperimentRunner for your experiment

1. Inside your TutorialExperiment Folder, right click and create a new C# script
   1. Right click on Assets > TutorialExperiment Folder > Create > C# script
   2. Name it TutorialExperimentRunner.
   3. Double click on the created file
2. The file will load in VisualStudio and unity has already populated it with some code.
   1. Delete the Start and Update methods.
   2. The script automatically inherits from the MonoBehaviour class. However, we want it to inherit from ExperimentRunner.
   3. Replace the word MonoBehaviour with ExperimentRunner.
   4. This makes our script “an ExperimentRunner”, to which we can add custom functionality for our experiment’s needs.
   5. Visual studio should automatically import the correct namespace
   6. If you don’t have VisualStudio set up properly, you may get an error or see ExperimentRunner underlined in red. To fix this we need to import the correct namespace by typing at the top of the file:

using BML\_ExperimentToolkit.Scripts.ExperimentParts;

* 1. Your file should look like this:

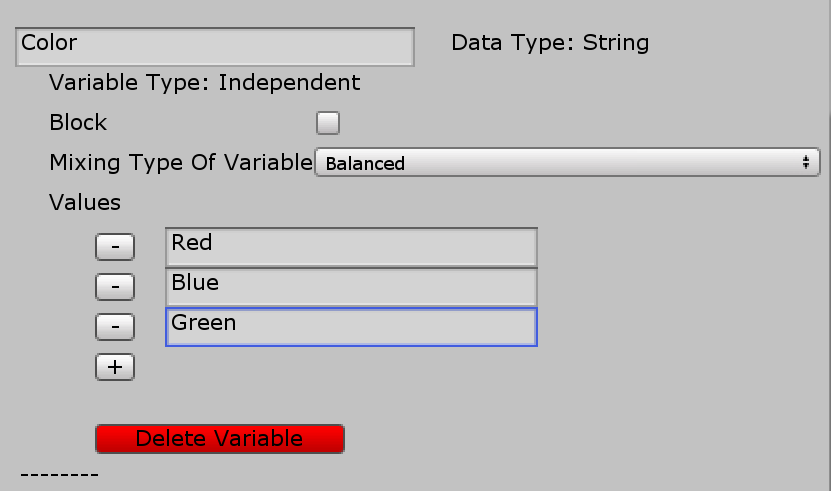


1. Now we can drag this script onto our Experiment GameObject in the scene.
2. Click on the Experiment GameObject. Notice how in the inspector, the script appears, and is automatically asking for a Config File. We need to configure our experimental design, and reference it here.
3. Let’s create our config file

# Configuring the experimental design

The toolkit comes with a powerful system for setting up your experimental design. We need to define it using a config file.

1. Create a new config file. The toolkit adds a custom menu to the asset creation context menu which allows you to create assets necessary to define the functionality and customization for your experiment.
   1. Right click the TutorialExperiment Folder and choose
      1. Create > BML Assets > Variable Configuration
      2. This creates a new asset.
      3. Name it TutorialConfig.
2. Clicking on our config file reveals an inspector with several options. For more detail on what these settings do please see the main documentation. Notice how there are sections for different types of variables. Briefly:
   1. Independent variables: any manipulated variables that change between trials or blocks in your experiment (e.g. which stimuli, presentation time, distance from participant etc.)
   2. Dependent variables: outputs or measurements from your experiment (i.e. response time, selection etc.)
   3. Participant variables: Collected at the start of the session from each participant (e.g. gender, ID, etc.)
3. Let’s set up our experiment’s variables.
4. We want to vary the color of the experiment in each trial. So let’s define it as an independent variable named Color.
   1. In the config file’s inspector, in the Variable Creation section, under type to create, choose “String” to make a text-based data variable.
   2. Select Independent.
   3. Click Create Variable
      1. You should see a new variable appear in the Independent Variables section.
      2. We want to vary color every trial, so keep Block unchecked.
      3. We want an the same number of trials of each color, so choose Balanced for the mixing type.
   4. Define values for our variable. Click the plus button to add a values
      1. Add values Red, Blue, Green.
   5. You should see something like this:



* 1. Let’s create our Distance variable. In this case we want to do all the trials at one distance, and then do all the trials at another distance. So in this case we want to distance to be a blocked variable.
     1. Create a float independent variable.
     2. Name it Distance
     3. Check block
     4. Choose balanced
     5. Add values 1,2,3
  2. Let’s create our size variable.
     1. Create a float independent variable
     2. Name it Size
     3. Uncheck Block
     4. Choose balanced
     5. Add values 1, 1.5, 2.
  3. We want to record the participants gender and age.
     1. Create a String Participant variable named Gender
        1. Check Constrain values. This ensures that the value is restricted to a set of values that we define.
        2. Type in Male, Female, Other
     2. Create an int Participant variable named Age.
        1. Keep Constrain values unchecked so we can type in their age.
  4. We want to record how closely the participants match the size of the stimulus. We’ll record that in a float dependent variable called SelectedSize. The default value will be assigned to any missing values in case of problems or stopping the experiment early.

