CITRIS UC Davis
Visual Thinking Toolkit

Release 1.0.0

Overview

- Facilitates the creation of interactive VR "visual thinking" puzzles
- A set of application assets (e.g. 3D models, textures, scenes, source code)
- Based on the widely used Unity application framework
- Currently supports Oculus Quest VR headset
- Extensible: currently supports three general puzzle templates
- Examples: currently has three example puzzle applications (one based on each puzzle template)

Puzzle Templates

The CITRIS Visual Thinking Toolkit currently supports three general types of "visual thinking" puzzles:

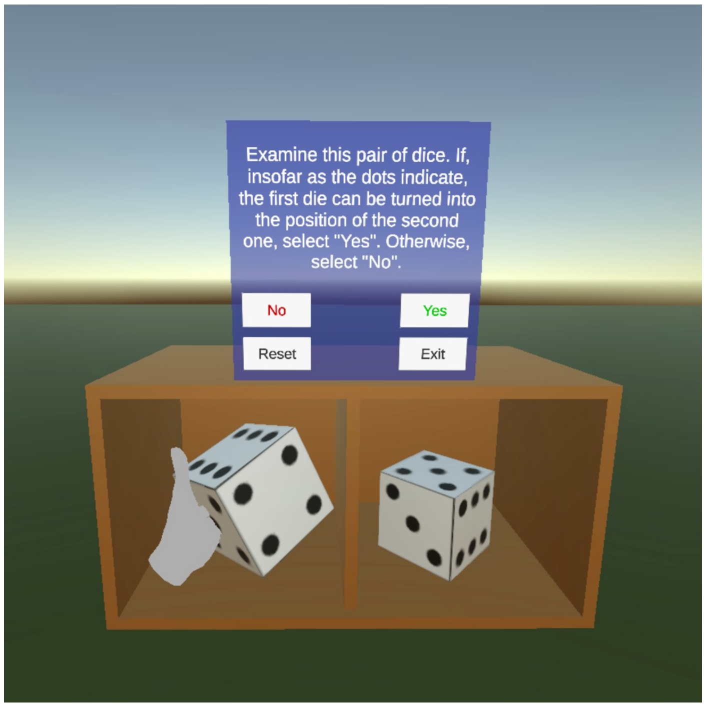
- Related Pair
- Related Objects
- General Puzzle

In order to be accessible to puzzle designers without programming skills, the Related Pair and Related Object puzzle templates do not procedurally (i.e., in code) generate each puzzle. Rather, each puzzle combination can be explicitly specified by the puzzle designer by creating and importing 3D models, then dragging and dropping them into specially created components in the Unity IDE.

Related Pair

The "Related Pair" puzzle template allows for the creation of puzzles which present the player with two interactive objects. The player can view and interact with (pick up, move, rotate) each object. The goal is for the player to determine whether or not the two objects are related or otherwise match in some way (the meaning of this relation depends on the actual puzzle itself). The player can then select "Yes" or "No" to see whether the choice was correct.

An example puzzle application, called "Rotating Dice", is provided which is built on the "Related Pair" puzzle template. This puzzle presents a set of two dice and challenges the player to visually determine whether it's possible, given the three visible sides of each die, for the first die to be rotated in such a way as to appear identical to the second die. If the player has difficulty determining this by visual inspection alone, each die may be picked up, examined, rotated, and placed back down in order to help the player solve the puzzle.

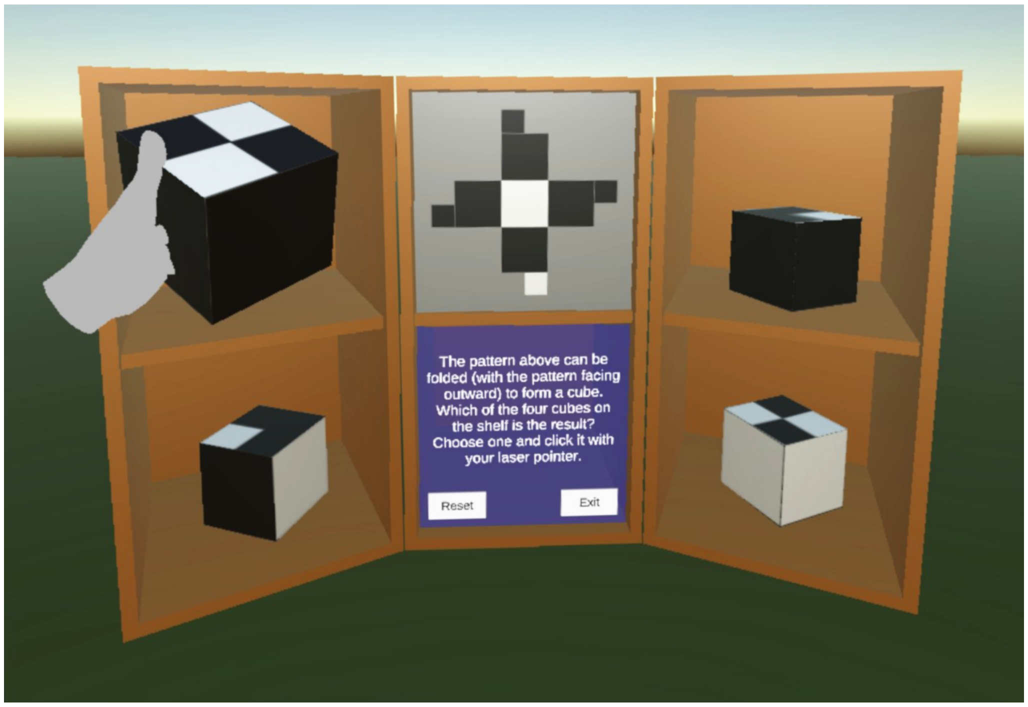

Examine this pair of dice. If, insofar as the dots indicate, the first die can be turned into the position of the second one, select "Yes". Otherwise, select "No".

