Word Blocks Documentation

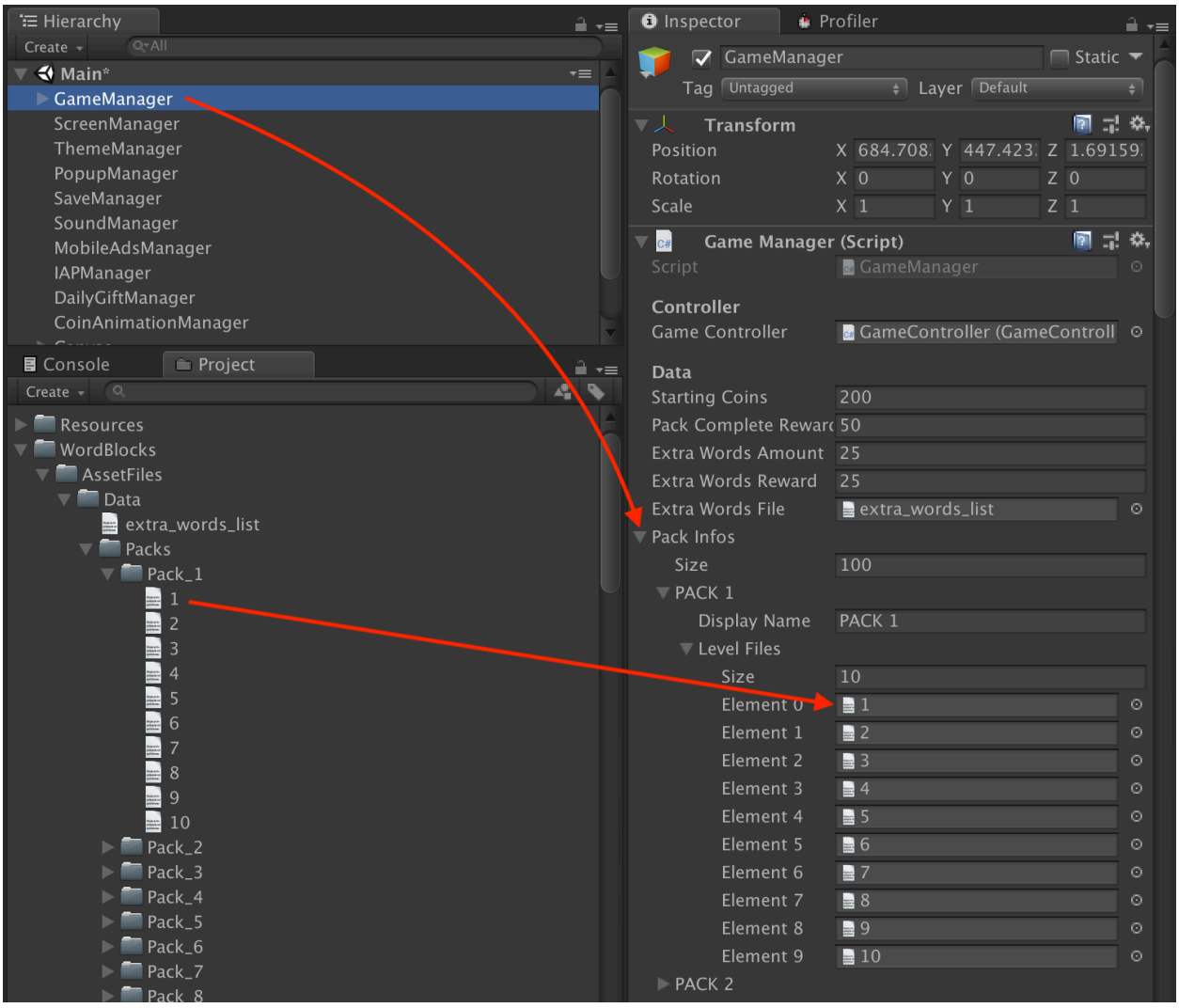
Creating Levels

Each level file is a text file which contains the **hint** on the first line followed by all **words** in the level on their own lines.

