Word Game Documentation

Welcome to the Word Game documentation. Below you will find a brief overview of the important scripts used in the game as well as instructions on how to create new board data files. The most important scripts are GameManager, LetterBoard, WordGrid, and WordBoardCreator.

Generating Board Data Files

Board data files are CSV files that contain all the required information for setting up a board in the game. It contains information such as what words are in the board, the positions of each of the letters, etc. They are generated using the Board File Creator window. Below are the steps needed to generate new board data files.

Steps:

1. Add all the categories and levels you want in the GameManagers Category Infos section and all the daily puzzles. The **Name** file in each category will be used as the file names for the board files. This field cannot contain special characters.
2. Open the Board File Creator window by navigating to **Window -> Board File Creator** or by clicking the **Open Board File Creator Window** button in the GameManagers inspector.
3. The GameManager will automatically be set if one exists in the currently open scene.
4. Click the Create Board Files button.

Depending on how many levels you have and the size of each of them this process could take a while.

NOTE: If you change the words in a level then you will need to delete the corresponding board data file in Resources. (Or you could check the **Re-Create All** box which will delete all the old boards and re-create them).

WordBoardCreator

This is the script that handles taking a list of string words and creating a scrambled board where all the words can be selected. Currently it is only used by the Board File Creator window which creates a GameObject and adds the WordBoardCreator as a component during board generation. There is one public method that can be called to start generating a board:

StartCreatingBoard(string id, string[] words, OnBoardFinished callback)

Since generating a board could take up to a couple seconds, WordBoardCreator will start a new thread to generate to board in the background. Once the board is complete the OnBoardFinished callback will be invoked with the finished WordBoard object.

Board generation can also be canceled by calling the public method **AbortBoardCreation(string id)**, pass it the id you use in StartCreatingBoard.

GameManager

The GameManager handles most of the game logic as well as communicating to all the other game components. It handles keeping track of what level is currently being played, the state of the game board (eg. what words have been found), saving and loading game data, loading board data files, setting up GameBoard and WordGrid.

Inspector Properties:

Enable Ads	Check this if you would like Unity Ads to appear between levels.
Zone Id	The zone ID to use when displaying Unity Ads.
Ad Level Complete Amount	The amount of levels that need to be completed before an Ad will appear.
Starting Hints	The number of hints that a player gets when they first start the game.
Letter Board	The GameObject from the Hierarchy that has the LetterBoard component attached to it.
Word Grid	The GameObject from the Hierarchy that has the WordGrid component attached to it.
Scene UI Manager	The GameObject form the Hierarchy that has the SceneUIManager component attached to it.
Letter Tile Prefab	The prefab from the Project folder that has the LetterTile component attached to it.
Category Infos	This is the list of all the levels that can be played in the game which are group by categories.
Daily Puzzles	The list of levels to be used as daily puzzles. One of these levels will be randomly chosen each day.
Daily Puzzle Icon	The sprite that will appear on the game screens top bar while the daily puzzle is being played.

LetterBoard

LetterBoard is responsible for taking a BoardState object and creating a "letter board" out of it. A letter board is the letters that appear on the screen that the player needs to select to try and find words. It keeps track of what letter tiles are being selected and when the player "lets go" it checks if what they selected is a word on the board.

There are two events that cant be listened to:

OnSelectedWordChanged : LetterBoard will invoke this event every time the selected word changes, it passes the newly selected word as an argument. Currently it's being used by SceneUIManager to update what word is selected in the UI.

OnWordFound : LetterBoard will invoke this event when a word is found by the player. It passes the string word that was found, a List of LetterTiles which are the tiles that make up the word, and a boolean which is true if all the words on the board have been found, false otherwise. Currently this is being used by GameManager to update the BoardState and notify WordGrid so it can place the word in its grid.

Inspector Properties:

UI Canvas	The Canvas that letterTileContainer is in. This is used to get the size of the tiles in relation to the actual screen size so we can tell when the mouse is over a tile.
Letter Tile Container	The GridLayoutGroup that each of the tiles will be added to.
Tile Spacing	The amount of world space to place in-between each of the letter tiles on the board.
Tile Touch Offset	Used to control how much of the tile can be touched. 0 means all of the tile can be touched, 0.5 means only half the tile, etc. This makes it so when the player is dragging their finger across the board to select other tiles they don't accidentally select tiles they didn't mean to.
Enable Line	Used to enable/disable the line that will appear when selecting words.
Line Segment Prefab	The prefab to instantiate and use as the line between selected tiles.
Line End Prefab	The prefab to instantiate and use at the end of the line.
Line Container	The GameObject to use as the parent for the line segments and line ends.

WordGrid

WordGrid is responsible for taking a BoardState object and creating those little squares under the game board. When a word is found, GameManager will call FoundWord in WordGrid passing the word and list of tiles from the LetterBoard. The WordGrid will then animate those tiles from the LetterBoard to the proper place in the WordGrid. WordGrid also handles figuring out what letter to show when a hint is to be displayed.

Inspector Properties:

Tile Prefab	The prefab that will be instantiate and used as an empty placeholder for each letter in a word.
Tile Container	The container that each tile will be placed in. Should be a RectTransform with the desired width, height of 0, and placed where the center of the grid should go.
Animation Container	The container that will be used for animating the letter tiles from the LetterBoard to their place on the letter grid.
Tile Size	The size of the each of the tiles in the grid.
Space Between Tile Letters	The amount of space between each of the tiles.
Space Between Words	The amount of space between each group of tiles that make up a word.
Space Between Rows	The amount of space between each row of tiles.