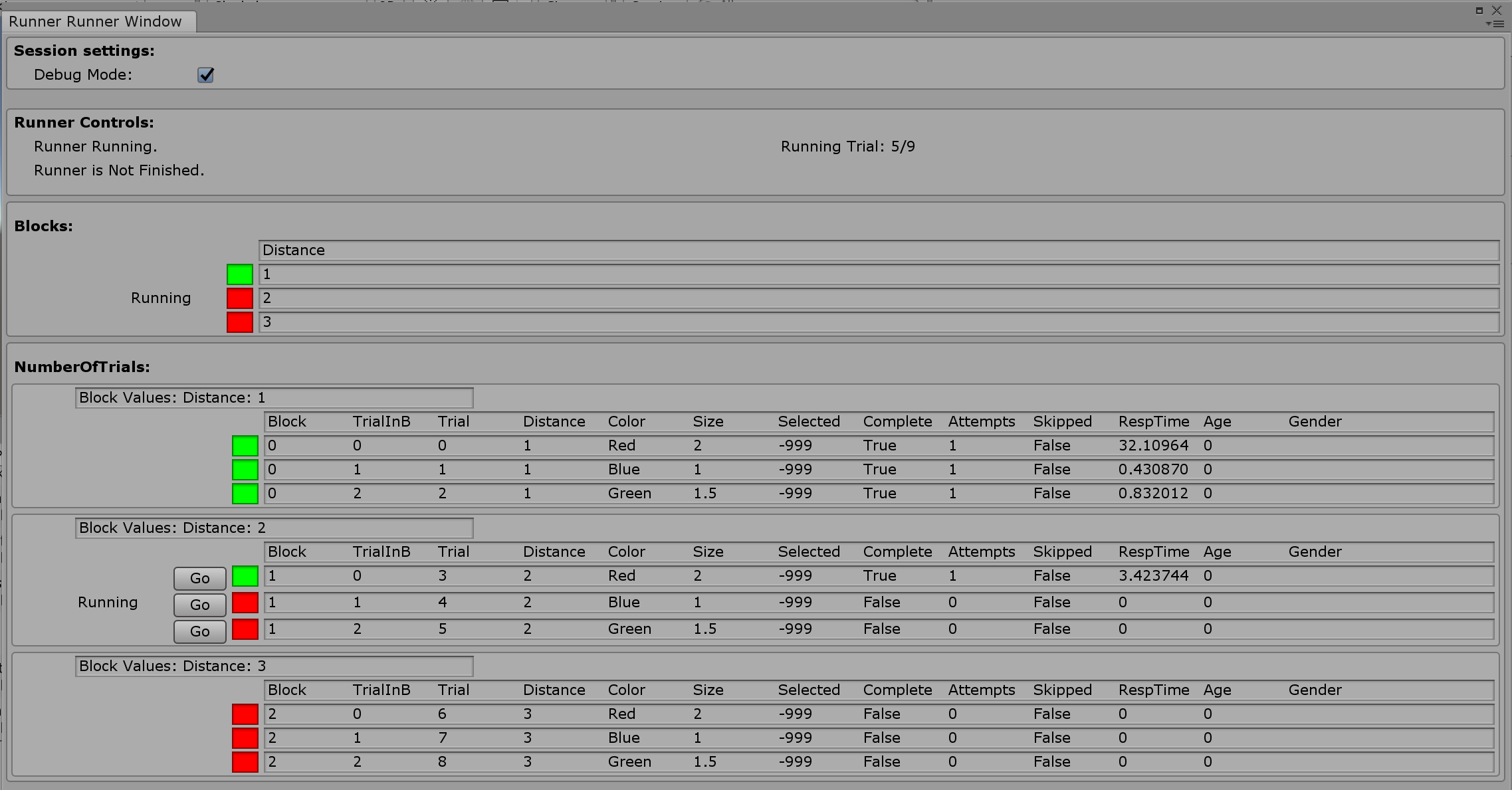
1. Finally, we want to randomize the trial order completely.
   1. In the top of the config file
      1. Check Shuffle Trial Order
      2. Check Shuffle Differently For Each Block
      3. We don’t want to repeat anything so leave Repeat empty.
2. We have to tell our ExperimentRunner GameObject where to find the config file.
   1. Click on the GameObject, and then drag the config file into the appropriate field in the inspector.



# Test out our experiment design using basic components

The BML toolkit comes with premade parts of experiments already defined to let you get up and started quickly. Lets test out if everything is working as expected before moving forward.

1. Open the BML Menu, and Open the Experiment Runner Window.
2. Press play in the editor. You’ll see the window show some controls.
3. Keep it Debug Mode checked. Debug mode is useful to check functionality during development.
4. Click Confirm Order, which selects the first possible order in which to present the blocks. This order is important when counterbalancing block order between participants.
5. Press Start Experiment.
6. Once Start Experiment is pressed, the toolkit automatically constructs your blocks and trials based on how you defined them in the config file and starts running the first trial
7. Take a look at the experiment table and ensure everything is set up properly. Note that the toolkit adds some useful columns to track progress and other metrics.
8. Because it’s debug mode, our participant variables will not work properly yet. We’ll be able to set values for them later when we leave debug mode.