For example level one's hint is SCHOOL SUBJECTS and the words are GYM, ART, and DRAMA so it's level file contains:

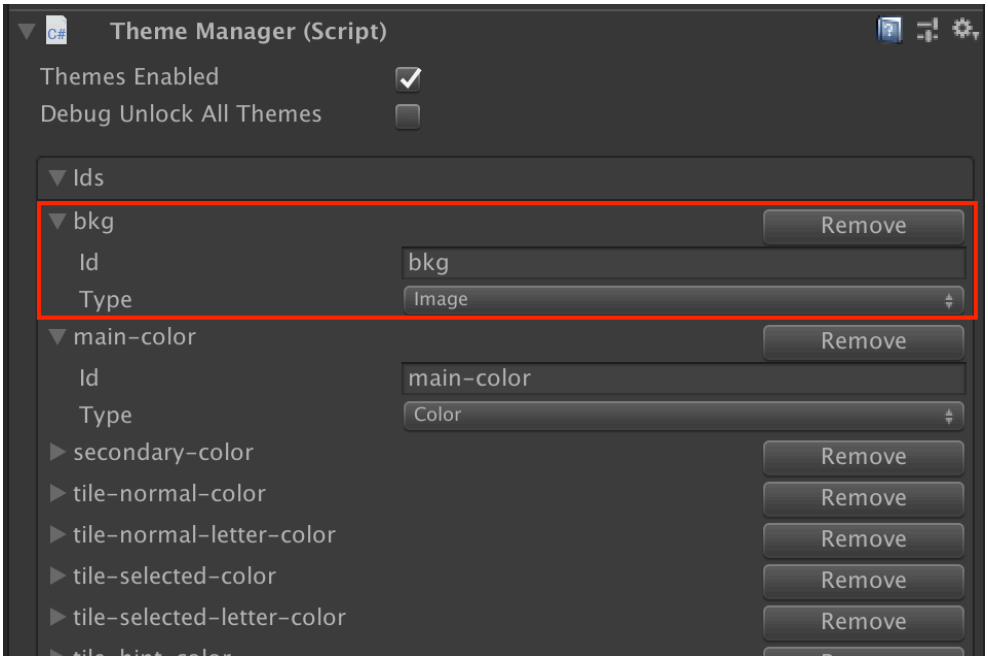
SCHOOL SUBJECTS
GYM
ART
DRAMA

After you create the level text file it can be added to the game using the **GameManager**'s inspector. Click on the GameManager then expand **Pack Infos** and create or expand the pack you would like to add the level to and add the level file to the **Level Files** list.

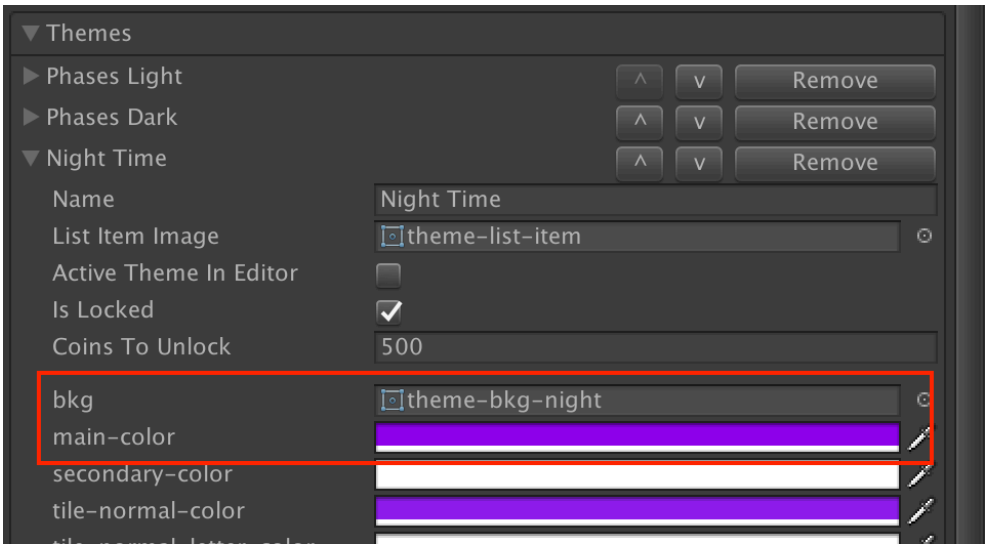


Themes

The themes are created and controlled using the **ThemeManager**. The ThemeManager works by first adding **Ids** and assigning them to a **type** (Image, Color, or Prefab).