"Rotating Dice" puzzle, built on the "Related Pair" template

Related Objects

The "Related Objects" puzzle template allows for the creation of puzzles which present the player with one primary object and four other interactive objects. The player can view and interact with (pick up, move, rotate) each of the four objects. The goal is for the player to determine which of the four objects is related or otherwise matches in some way (the meaning of this relation depends on the actual puzzle itself). The player can then select the object by using a VR laser pointer to see whether the choice was correct.

An example puzzle application, called "Folded Pattern" is provided which is built on the "Related Objects" puzzle template. This puzzle presents a flat pattern representing an unfolded box and four folded boxes. It challenges the player to visually determine which of the four folded boxes could possibly be the result of folding the unfolded box pattern such that its white and black coloring faces outward. If the player has difficulty determining this by visual inspection alone, each box may be picked up, examined, rotated, and placed back down in order to help the player solve the puzzle.

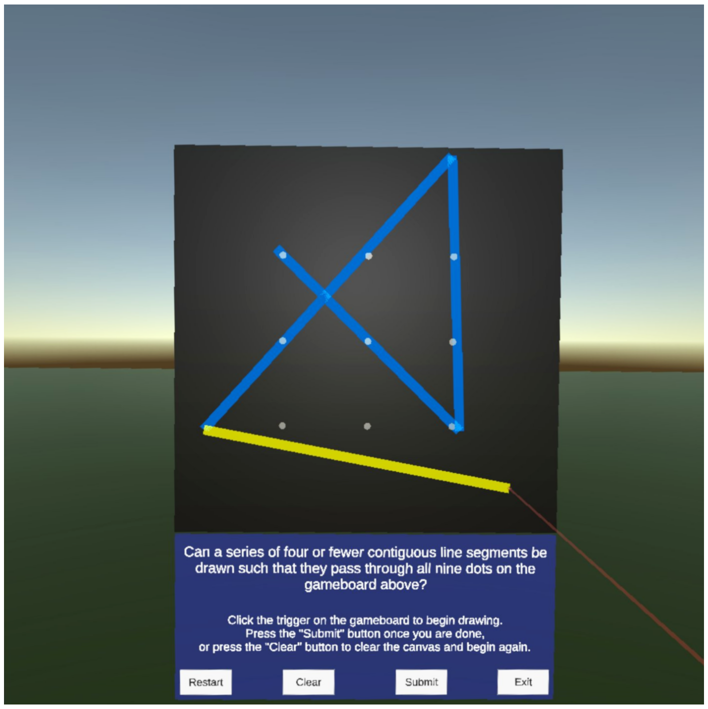

The pattern above can be folded (with the pattern facing outward) to form a cube. Which of the four cubes on the shelf is the result? Choose one and click it with your laser pointer.

"Folded Pattern" puzzle, built on the "Related Objects" template

General Puzzle

The "General Puzzle" puzzle template allows for the creation of a wide variety of custom puzzles. It provides a basic 3D scene with a floor, lighting, camera, and a player with a set of hands and a virtual laser pointer. This puzzle template provides the greatest flexibility for the creation of new puzzles, but it also requires the puzzle designer to have programming skills and a deeper understanding of the Unity framework.

An example puzzle application, called "Nine Dots", is provided which is built on the "General Puzzle" puzzle template. This puzzle presents a set of nine dots in a square grid and instructs the player to draw (with a laser pointer) a series of connected line segments such that all nine dots are covered. At first, the player is challenged to do this with a maximum of four line segments. Next, with a maximum of three. Finally, the player is asked to consider how this feat might be accomplished with only a single line segment.


Can a series of four or fewer contiguous line segments be drawn such that they pass through *all* nine dots on the gameboard above?

Click the trigger on the gameboard to begin drawing.
Press the "Submit" button once you are done,
or press the "Clear" button to clear the canvas and begin again.

"Nine Dots" puzzle, built on the "General Puzzle" template

Toolkit Components and Dependencies

The CITRIS Visual Thinking Toolkit is provided in the form of a Unity "asset package". Unity asset packages are essentially archives of files and associated Unity metadata which can be imported into a Unity project as a set of preconfigured Unity "assets". In Unity, the term "asset" refers to any of a number of types of elements which may be used in a Unity application, such as 3D models, textures, materials, source code, images, audio files, and more.

Once imported into a Unity project, the CITRIS Visual Thinking Toolkit can be found in the project's asset database with the following folder structure[1]:

```
VisualThinking/       :  Toolkit root
  Materials/          :  General toolkit object material definitions
  Prefabs/            :  General toolkit "prefab" object definitions
  PuzzleTemplates/    :  Puzzle templates
    GeneralPuzzle/    :  "General Puzzle" template scene definitions and scripts
    RelatedObjects/   :  "Related Objects" template scene definitions and scripts
    RelatedPair/      :  "Related Pair" template scene definitions and scripts
  Scenes/             :  General toolkit scenes
  Scripts/            :  General toolkit scripts
  Sprites/            :  General toolkit sprite images
```

The toolkit is based on Unity LTS (long-term support) release 2018.4.11f1 and depends upon version 1.43 of the "Oculus Integration" package. It is difficult to acquire earlier versions of the "Oculus Integration" package once new versions are released, so version 1.43 is distributed along with the Visual Thinking toolkit so that it may be easily installed.[2]