UIScreenController

The UIScreenController handles transitioning between screens. You can make a new screen by creating a script that extends UIScreen and adding it in the UIScreenControllers inspector. Each UIScreen has a unique id set in its inspector that is used when you want to display a screen. UIScreenController handles keeping track of the current screen and hiding it when a new screen is to be displayed. You can display a screen by calling the following method:

UIScreenController.Instance.Show(string id)

There are a number of optional arguments you can pass the Show method:

fromLeft - Specifies which side the screen will animate in from. If its true it will slide in from the left, if false it will side in from the right.

animate - If this is true the the screen will animate in, if its false then the screen will instantly snap into place.

overlay - Specifies if the screen should be shown an an overlay. An overlay is a screen that will slide in over top of the currently displayed screen (Instead of having the currently displayed screen slide out). It must be hidden using the **HideOverlay** method (Calling the Show method will not automatically hide the overlay, it will hide the screen that is “under” the overlay).

style - Specifies the style of the animation.

onTweenFinished - A callback that will be invoked once the animation is completed.

data - Some data to pass to the screen that is being shown. The data will be passed to the UIScreens **OnShowing** method.

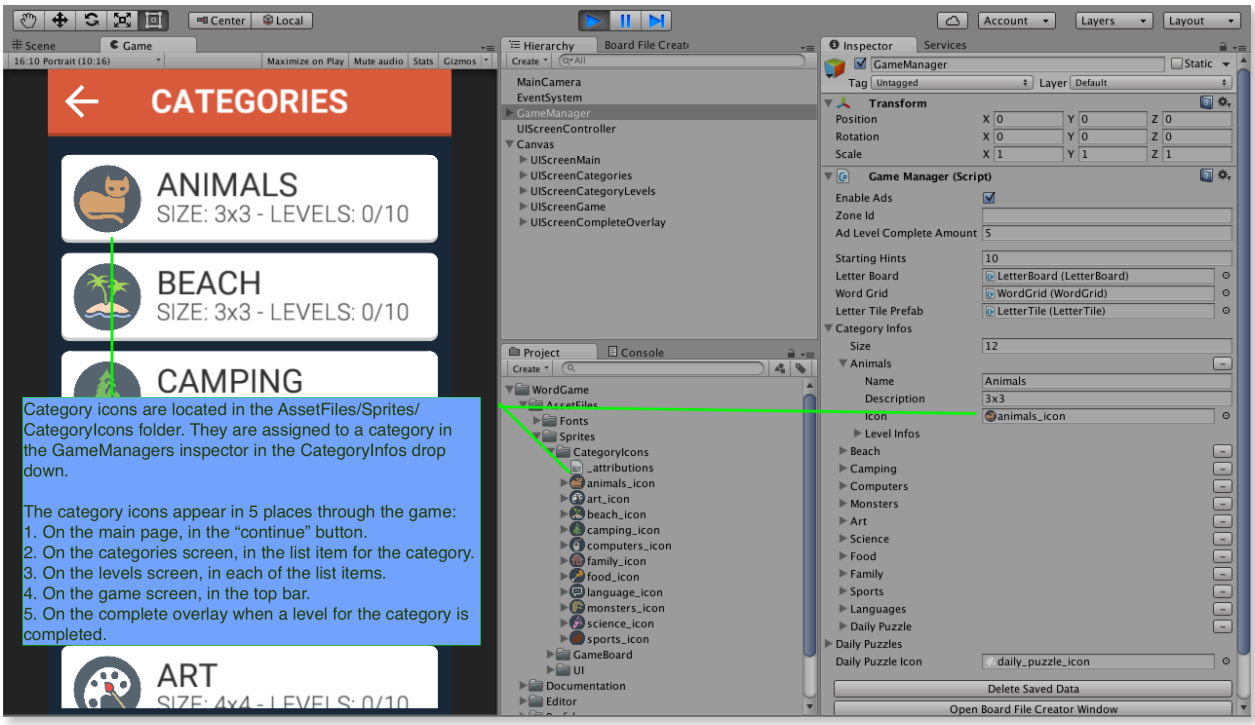
UIScreen

The script that is used by UIScreenController to display / transition screens. Each UIScreen must have a unique **id** set int the inspector. The method **OnShowing** is called when the screen is about be be displayed, this is where any UI for the screen should be set / updated. the **data** is id the object that way passed to UIScreenControllers Show method.