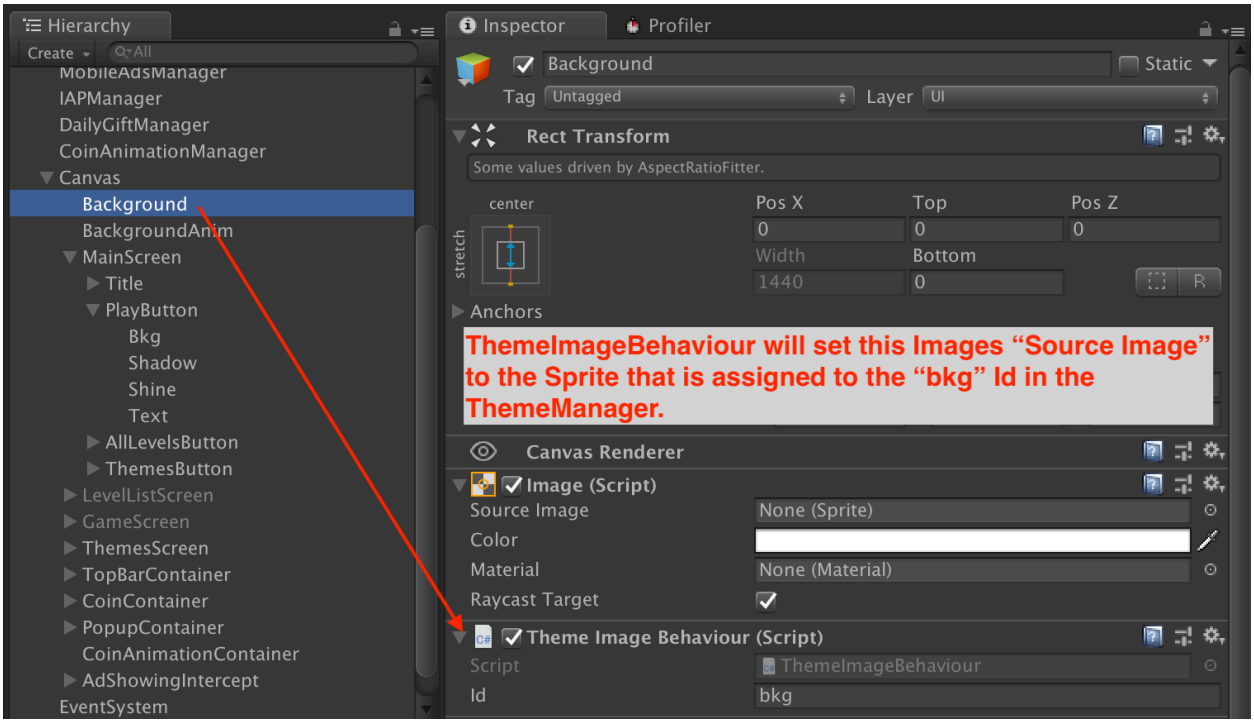
A new field will appear in each **Theme** where you can set the Sprite, Color, or prefab reference for the Id you just created:



To use the themes in the application you can use one of the three components: **ThemeImageBehaviour**, **ThemeColorBehaviour**, and **ThemePrefabBehaviour**. Or you can programmatically fetch the ThemeItem using the ThemeManager.