The toolkit was developed and tested on an Oculus Quest headset with the following component versions:

```
System Version:   4342600050300000
Version:          12.0.0.190.469.187244423
Runtime Version:  12.0.0.190.469.187244352
OS Version:       user-434260.5030.0
```

Puzzle App Creation

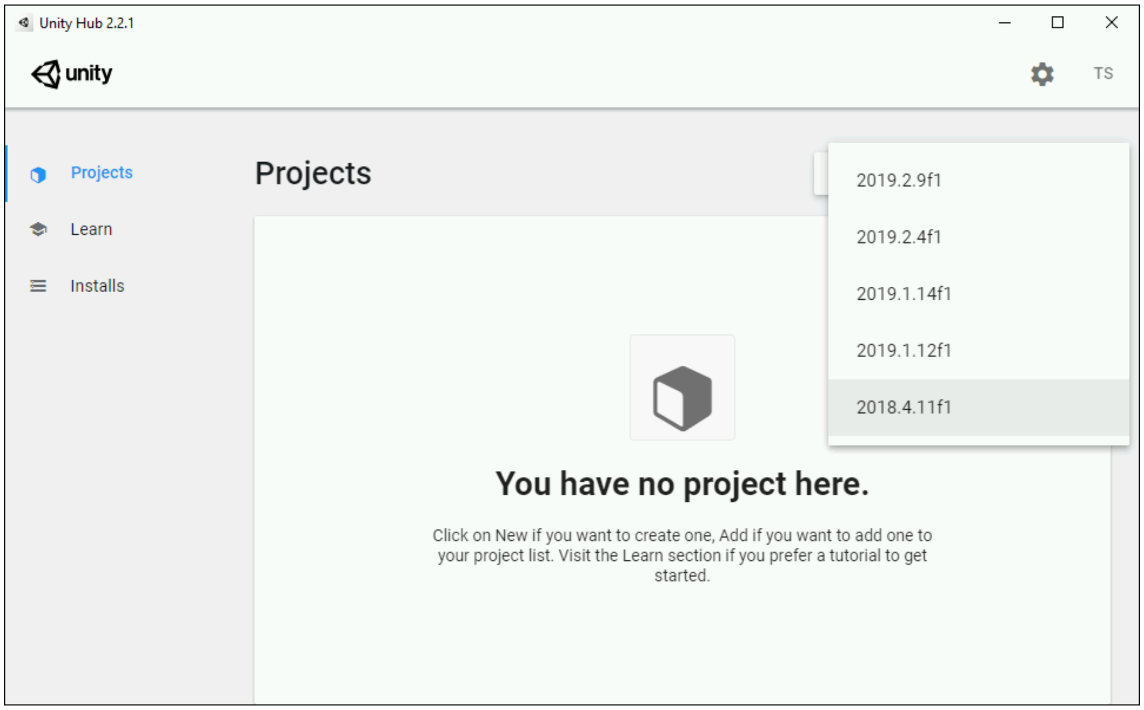
As a guide to understanding how the Visual Thinking toolkit can be used to create an application for the Oculus Quest, we present a high-level walkthrough of the process in this section. The

[1] as of `visthink-1.0.0.unitypackage`
[2] `oculus-integration-1.43.unitypackage`

Unity framework and IDE, the C# language, and general Oculus Quest development are broad topics that are well beyond the scope of this document. However, documentation is very good[3][4][5], and there are numerous other useful resources (official and third-party) available online.

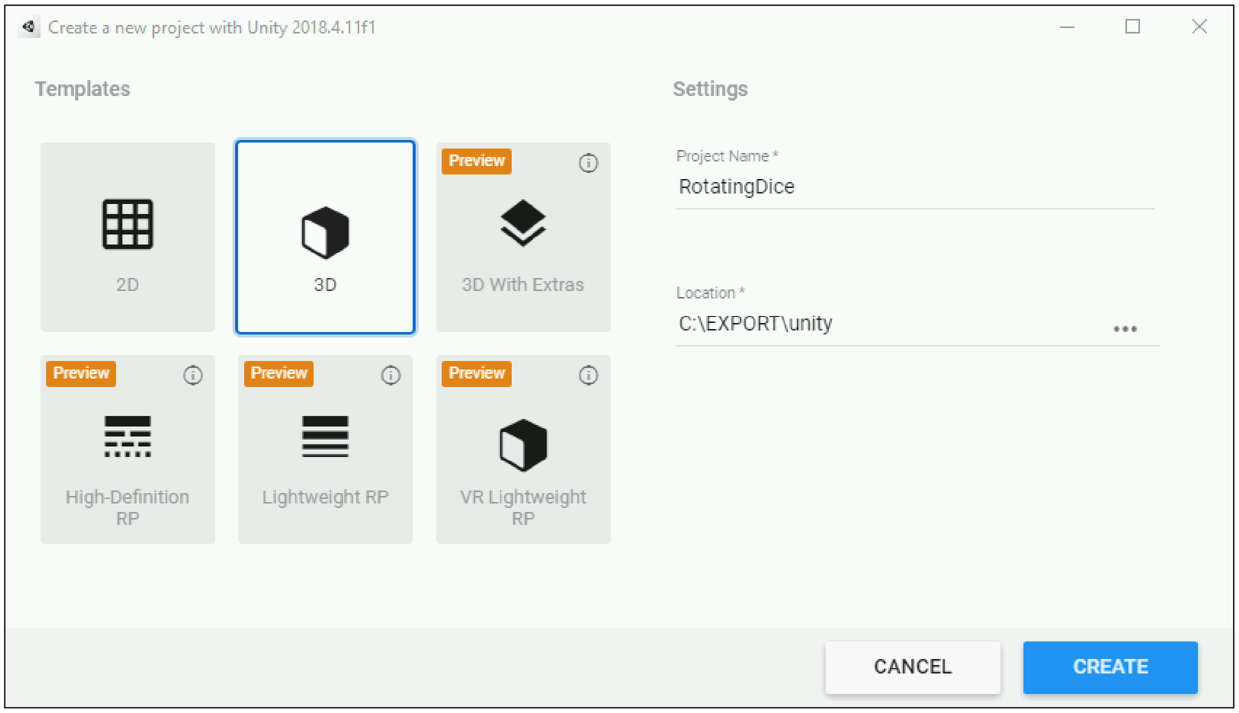
Create Unity Project

To begin, run the Unity Hub application[6] and click the drop-down next to the "NEW" button, selecting "2018.4.11f1"[7]:


Unity Hub 2.2.1
unity
TS
Projects
Learn
Installs
Projects
2019.2.9f1
2019.2.4f1
2019.1.14f1
2019.1.12f1
2018.4.11f1
You have no project here.
Click on New if you want to create one, Add if you want to add one to your project list. Visit the Learn section if you prefer a tutorial to get started.

Select the "3D" template, name the project "RotatingDice", and choose a filesystem location for the project directory:

[3] https://docs.unity3d.com/2018.4/Documentation/Manual/index.html
[4] https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/
[5] https://developer.oculus.com/quest/
[6] These instructions are based on the version of Unity Hub that was current at the time they were written; some steps may have changed in later versions of Unity Hub.
[7] This version may need to be installed from the Unity archives before this selection is available to you.


Create a new project with Unity 2018.4.11f1

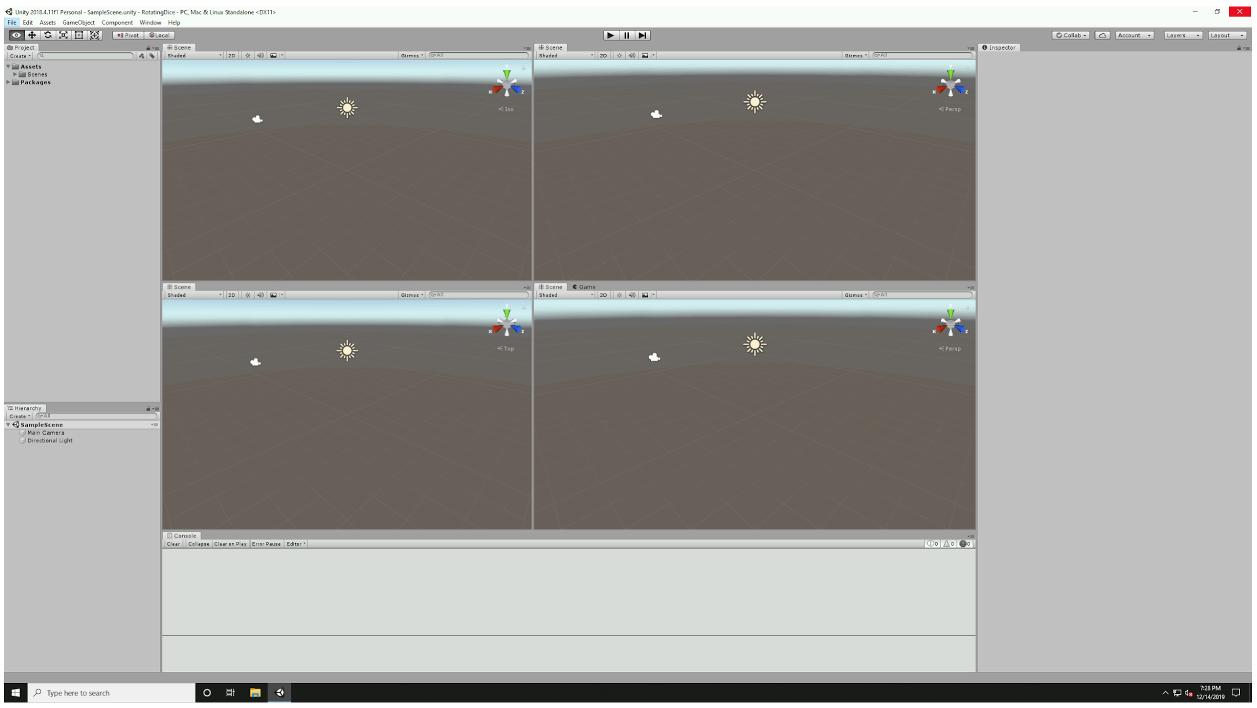
Templates

Settings

Project Name * RotatingDice

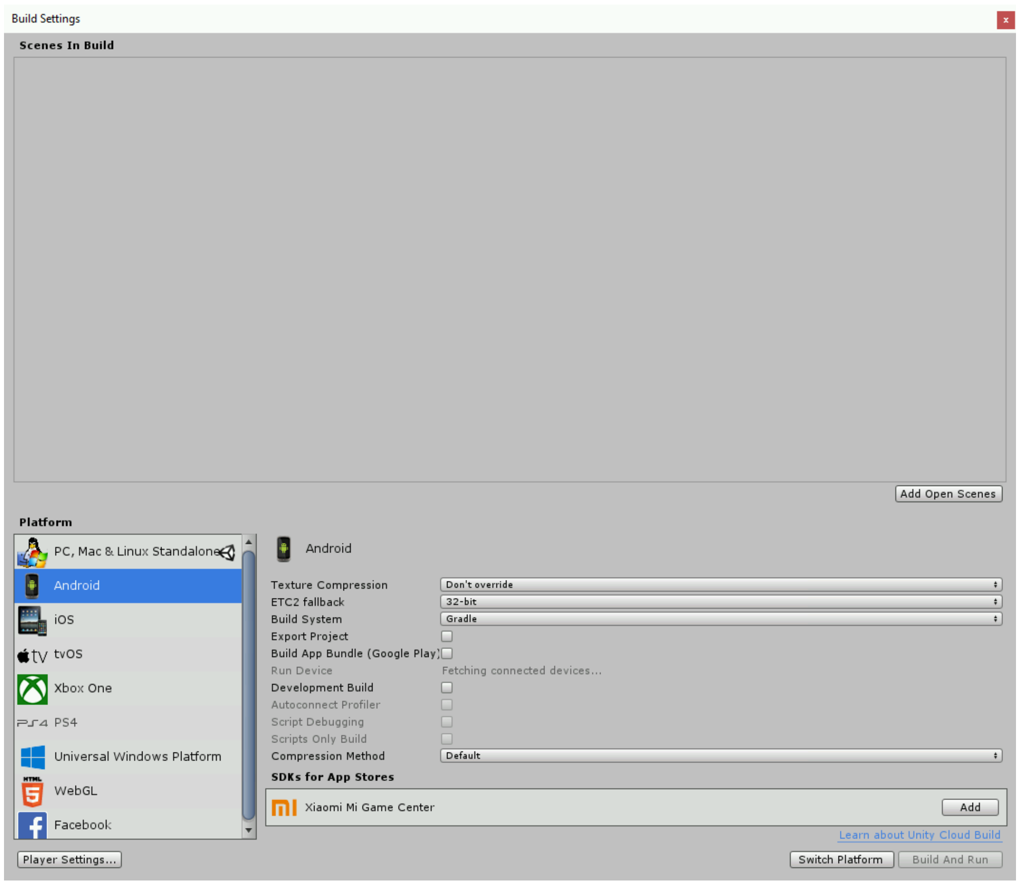
Location * C:\EXPORT\unity

Then, click "CREATE" and the Unity Editor will open the new project:


Unity 2018.4.11f1 Personal - SampleScene.unity - RotatingDice - PC, Mac & Linux Standalone <DX11>

Configure Project

Next, navigate to File → Build Settings. Select the "Android" platform, then click the "Switch Platform" button at the bottom right of the Build Settings window:



Build Settings

Scenes In Build

Platform

PC, Mac & Linux Standalone
Android
iOS
tvOS
Xbox One
PS4
Universal Windows Platform
WebGL
Facebook

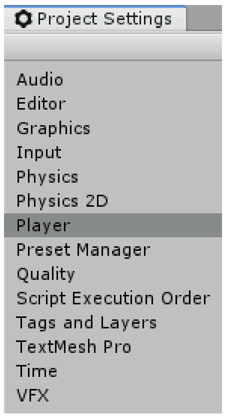
Android

Texture Compression	Don't override
ETC2 fallback	32-bit
Build System	Gradle
Export Project	☐
Build App Bundle (Google Play)	☐
Run Device	Fetching connected devices...
Development Build	☐
Autoconnect Profiler	☐
Script Debugging	☐
Scripts Only Build	☐
Compression Method	Default

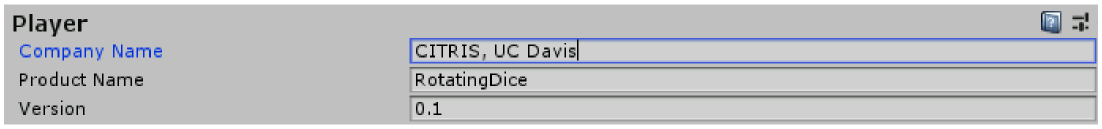
SDKs for App Stores

Xiaomi Mi Game Center

Close the Build Settings window, then navigate to Edit → Project Settings. Select "Player" in the list at the left:



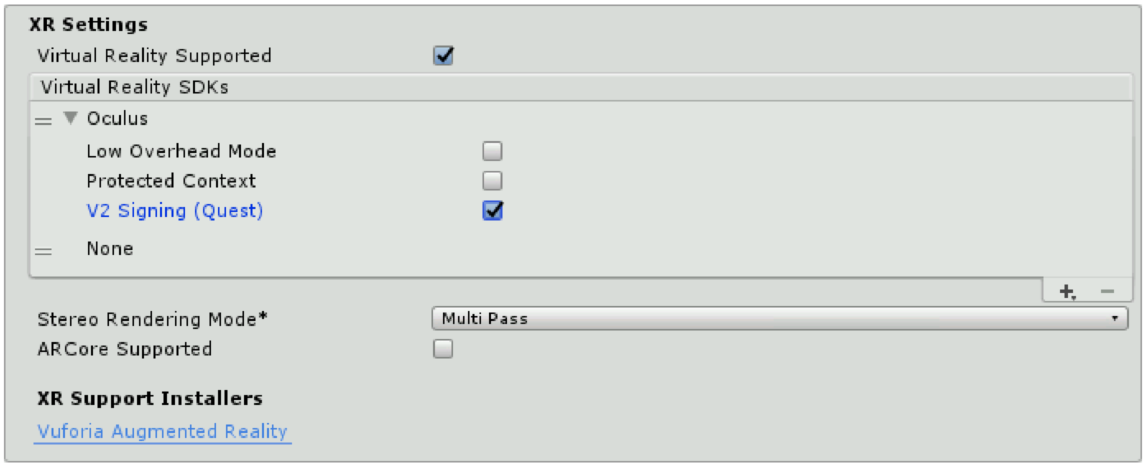
In the "Player" region at the top right of the window, enter "CITRIS, UC Davis" in the "Company Name" field. The "Product Name" field should already be populated with the project name ("RotatingDice"):