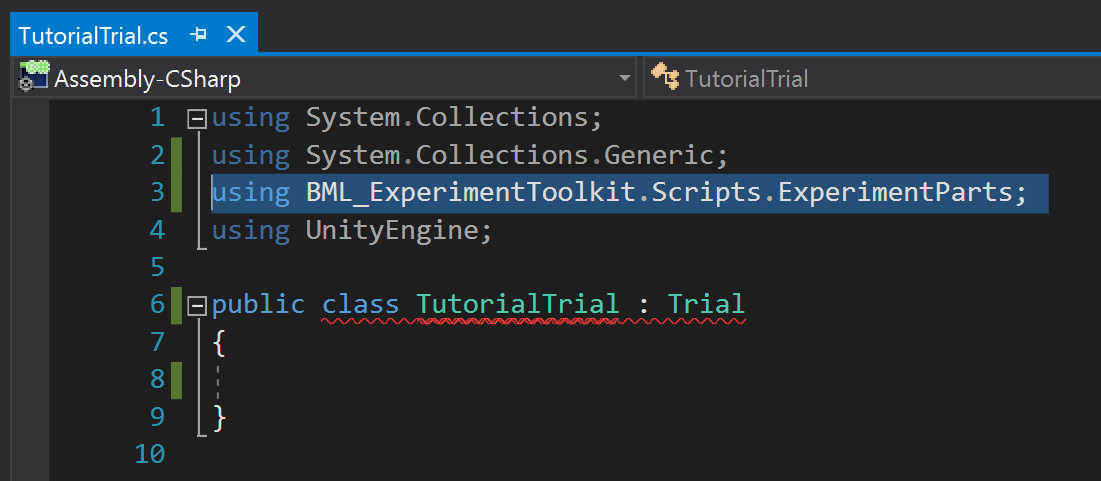
The built-in experiment parts have almost no functionality. A trial is simply defined as pressing the return key. Press it a few times to see how the trials advance, and the values of the independent variables change.

It looks like everything is working properly. Lets move away from the built in experiment parts and define what happens in our experiments’ trial.

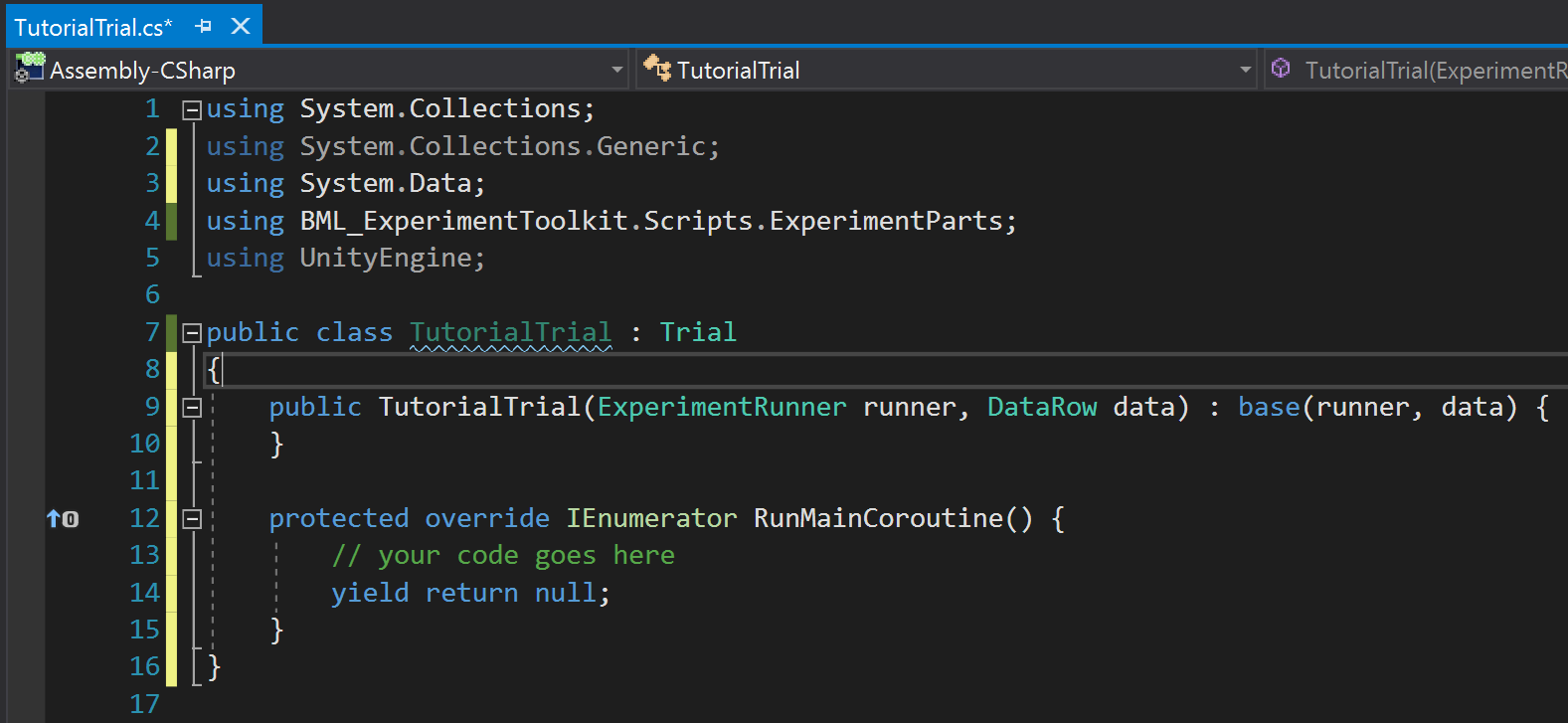
# Define what happens in a trial

Now that we have configured the design of our experiment, and set up our experiment runner object, we have to define what occurs in a trial. We need to create a new custom script that inherits from the Trial Class. In our custom Trial script, we define behavior that occurs during each trial based on the values of the independent variables for that trial. Once we’ve defined the trial script we have to edit our custom ExperimentRunner script to tell it that we’re changing the trial from the built-in basic trial to the custom trial script.