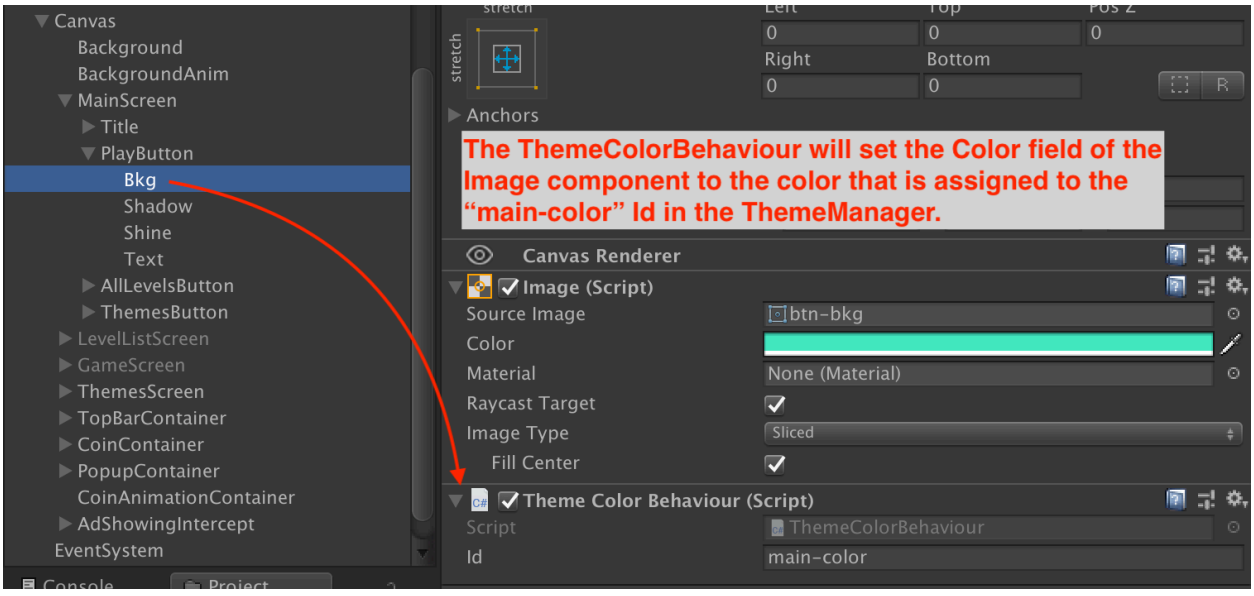
ThemeImageBehaviour

The ThemeImageBehaviour is used for theme Ids that are set to the Image type. When the game runs it sets the **Source Image** of the **Image component** that is attached to the same GameObject to the Sprite set in the ThemeManager for the specified Id. Example:



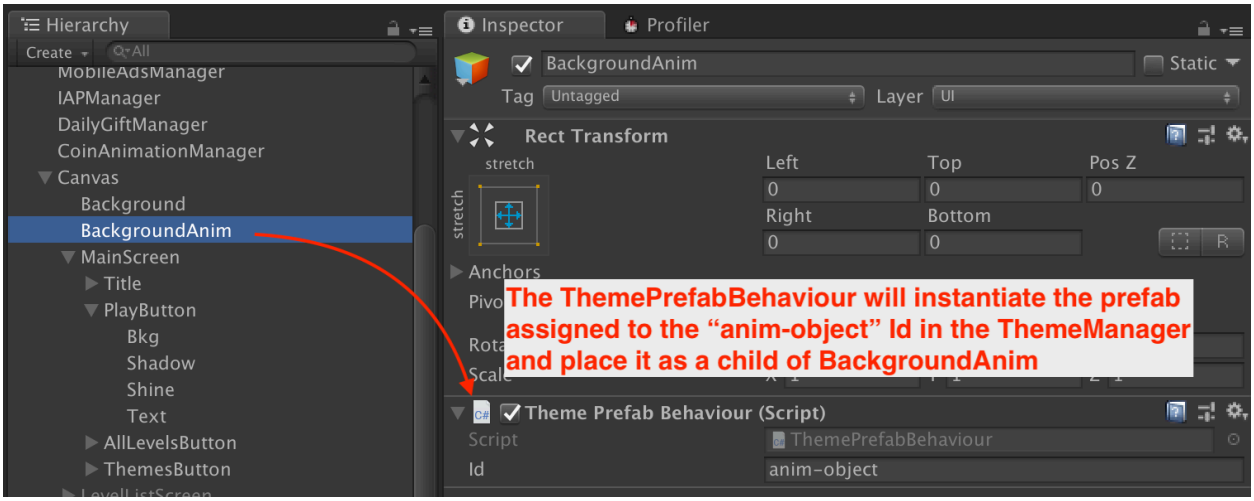
ThemeColorBehaviour

The ThemeColorBehaviour is used for theme Ids that are set to the Color type. When the game runs it sets the Color of the Graphic component (Image, RawImage, Text) to the the color set in the ThemeManager for the specified Id. Example:



ThemePrefabBehaviour

The ThemePrefabBehaviour is used for theme Ids that are set to the Prefab type. When the game runs it will instantiate the prefab set in the ThemeManager for the specified Id and set it as a child. Example:



Programmatically

To fetch a ThemeItem for an Id you can call the following method:

ThemeManager.Instance.GetThemeItem(string id, out ThemeItem themeItem, out ItemId itemId);

This will get the ThemeItem object for the active theme with the given id. An example of this can be found in **GridLetterTile.cs** in the **SetColor** method where it gets the various colors for different states of the tile (normal, selected, etc).

Sounds

Sounds in the game are controlled using the SoundManager. On the SoundManager's inspector you will find a number of Sounds Infos already created and used in the game.