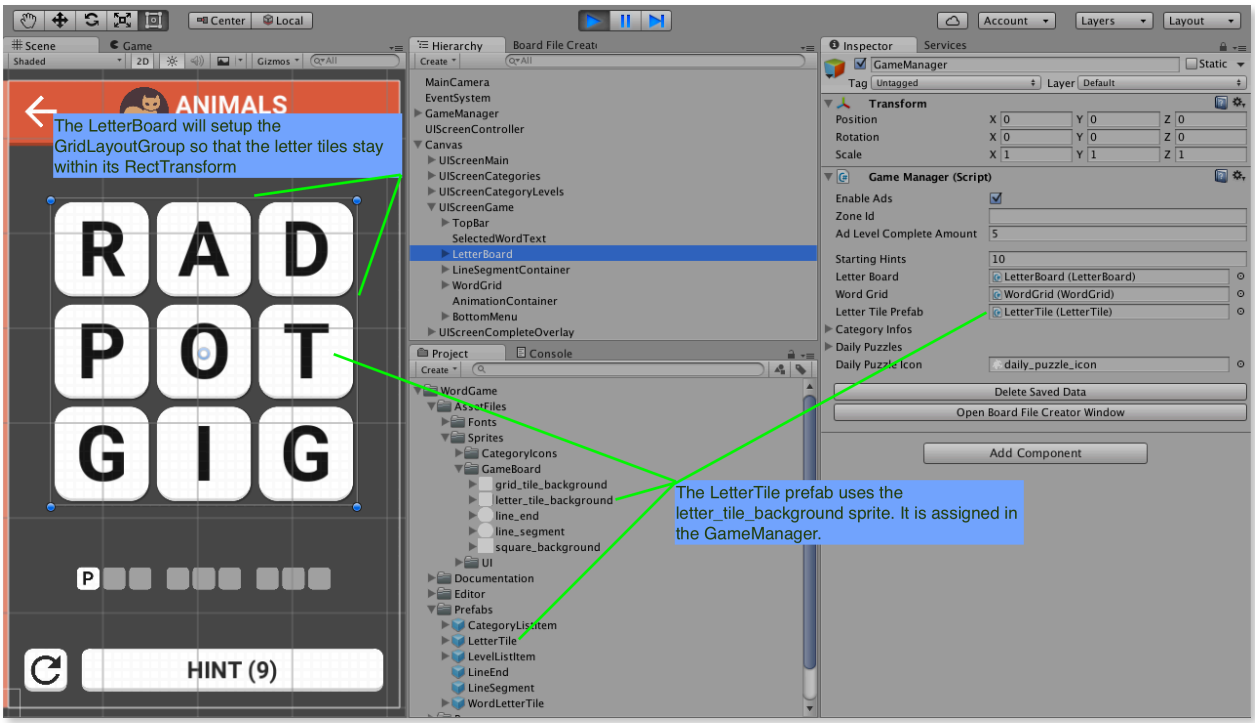
Project Setup

Below are some images that describe how some of the important parts of the project are setup.

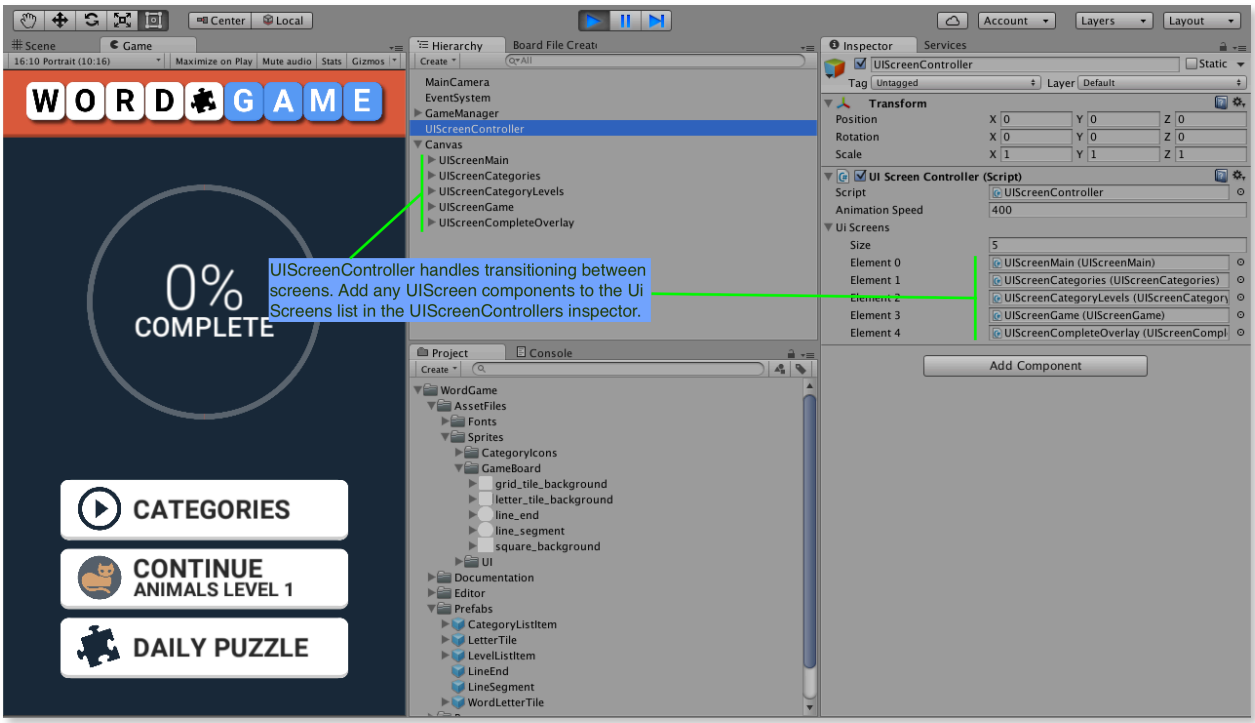
Category Icons:



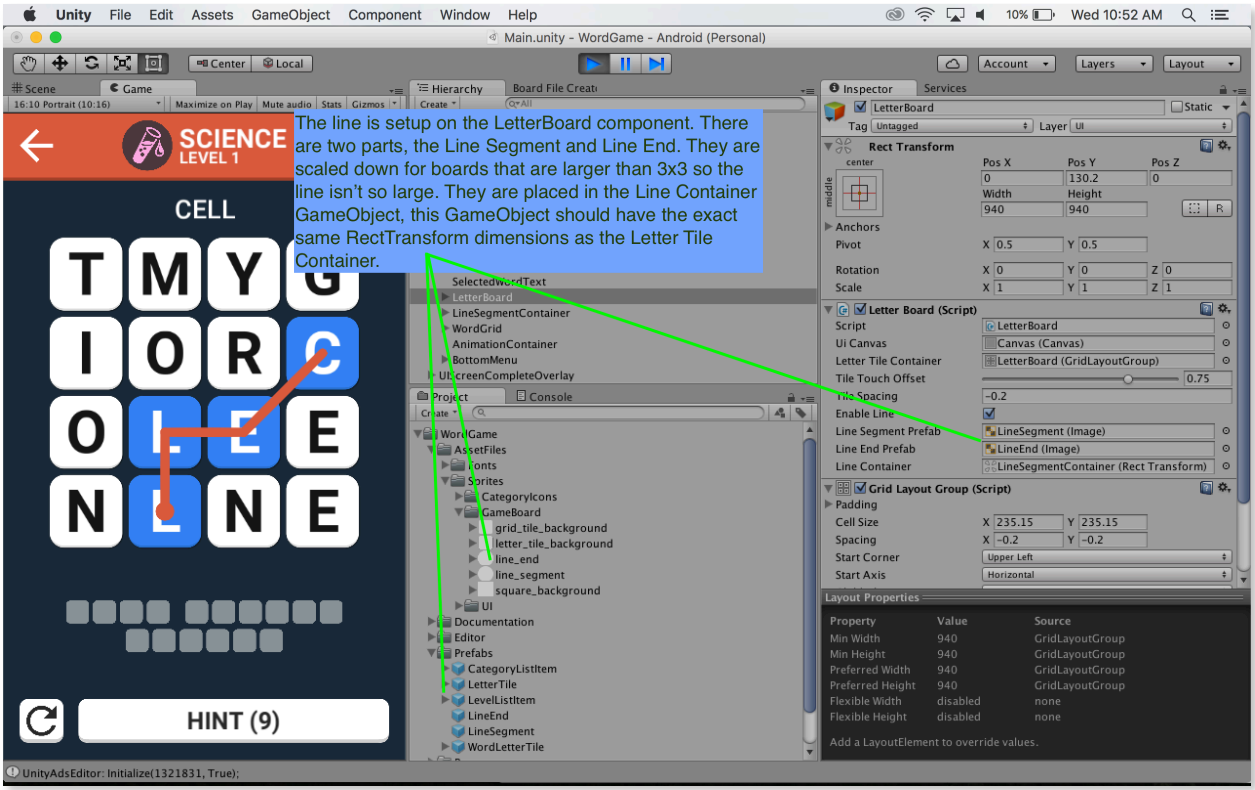
LetterBoard / LetterTiles:



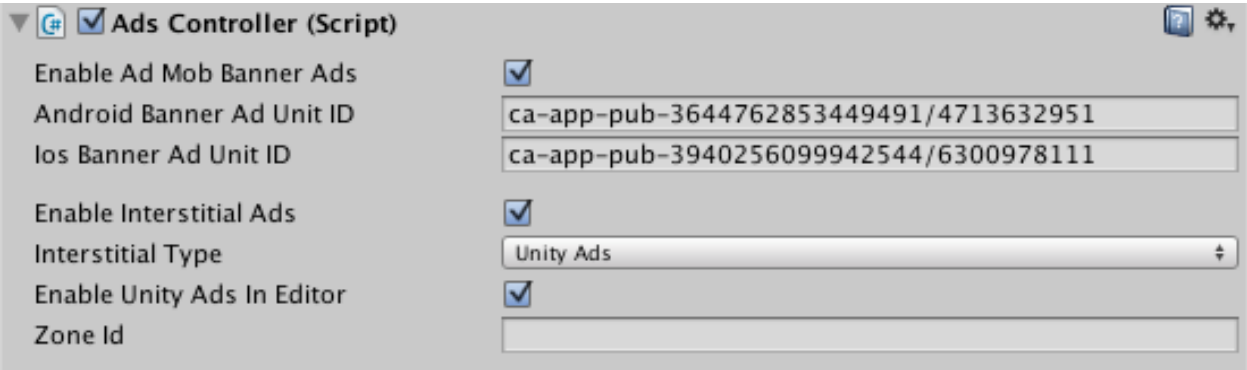
UIScreenController:

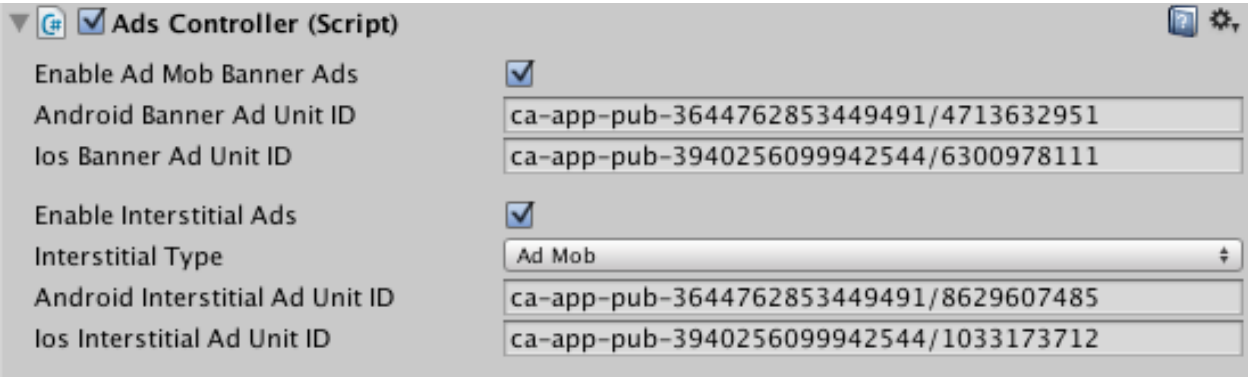


LetterBoard Line:



Ads



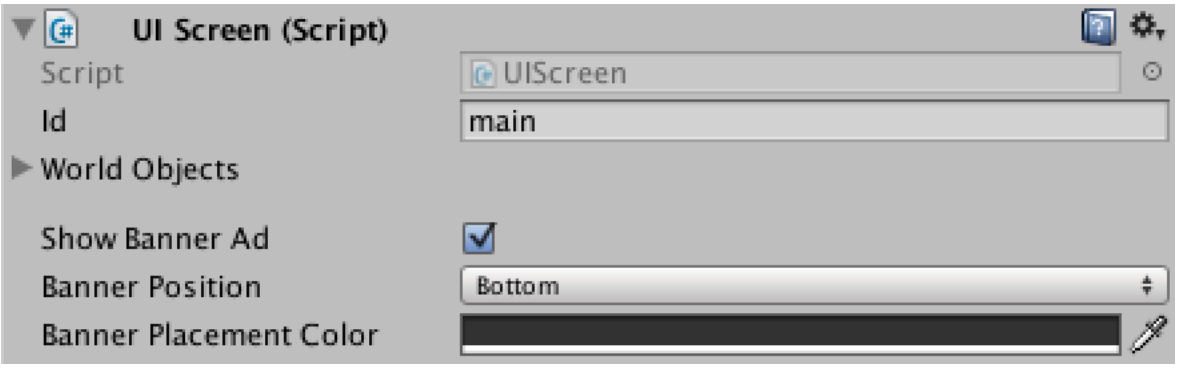


You can enable / disable banner and interstitial ads in the **AdsController** inspector. You can use either Unity Ads or AdMob for interstitial ads. AdMob ads will only appear on device, Unity Ads can be enabled to appear in the Unity Editor for testing purposes.

Interstitial ads will display right after a level is completed. You can set the number of levels that must be completed before a new interstitial ad appears by changing the **Ad Level Complete Amount** field on the **GameManager**.

The AdMob Unit IDs that come with the asset are Googles test ids, you will need to replace them with your own if you would like to use AdMob.