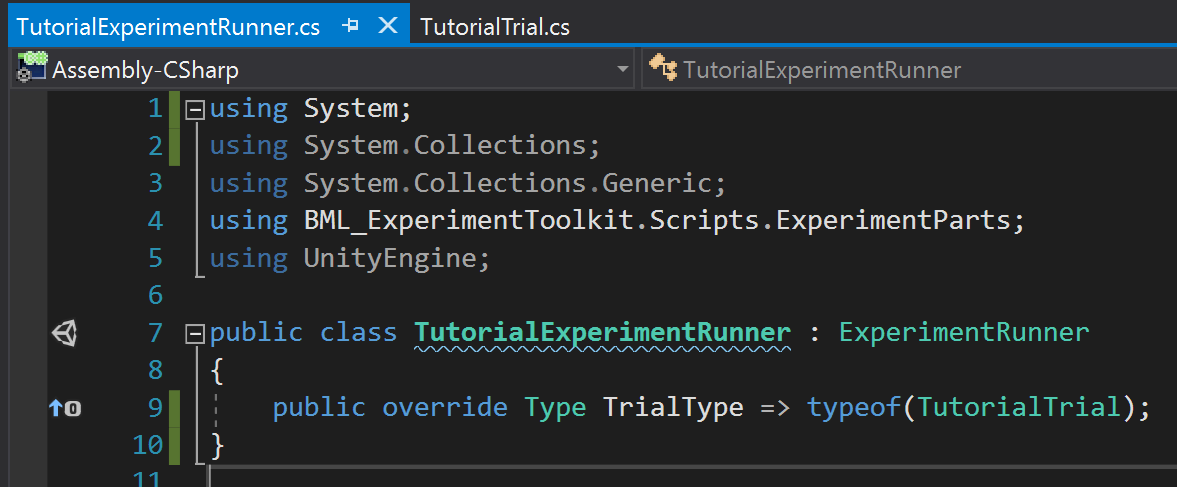
1. Create a new C# script in the TutorialExperiment folder and name it TutorialTrial.
2. Open the script in Visual Studio and delete the Update and Start methods.
3. Like before, we don’t want our script to be a basic MonoBehaviour.
   1. We want our TutorialTrial class to inherit from Trial.
   2. Delete MonoBehaviour, replacing it with Trial.
   3. You may need to import the proper namespace again.
      1. Using BML\_ExperimentToolkit.Scripts.ExperimentParts;



1. Notice how Visual Studio is giving an error underlining our class. This is because the Trial Class forces you to write some methods to make it work with the toolkit. Most editors (including Visual Studio) can solve this easily.
   1. Click on the word TutorialTrial. Visual studio will show a little red lightbulb icon to the left in the margin. Click on the lightbulb and there should be an option similar to “Implement Missing Members”. Select it and some code should appear.
   2. The first part of this is a constructor so that the toolkit can properly create trials when it needs to. This will never need to be changed unless for very advanced customization.
   3. The second part is the important part. This is the main method of our trial in which we define the majority of code that occurs in the trial.
      1. This method is a Coroutine, which means it needs to have at minimum one yield return statement. If you don’t know about coroutines, there is a section in the documentation explaining them.



1. Let’s start simple and just have it print something to the console.
   1. Replace the “// your code goes here” with:
      1. Debug.Log(“Trial Running”);
2. Now each trial will simply output that text, and then immediately move on to the next trial.
3. We have to tell the ExperimentRunner to use this custom trial rather than the built-in one.
   1. Open your TutorialExperimentRunner script.
   2. The ExperimentRunner class has a method called TrialType which we can override to point it to our new custom Trial script.
   3. Start typing the word “override”. After a few letters, Visual Studio will show you a popup with some suggestions. Use the arrow keys to scroll down to public override TrialType and hit return. Visual studio will automatically create this method for you.
   4. We want it to point to our TutorialTrial script.
   5. Edit the created method to read:
      1. public override TrialType => typeof(TutorialTrial);
   6. this means that when ExperimentRunner looks for its TrialType it will get the type of trial of your TutorialTrial.



1. Let’s run our experiment from the runner window again to see if it worked. The Trials should all finish one after another. Look in the console to see if it output the correct text. Make sure all the scripts are saved first!

# Setting up the Stimuli

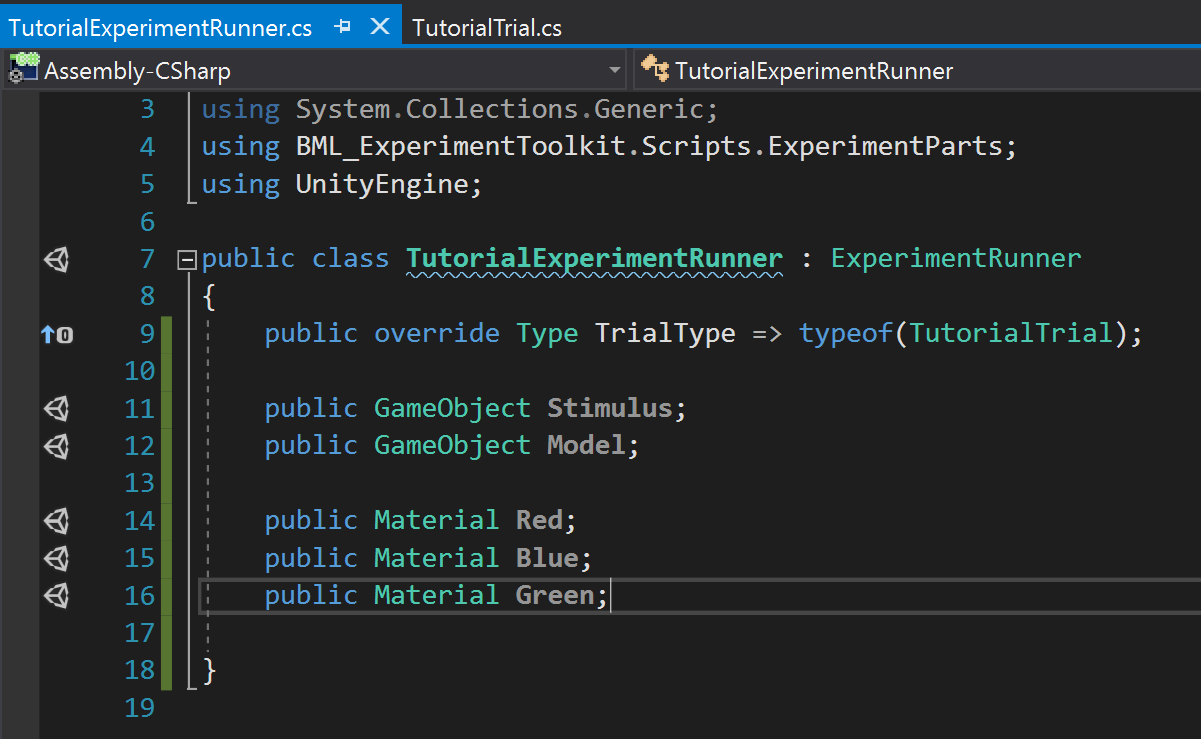
Now that we have our trial structure set up, we need to create some objects to use as stimuli.