Player

Company Name	CITRIS, UC Davis
Product Name	RotatingDice
Version	0.1

In the tab identified with the Android droid logo, open the collapsible "XR Settings" menu. Place a check in the "Virtual Reality Supported" checkbox. Then, in the "Virtual Reality SDKs" region, click the "+" symbol at the bottom right and select "Oculus". Click the "+" symbol one more time and select "None". Then, place a check in the "V2 Signing (Quest)" checkbox under the Oculus section:



XR Settings

Virtual Reality Supported ✓

Virtual Reality SDKs

Oculus

Low Overhead Mode	☐
Protected Context	☐
V2 Signing (Quest)	✓

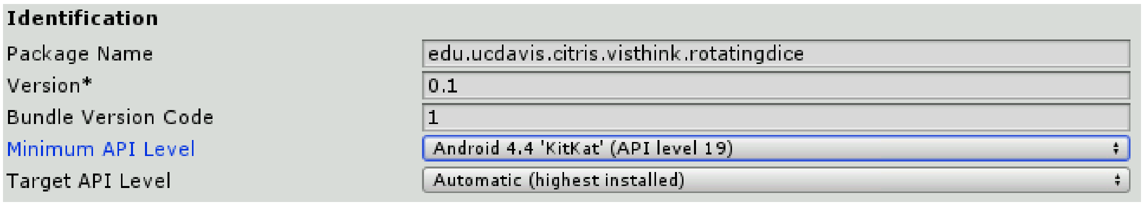
None

Stereo Rendering Mode*	Multi Pass
ARCore Supported	☐

XR Support Installers

Vuforia Augmented Reality

Also in the tab identified with the Android droid logo, open the collapsible "Other Settings" menu. In the "Identification" region, set the "Package Name" field to `edu.ucdavis.citris.visthink.rotatingdice` and set the "Minimum API Level" to "Android 4.4 'KitKat' (API level 19)":



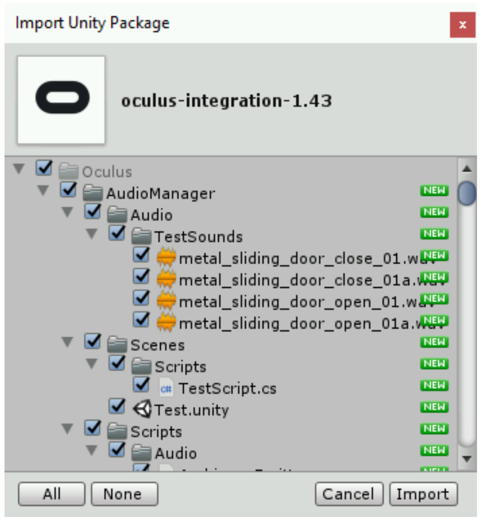
Identification

Package Name	edu.ucdavis.citris.visthink.rotatingdice
Version*	0.1
Bundle Version Code	1
Minimum API Level	Android 4.4 'KitKat' (API level 19)
Target API Level	Automatic (highest installed)

You may now close the Project Settings window.

Import Assets

Now, we will import the Oculus Integration package. Navigate to Assets → Import Package → Custom Package and locate your `oculus-integration-1.43.unitypackage` file in the file dialog. You will see a package import dialog appear:

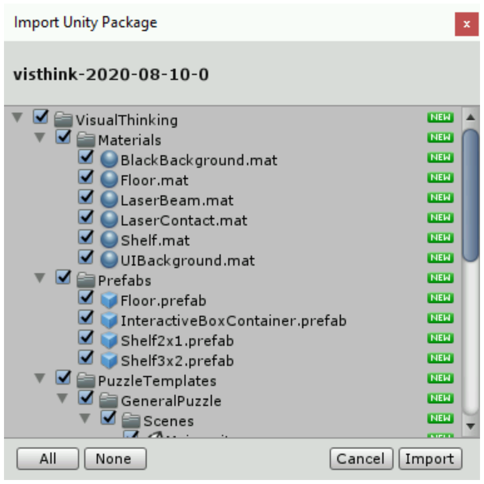

Import Unity Package

oculus-integration-1.43

```
Oculus
  AudioManager                            NEW
    Audio                                 NEW
      TestSounds                          NEW
        metal_sliding_door_close_01.wav   NEW
        metal_sliding_door_close_01a.wav  NEW
        metal_sliding_door_open_01.wav    NEW
        metal_sliding_door_open_01a.wav   NEW
    Scenes                                NEW
      Scripts                             NEW
        TestScript.cs                     NEW
      Test.unity                          NEW
    Scripts                               NEW
      Audio                               NEW
```

Click the "All" button, then click the "Import" button. You may see one or more dialogs appear, asking you to agree to upgrade plugins and/or restart Unity. Agree to these.

Now, we will import the Visual Thinking package. Navigate to Assets → Import Package → Custom Package and locate your `visthink-1.0.0.unitypackage` file in the file dialog. Again, you will see a package import dialog appear:


Import Unity Package

visthink-2020-08-10-0

```
VisualThinking                         NEW
  Materials                            NEW
    BlackBackground.mat                NEW
    Floor.mat                          NEW
    LaserBeam.mat                      NEW
    LaserContact.mat                   NEW
    Shelf.mat                          NEW
    UIBackground.mat                   NEW
  Prefabs                              NEW
    Floor.prefab                       NEW
    InteractiveBoxContainer.prefab     NEW
    Shelf2x1.prefab                    NEW
    Shelf3x2.prefab                    NEW
  PuzzleTemplates                      NEW
    GeneralPuzzle                      NEW
      Scenes                           NEW
```

Again, click the "All" button, then click the "Import" button.