Sound Info fields

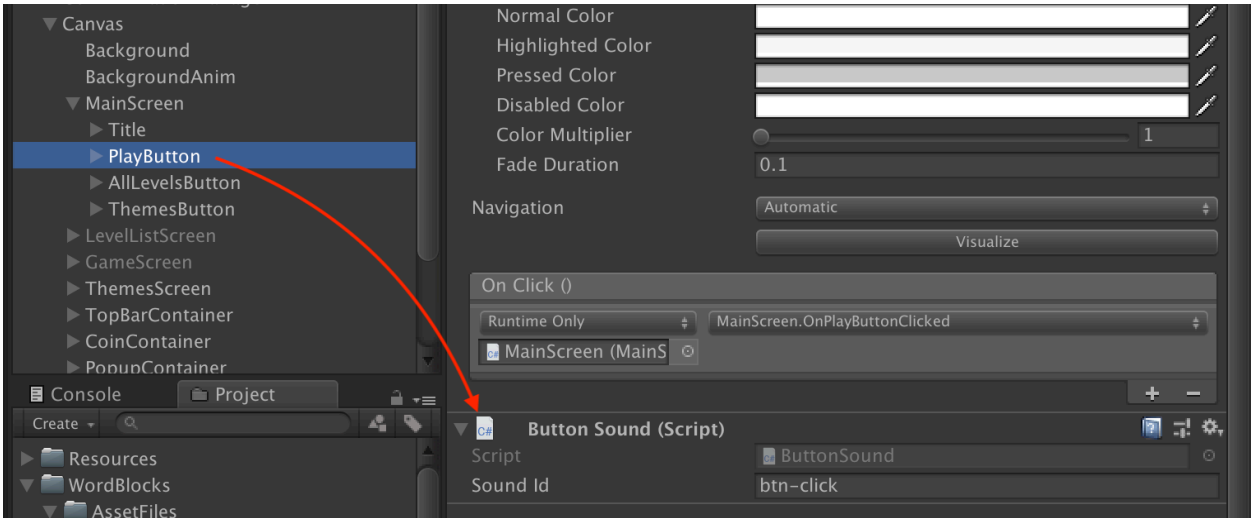
Id - The Id used to play the sound in the game.
Audio Clip - The sound file from your project.
Type - The type of sound (Sound Effect or Music), this is used to turn on/off all sounds of a particular type.
Play And Loop On Start - If selected the Audio Clip will play when the game starts and will loop forever unless it is stopped.
Clip Volume - Sets the volume of the sound when it is played.

Playing Sounds

Sounds can be played by calling the **Play** method on the SoundManager like so:

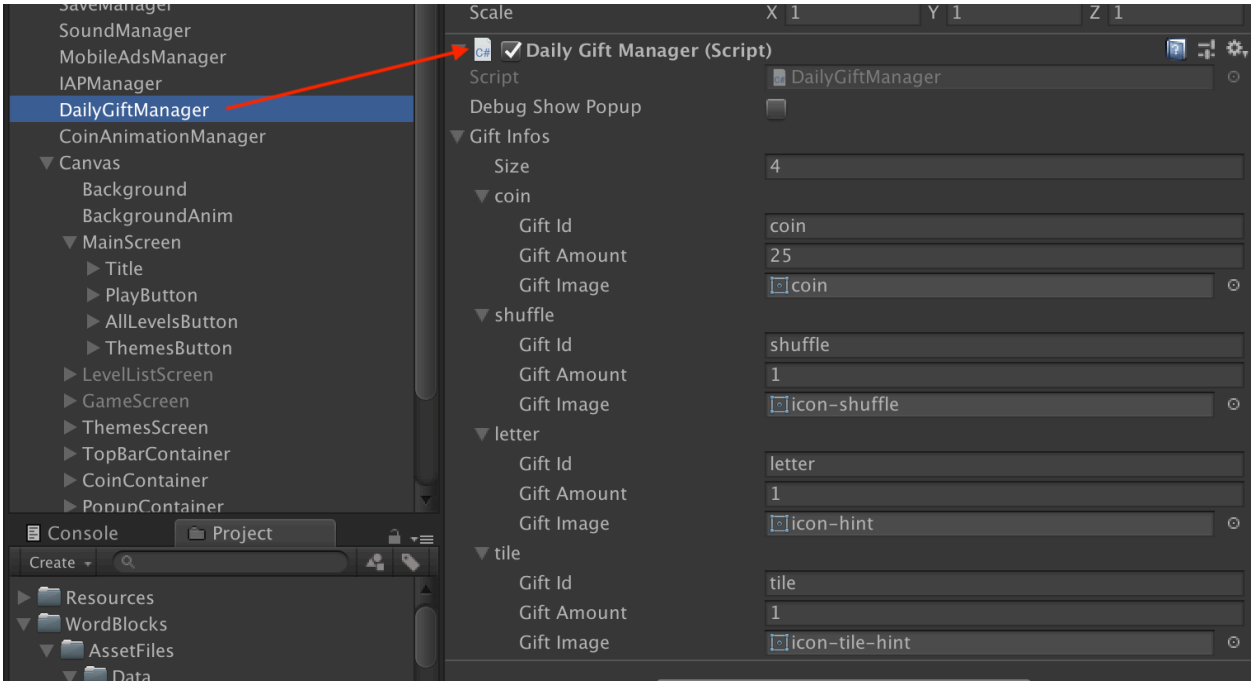
SoundManager.Instance.Play(string id);