1. Let’s first position our Main Camera to 0,0,0, and have it pointing along the Z axis.
   1. Click the Main Camera in the scene and reset its transform component.
2. Let’s create the object that will be the model to which participants are trying to match.
   1. Create a Capsule GameObject, Name it ModelObject
   2. We want the model to be off to the side.
      1. Set its X position to 2
3. Let’s create the stimulus object
   1. Create a 2nd Capsule GameObject, Name it SimulusObject
4. Now we need a materials to change the stimulus color.
   1. To create material Right-click on folder > Create > Material
   2. Create a 3 materials in the experiment folder, call them RedMaterial, BlueMaterial, GreenMaterial
   3. For each material, in its inspector click the little color next to the albedo, and change it to red/blue/green.

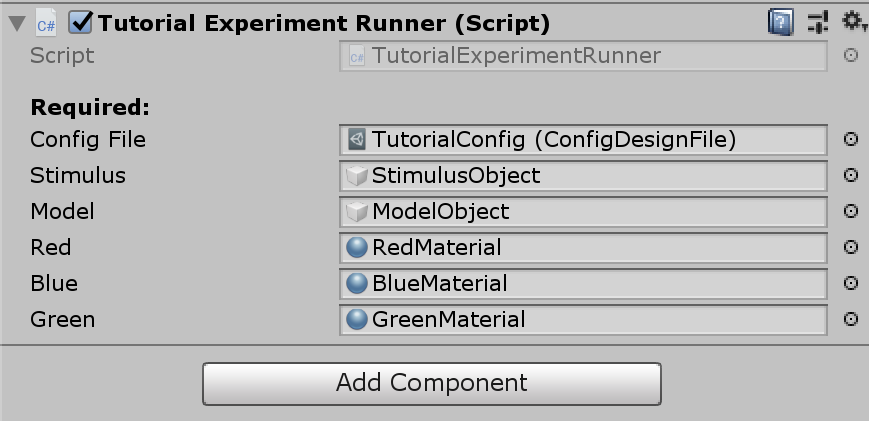
# Referencing GameObjects in your experiment

Now that we’ve set up the objects and materials for our stimuli, we need to be able to reference them in our scripts. We use our experiment runner as our main window to our scene.