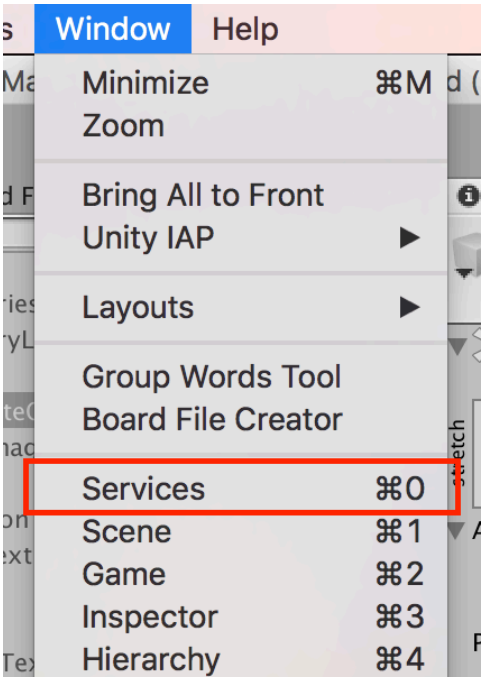
UIScreen / Banner Ads

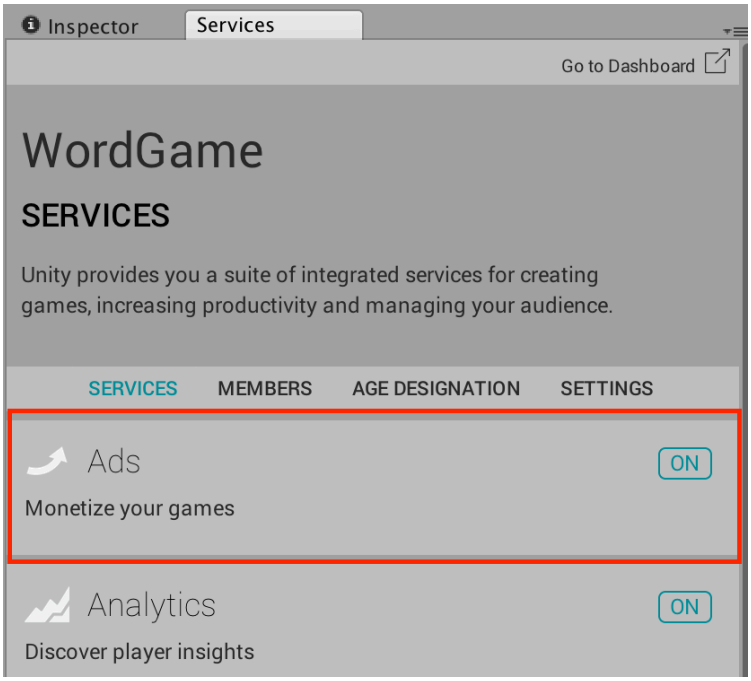


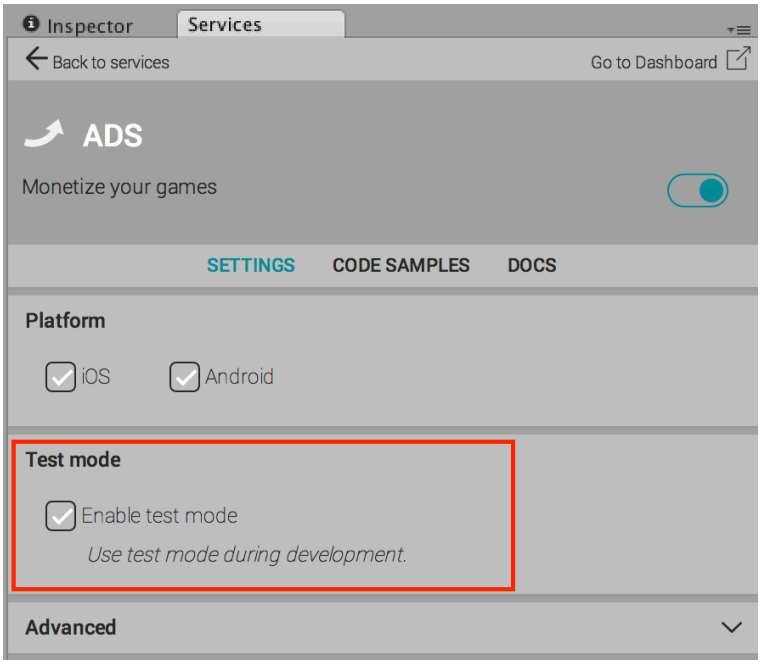
The UIScreen inspector can be used to enable / disable banner ads and also set the position of the banner ads for each screen. If banner ads are enable on a UIScreen then at run time the UIScreen will automatically adjust it's layout to make room for the banner ad so that it does not block any UI.

Unity Ads Setup

In order to enable Unity Ads in the project, navigate to **Window -> Services** and enable **Ads**. When testing Unity ads make sure you click the **Enable test mode** on the Ads Services panel.