You can easily play a sound when a Button is clicked by adding the **ButtonSound** component to a GameObject with a **Button** component. The ButtonSound will play the sound with the specified Id every time the button is clicked.



Daily Gifts

The DailyGiftManager is used to award a gift to the user when they open the application the first time each day. There are 4 gifts in the Word Blocks asset that can be given to the player: coins, shuffle, letter hints, and tile hints. The amount the player gets can be assigned on the DailyGiftManagers inspector.



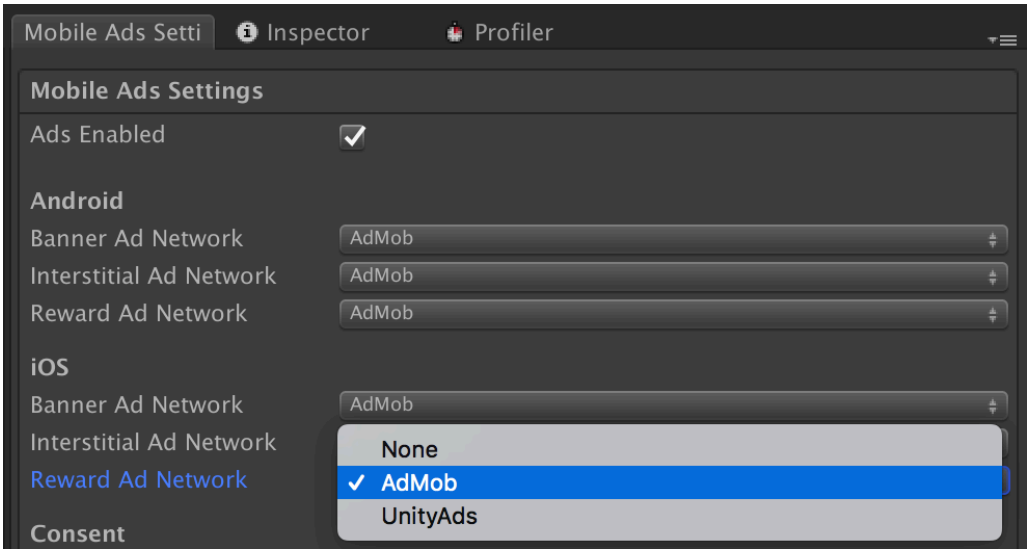
Ads

Ads are setup using the **Mobile Ads Settings** window which can be opened by selecting the menu item **Window -> Mobile Ads Settings** (Or clicking the button on the MobileAdsManager inspector).

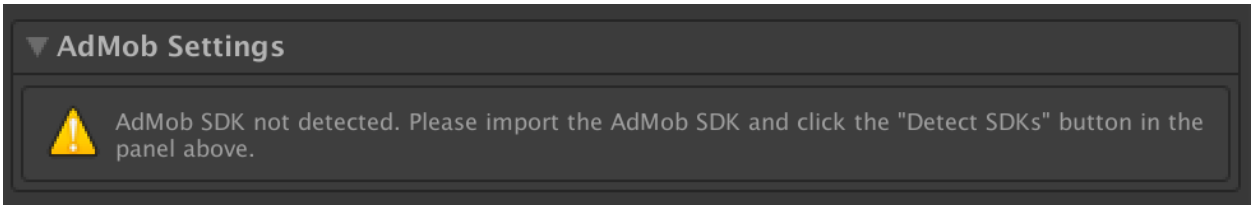
On the Mobile Ads Settings window you can select either **AdMob** or **Unity Ads** to be used for Banner, Interstitial, and/or Reward ads for both Android and iOS platforms. Selecting **None** will disable ads for that platform / ad type.