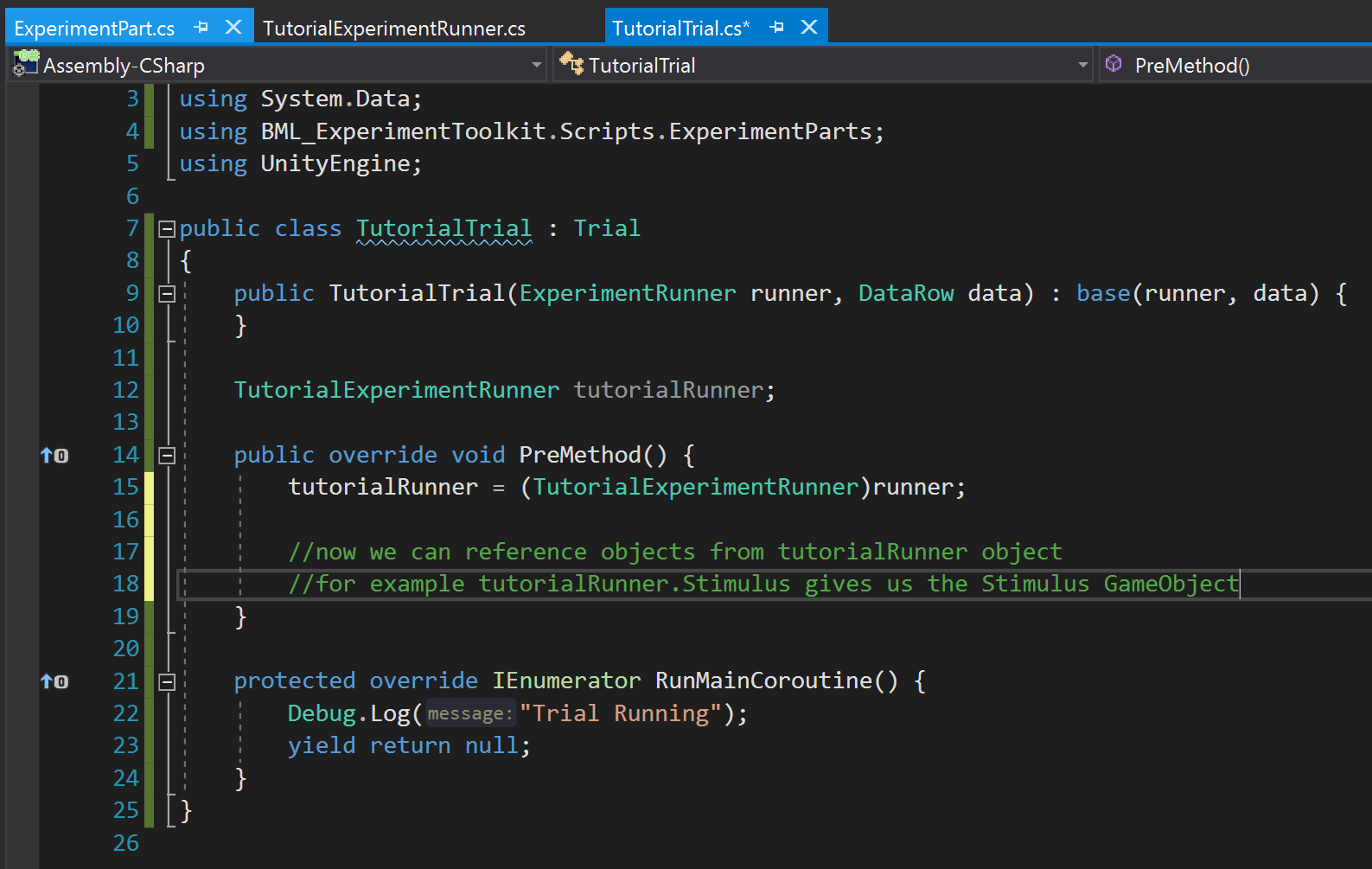
1. Open the TutorialExperimentRunner script.
   1. Create public fields for the two capsules and the 3 materials.
   2. It should look like this:



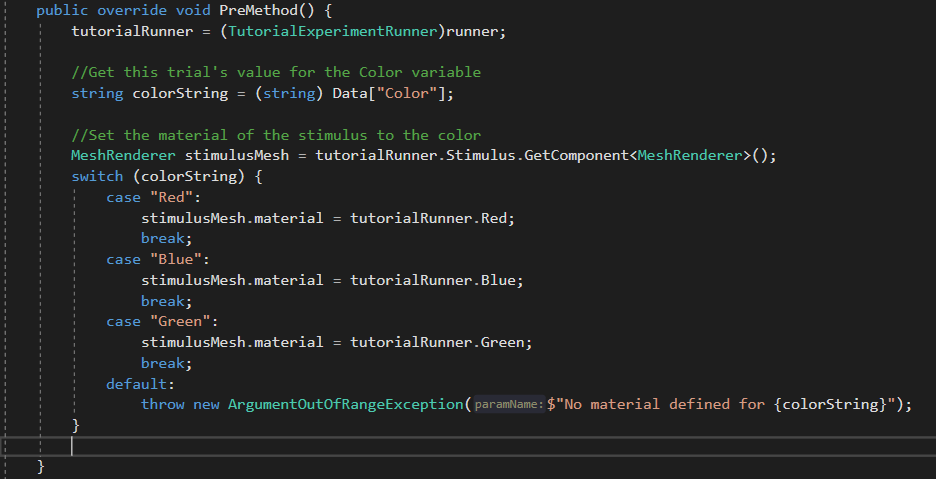
1. In the scene, click on the experiment object,
   1. In the inspector you should see fields now.
   2. Drag the appropriate items into the fields
   3. It should look like this:



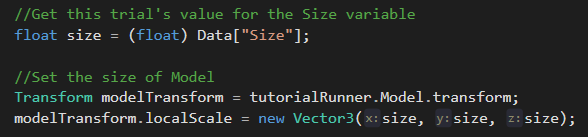
1. Now we need to reference these in our trial script. We could put this code inside our main coroutine method, but this makes more sense in a setup method since it doesn’t happen during the trial, but rather before each trial.
   1. Open the TutorialTrial script.
   2. We want to override the PreMethod method, which is not a coroutine and is used for setting up a trial. It gets automatically called at the start of each trial.
      1. public override void PreMethod() {
   3. Now we need a reference to our custom TutorialExperimentRunner and the objects we defined in it.
   4. Let’s create a field in TutorialTrial to store a reference to it.
   5. The base Trial class already has a reference to it, but it’s not stored as our custom runner but a generic ExperimentRunner. We need to cast it to our custom class TutorialExperimentRunner.
   6. We do the cast in the PreMethod. See the code below.



1. Now we can reference our scene’s objects to set them up. But how do we know what size, distance, and color to use? Our Trial object has a Data object that stores the values for each trial. We access our variables stored in this Data object to set up each trial properly. However, remember we defined distance as a block variable. We’ll set that up with our blocks later.
2. Let’s set the color first.
   1. We want to access the value for our Color variable for each trial:
      1. string colorString = (string)Data[“Color”];
      2. note the cast to a string. This is necessary to remind C# that your Color variable in the config file is actually a string.
      3. Now we want a switch statement to set the stimulus’ material to the correct color. See the code below:



1. Save and run the experiment. You should see it quickly run through the trials and the cylinder flashes different colors.
2. Now let’s adjust the size in the PreMethod as well:



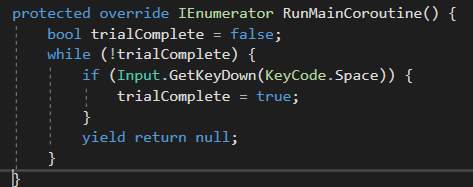
1. Test it again, and you should see the size change as it races through the trials.