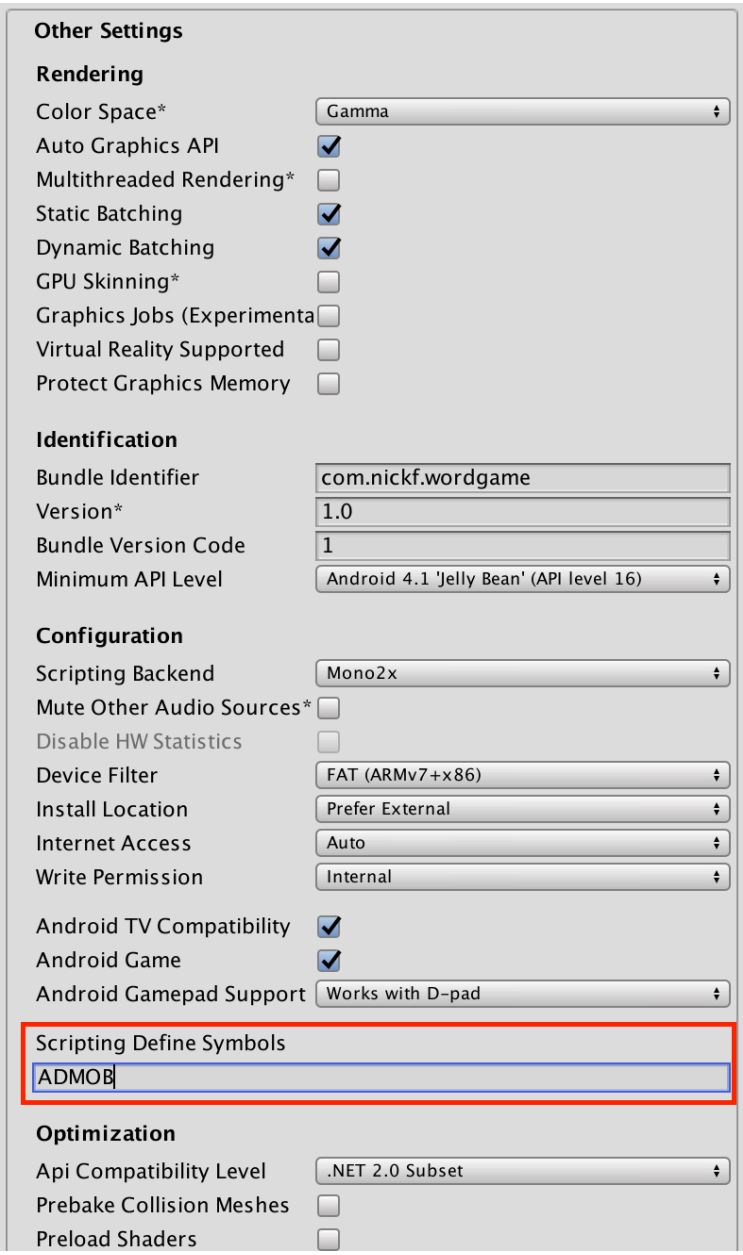




AdMod Setup

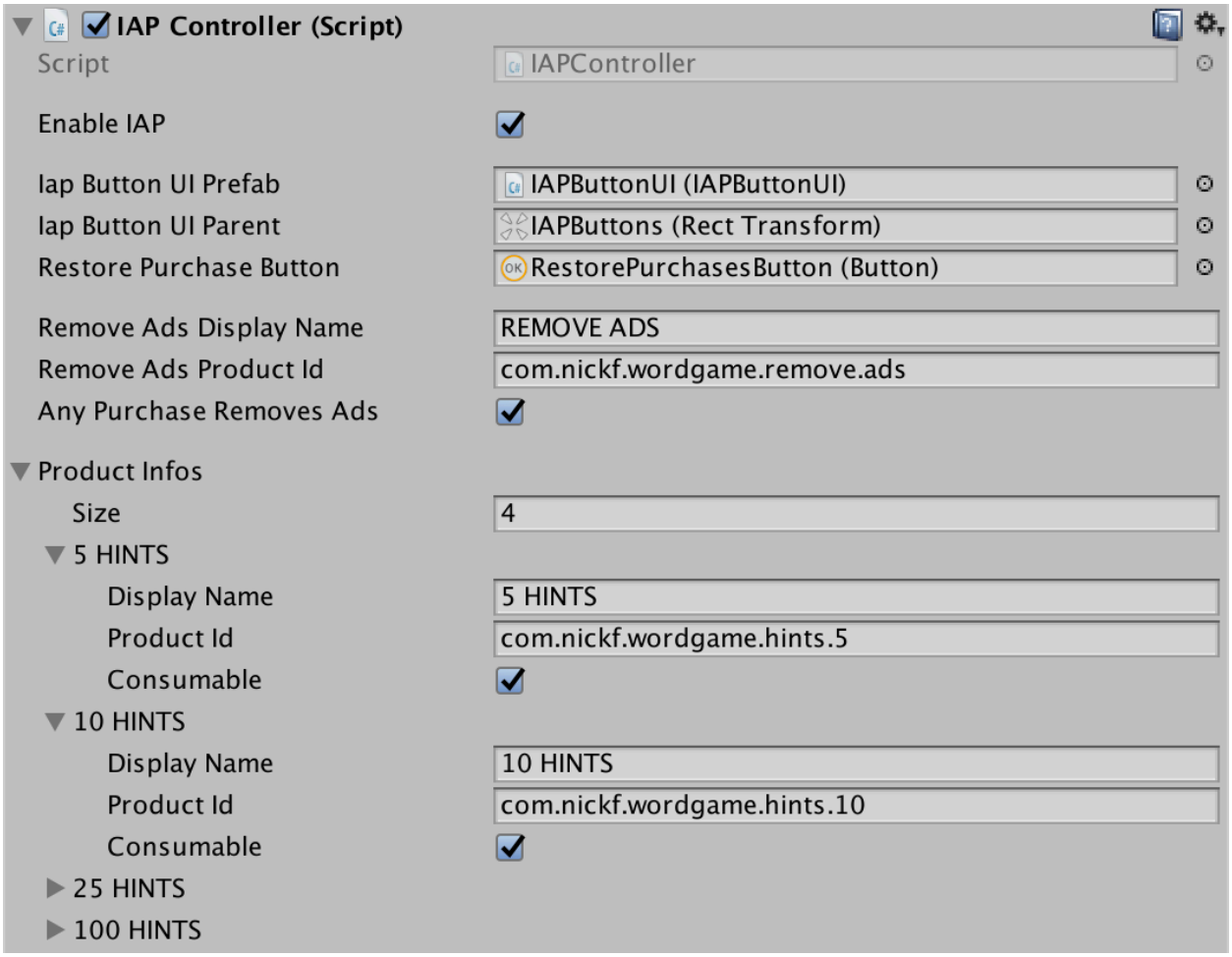
In order to enable AdMob first you need to import the AdMob unitypackage located at **Assets/WordGame/AdMob/GoogleMobileAds.unitypackage**. To import it simply double click in.

Next navigate to the **Player Settings** window and under the **Other Settings** add **ADMOB** to the **Scripting Define Symbols**.



NOTE: This setting is not shared between platforms. You need to add it to the Player Settings on both Android and iOS platforms.