Finally, we will import the 3D models for the application. Navigate to Assets → Import Package → Custom Package and locate your models package in the file dialog. Once more, click the "All" button, then click the "Import" button. If needed, move the models assets to an appropriate place within your project's Assets tree.

The new project's asset database is now ready.

Customize Puzzle

Since our application will be based on the "Related Pair" template, let's make a copy of that template's scene. In the "Project" tab, right-click on `Assets/VisualThinking/PuzzleTemplates/RelatedPair/Scenes/Main` and select "Open".

Then, navigate to File → Save As and, in the file dialog, navigate to the `RotatingDice/Assets/Scenes` directory, and save the scene as `Main.unity`.

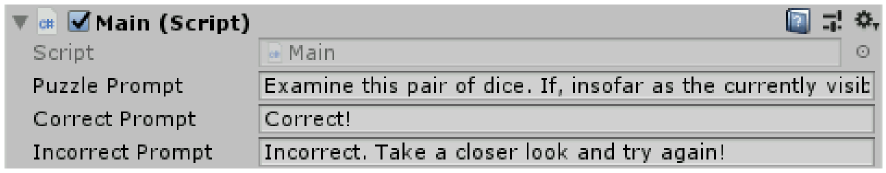
Now that we've made our own copy of the Related Pair scene, let's make sure it's open. In the "Project" tab, right-click on `Assets/Scenes/Main` and select "Open".

When we created the project, Unity generated an example scene. We don't need it, so let's delete it. In the "Project" tab, right-click on `Assets/Scenes/SampleScene` and press the Delete key.

Now, let's begin setting up our puzzle. In the "Hierarchy" tab, click on `Main/MainObject`. The "Inspector" tab will then display various components and attributes belonging to `MainObject`. Among those, under the "Main" script component, you will see three "Prompt" fields:

- Puzzle Prompt: This field will contain the text that will be presented to the player at the beginning of the puzzle.
- Correct Prompt: This field will contain the text that will be presented if the player's choice is correct.
- Incorrect Prompt: This field will contain the text that will be presented if the player's choice is incorrect.

Fill out each field with an appropriate message:



Script	Main
Puzzle Prompt	Examine this pair of dice. If, insofar as the currently visib
Correct Prompt	Correct!
Incorrect Prompt	Incorrect. Take a closer look and try again!

Next, we need to specify the valid puzzle combinations.

In the "Project" tab, navigate the asset database to locate the `Assets/VisualThinking/PuzzleTemplates/RelatedPair/Scripts/Puzzle` script asset and drag it into the open space at the bottom of the "Inspector" tab. Before dragging, take care to ensure that "MainObject" is still selected in the "Hierarchy" tab. If it is not, reselect it and try again. Repeat this process once for each puzzle combination that you'd like to create.

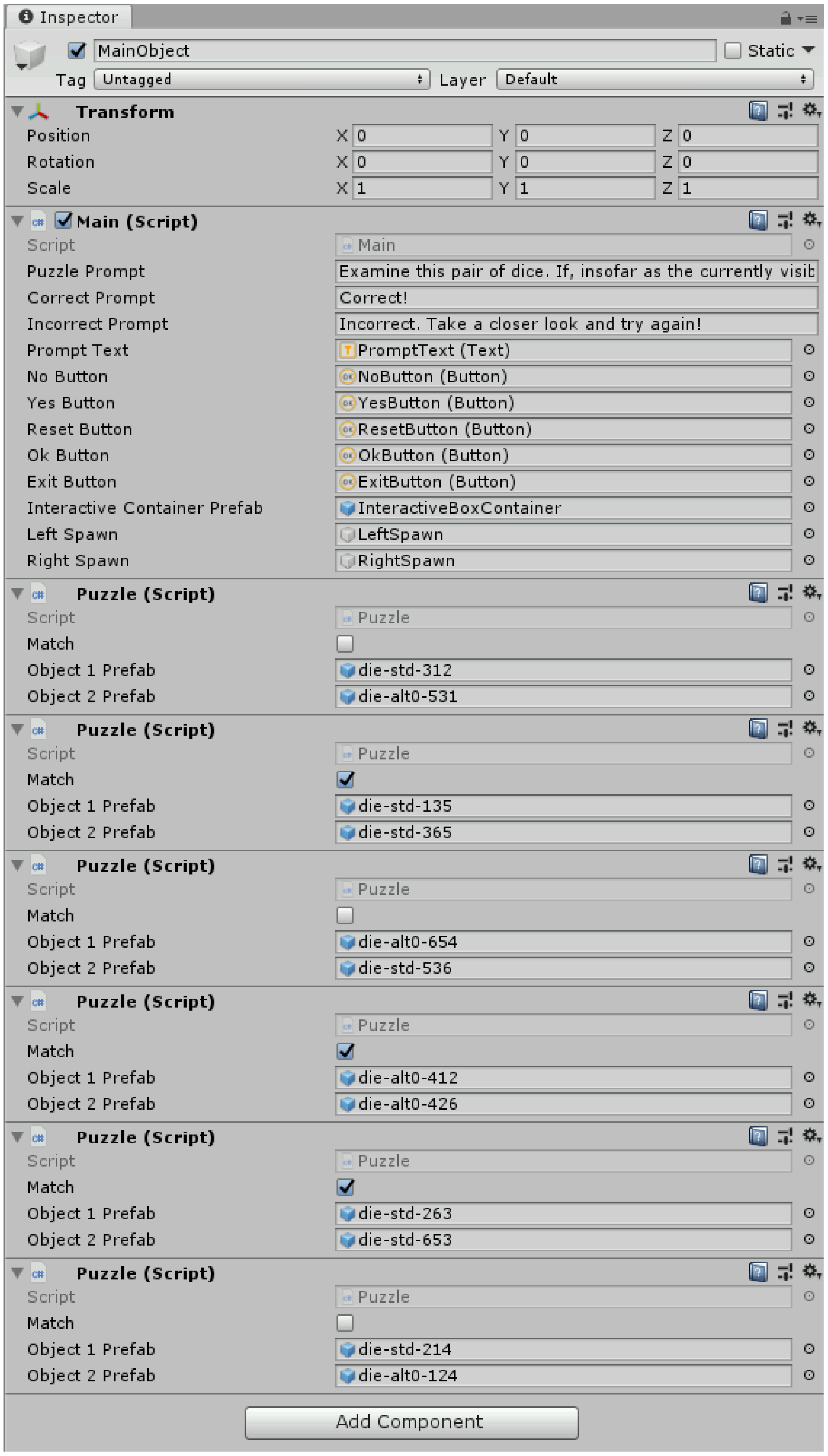
Each "Puzzle" script has three fields:

- Match: a checkbox which specifies whether the two objects in this puzzle are to be considered a "match"
- Object 1 Prefab: the first 3D object for this puzzle
- Object 2 Prefab: the second 3D object for this puzzle

Now, configure each script component that you dragged into the Inspector for "MainObject". The "Match" checkboxes can simply be clicked to select or deselect. For the Object Prefab fields, drag a prefab (from your custom puzzle models that you imported earlier) from the project's Assets tree into the appropriate field in each Puzzle script.

Note that the objects (1 and 2) in each puzzle will not always be presented to the player in left-right order. This is randomized during gameplay. In addition, the puzzles also will not be presented in order; this too is randomized during gameplay.

When you are finished, the "Inspector" tab should look something like the following:



Inspector

☑ MainObject ☐ Static

Tag Untagged Layer Default

Transform

Position	X 0	Y 0	Z 0
Rotation	X 0	Y 0	Z 0
Scale	X 1	Y 1	Z 1

☑ **Main (Script)**

Script	Main
Puzzle Prompt	Examine this pair of dice. If, insofar as the currently visib
Correct Prompt	Correct!
Incorrect Prompt	Incorrect. Take a closer look and try again!
Prompt Text	PromptText (Text)
No Button	NoButton (Button)
Yes Button	YesButton (Button)
Reset Button	ResetButton (Button)
Ok Button	OkButton (Button)
Exit Button	ExitButton (Button)
Interactive Container Prefab	InteractiveBoxContainer
Left Spawn	LeftSpawn
Right Spawn	RightSpawn

Puzzle (Script)

Script	Puzzle
Match	☐
Object 1 Prefab	die-std-312
Object 2 Prefab	die-alt0-531

Puzzle (Script)

Script	Puzzle
Match	✓
Object 1 Prefab	die-std-135
Object 2 Prefab	die-std-365

Puzzle (Script)

Script	Puzzle
Match	☐
Object 1 Prefab	die-alt0-654
Object 2 Prefab	die-std-536

Puzzle (Script)

Script	Puzzle
Match	✓
Object 1 Prefab	die-alt0-412
Object 2 Prefab	die-alt0-426

Puzzle (Script)

Script	Puzzle
Match	✓
Object 1 Prefab	die-std-263
Object 2 Prefab	die-std-653

Puzzle (Script)

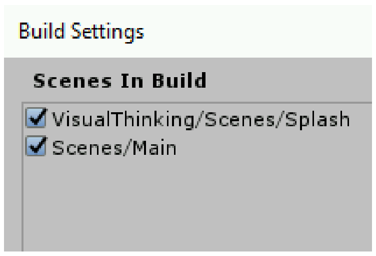
Script	Puzzle
Match	☐
Object 1 Prefab	die-std-214
Object 2 Prefab	die-alt0-124

Final Steps

At this point, we are almost ready to build the application, but we need to do one final thing in order to work around an Oculus bug (that would result in the player's hands being invisible). In the Unity menu bar, navigate to Oculus → Avatars → Edit Settings and enter a dummy Oculus app ID (e.g. `dummy123`) into the "Oculus Go/Quest or Gear VR" field.

We'd also like this application to start with a CITRUS splash screen, so let's copy that in from the VisualThinking assets. First, navigate to File → Save to save the current scene. Next, right-click on `Assets/VisualThinking/Scenes/Splash` and select "Open". Then, navigate to File → Save As and, in the file dialog, navigate to the `RotatingDice/Assets/Scenes` directory, and save the scene as `Splash.unity`.

Finally, we need to specify which scenes will be included in our build, and which scene is the initial scene. Navigate to File → Build Settings. The "Scenes In Build" list at the top of the Build Settings window needs to have `Scenes/Splash` and `Scenes/Main` scenes in it. It should be empty but, if any other scenes are in the list, select them and press the Delete key to delete them from the build. Now, drag `Assets/Scenes/Splash` into the "Scenes in Build" list. Then drag `Assets/Scenes/Main` into the "Scenes in Build" list, ensuring that the `Splash` scene remains at the top of the list. Make sure that the checkbox for each scene is checked. When you're done, it should look like this:


Build Settings

Scenes In Build

✓ VisualThinking/Scenes/Splash
✓ Scenes/Main

Build the Application

Now we're ready to build. click the "Build" button at the bottom of the window. A file dialog will appear. We'll be building an APK (Android package) file, and this dialog is to specify the file's name (e.g. `RotatingDice.apk`) and the directory you'd like it to be written to (the base directory of your Unity project is a reasonable choice). Once you've specified these and clicked the "Save" button, the build will begin. The first build may take a minute or two but, if all goes well, another file dialog should appear showing the newly created APK file in its directory.