# Participant input

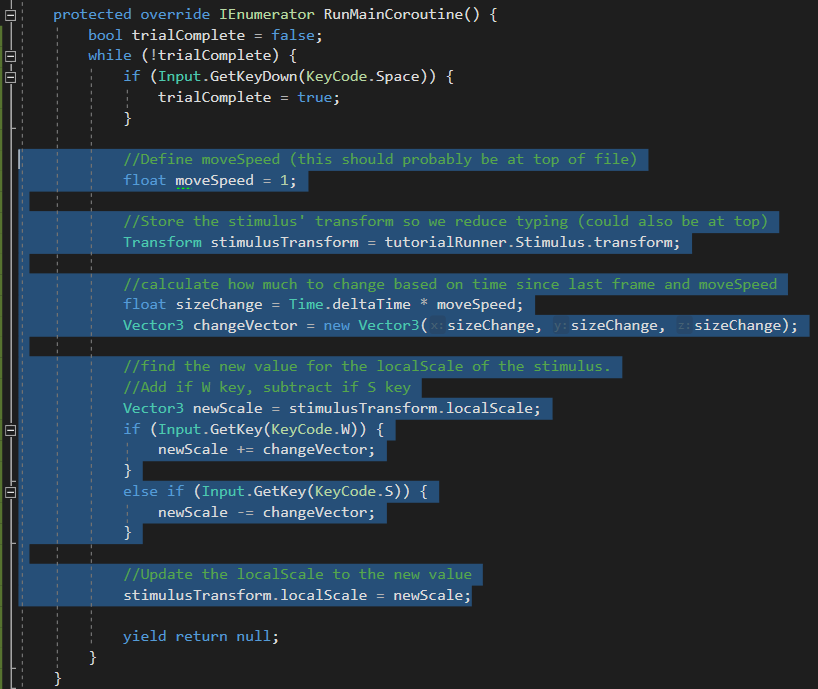
Now that we are properly setting up our trials, we need to allow the participants to resize the stimulus to try to match it to the model. Lets allow participants to resize the stimulus using the w and s keys on the keyboard. This will occur during the RunMainCoroutine Method. We want our script to wait until the trial is done, and listen for those particular keypresses, adjusting the stimulus’ size accordingly. We’ll end our trial using the spacebar.

First lets implement the functionality to keep waiting until the spacebar key is pressed.

1. Create a bool value called trialComplete and set it to false;
2. We want to use a while loop that continues as long as the trial is not complete.
   1. Remember to yield return inside the while loop so it waits to check each frame rather than hanging the program in an infinite loop.
   2. Tip: If you ever get stuck in an infinite loop, press Control-Alt-Delete, open the task manager and close the Unity Editor program.
3. Lets check if the spacebar key is pressed, and if so, set trialComplete to true. This will exit the while loop and finish the coroutine, which allows the program to continue.



1. Test out that it works. Unity only detects keypresses when the “Game window” is the focus in the editor. So make sure to click on the game window if it’s not picking up your spacebar presses.
2. Let’s listen to the w,s keys and increase/decrease the size of the stimulus accordingly
   1. Add if statements for the keys. In this instance we want GetKey instead of GetKeyDown to check if the participant held the key down.
   2. Adjust size of stimulus via its transform.localscale.
      1. We’ll create a value to change the speed of resizing so we can adjust it later
      2. We resize it depending on how much time has passed since last frame. This will keep the speed consistent on slower computers.
      3. We’ll make add the size if W is pressed and reduce the size if S is pressed.
      4. Your code should look like this (new additions highlighted):



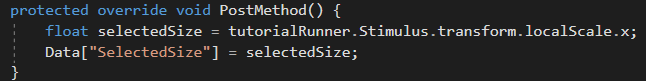
1. Test it out. We now have a pretty functional experiment. We can try to match the size of the colored stimulus to the model using the W and S keys, And when closely matched, we can hit space to move to the next trial.

Although our trial’s are set up well, we still need to output the participant’s matched size.