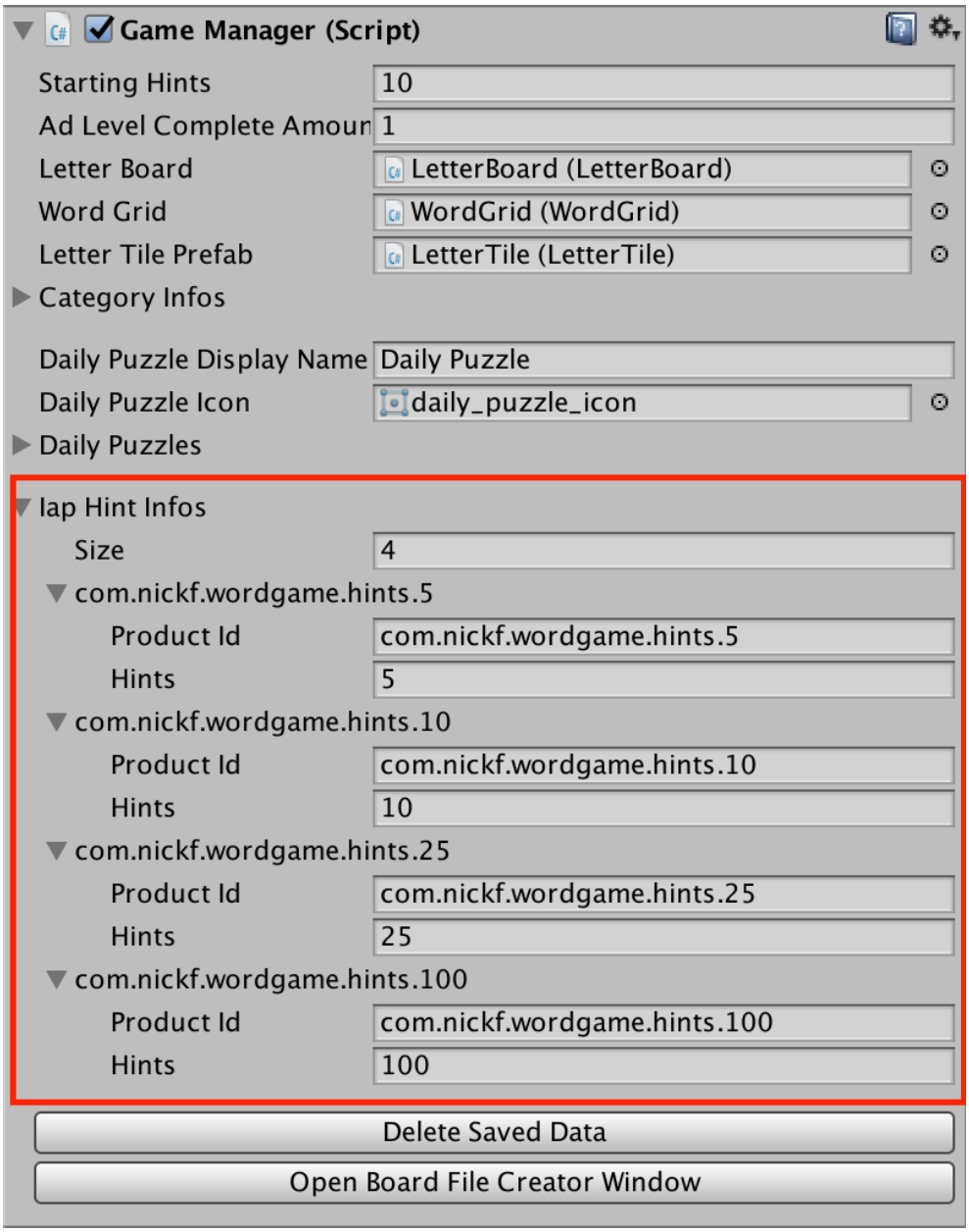
IAP



You can enable / disable IAP and the in app store in the **IAPController** inspector. Currently the Word Game asset is setup to handle two types of IAP purchases: removing ads and purchasing hints. A button will automatically be placed in the store popup for the remove ads (If the **Remove Ads Product Id** is set) and also for each item in the **Product Infos** list. The **Restore Purchase Button** will be disabled in all platforms other than iOS and Mac OSX (Since those platforms require explicit action from the user to restore purchases where as other platforms do it automatically)

All you need to do to enable removing ads is set the **Remove Ads Display Name** and the **Remove Ads Product Id**. If the player purchases this item then the AdsController will automatically turn off ads and remove banner ads (if they were enabled).

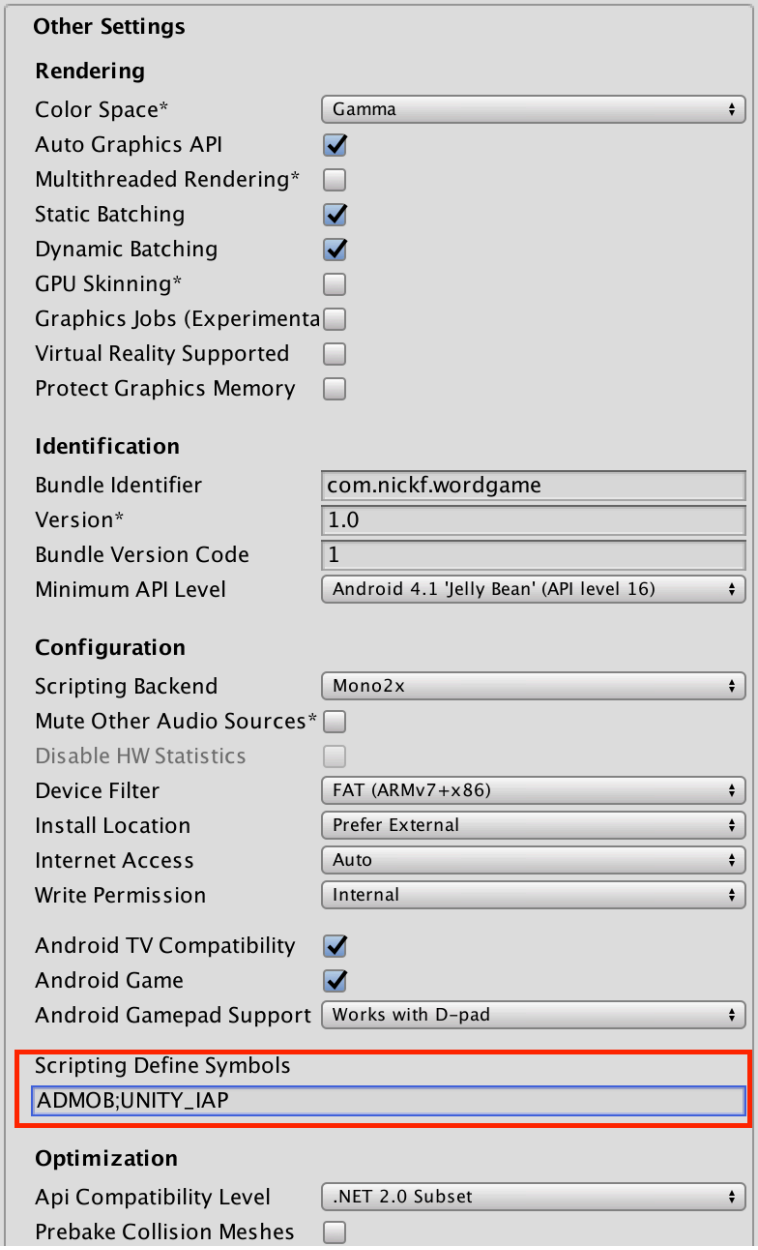
For unlocking purchasing hints, add a new item in the **Product Infos** list for each hint amount. Next copy the **Product Id** into the corresponding **Iap Hint Info** field located in the **GameManager** inspector:



IAP Setup

To enable Unity IAP, first open the Services window by navigating to **Window -> Services** and enable **IAP**.

Next navigate to the **Player Settings** window and under the **Other Settings** add **UNITY_IAP** to the **Scripting Define Symbols**. (If there are already scripting define symbols, you can add others by simply separating them with a semi-colon):



NOTE: This setting is not shared between platforms. You need to add it to the Player Settings on both Android and iOS platforms.