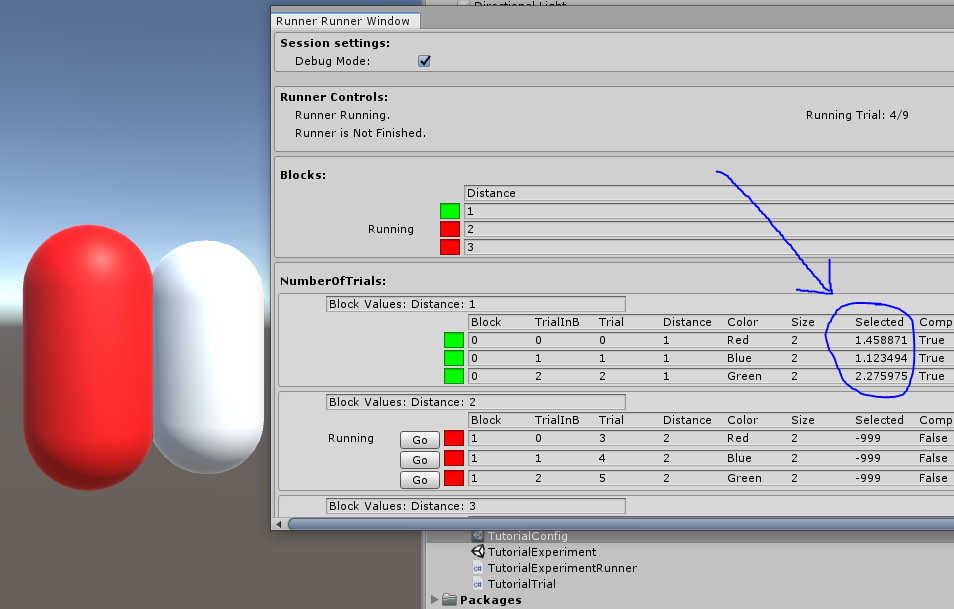
# Outputting Data

Outputting data is similar to reading the values. Recall that we created a variable called SelectedSize. We want to output our value into that variable. We do that after the trial is complete, and since writing won’t span more than one frame we will override the PostMethod() which gets automatically called at the end of each trial.

1. Override PostMethod
2. We need to get and store the size the participant selected.
   1. We can get the stiumulus’ local scale, and read its x value (since y and z will be identical)
3. We write that back to our Data Object;
   1. Data["SelectedSize"] = selectedSize;



1. Run the experiment, and make sure it gets updated in the runner window’s table;

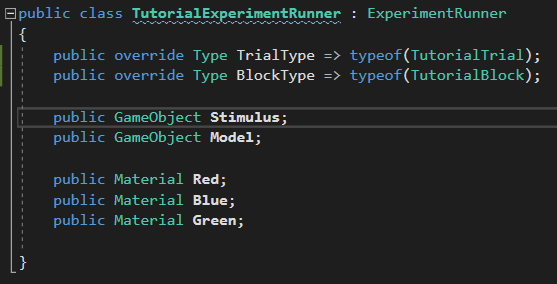


1. When debug mode is checked, the output is saved in a debug file located in your assets folder:
   1. Assets > BML\_Debug > debug.csv.
   2. Open it up and take a look to make sure the output is right.
2. When debug mode is un-checked you can enter in values for the participant variables and save the file to disk for data analysis. Try it out!

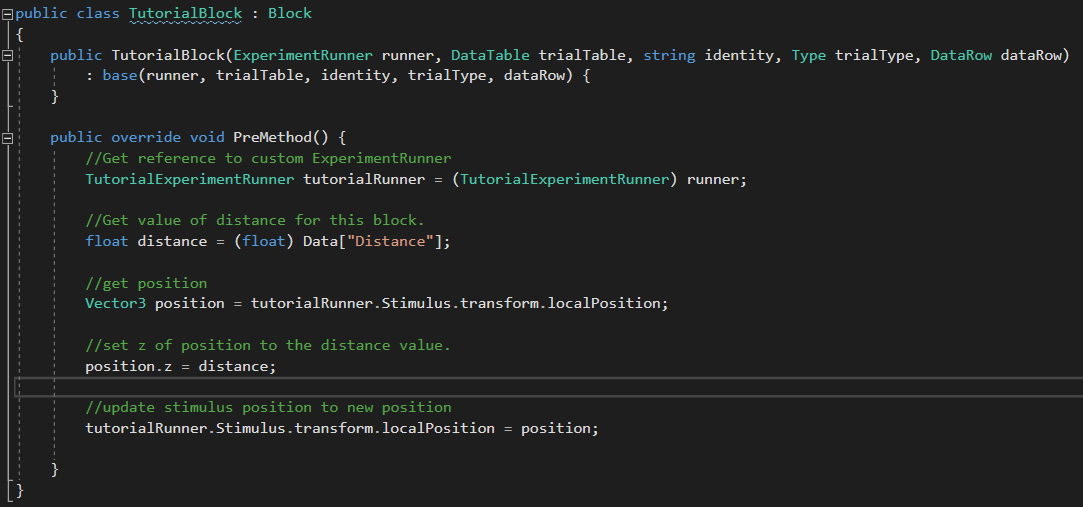
# Defining Blocks

Recall that we set up the distance variable as a block variable. We can define block-specific behavior similar to the way defined our TutorialTrial class. We create a script that inherits from Block. And we can overwrite all the same methods (except for the main coroutine – that’s the trial running).

1. Create a new script, and name it TutorialBlock
2. Open the script
   1. Delete the start and update methods, and make it inherit from Block rather than Monobehaviour.
   2. Remember to use the appropriate namespace:
      1. using BML\_ExperimentToolkit.Scripts.ExperimentParts;
   3. Fix the error by “implementing missing members”. The constructor will be automatically added
3. We need to let our experiment runner know that it should use our custom Block rather than the generic one.
   1. Open TutorialExperimentRunnerScript
      1. Override BlockType to point to TutorialBlock.



1. Go back to TutorialBlock script
2. Override the custom block’s PreMethod() to set up the block
   1. We want to read the Distance variable from the Data object.
      1. float distance = (float) Data["Distance"];
   2. Note: A Block’s Data object only has access to block variables, not variables important only to trials. However, a Trial has access to both block and trial variables.
   3. We want to move the stimulus to the different distances
      1. We have to first get access to our custom experiment runner as before.



1. Verify that it works! Run the experiment and make sure that the distance changes for each block.

# Moving forward

This tutorial got you started making a functional experiment. You learned how to set up a basic experiment, customize trial and block behavior, and output to a file.