AdMob Setup

Step1. Select AdMob in one or more of the drop downs.



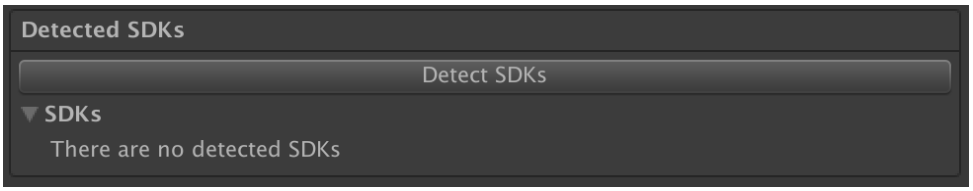
A new section will appear at the bottom of the window called **AdMob Settings**. Expanding it now will display the following warning:



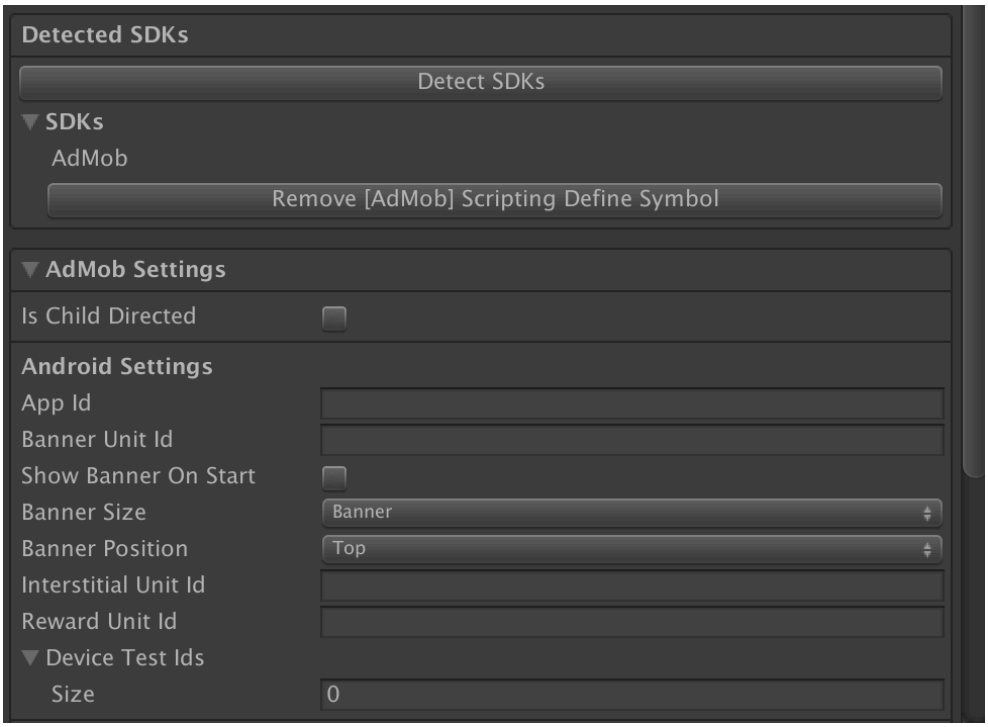
Step2. Download and import the AdMob Unity SDK by clicking on this link **https://github.com/googleads/googleads-mobile-unity/releases** and clicking the GoogleMobileAds.unitypackage

Download the GoogleMobileAds.unitypackage:

Step3. Once the GoogleMobileAds.unitypackge has finished importing into Unity click the Detect SDKs button on the Mobile Ads Settings window:



After Unity finishes compiling AdMob should appear under the SDKs list and the AdMob fields should appear under AdMob Settings



Step4. Add you App Id and Unit Ids to the fields under AdMob Settings

Step5 [Android only]. Open the AndroidManifest located in the folder Plugins/Android/GoogleMobileAdsPlugin and add the following lines in the **application** element, replace ADMOB_APP_ID with your App Id:

```
<meta-data
        android:name="com.google.android.gms.ads.APPLICATION_ID"
        android:value="ADMOB_APP_ID"/>
```

Your AndroidManifest should look something like this:

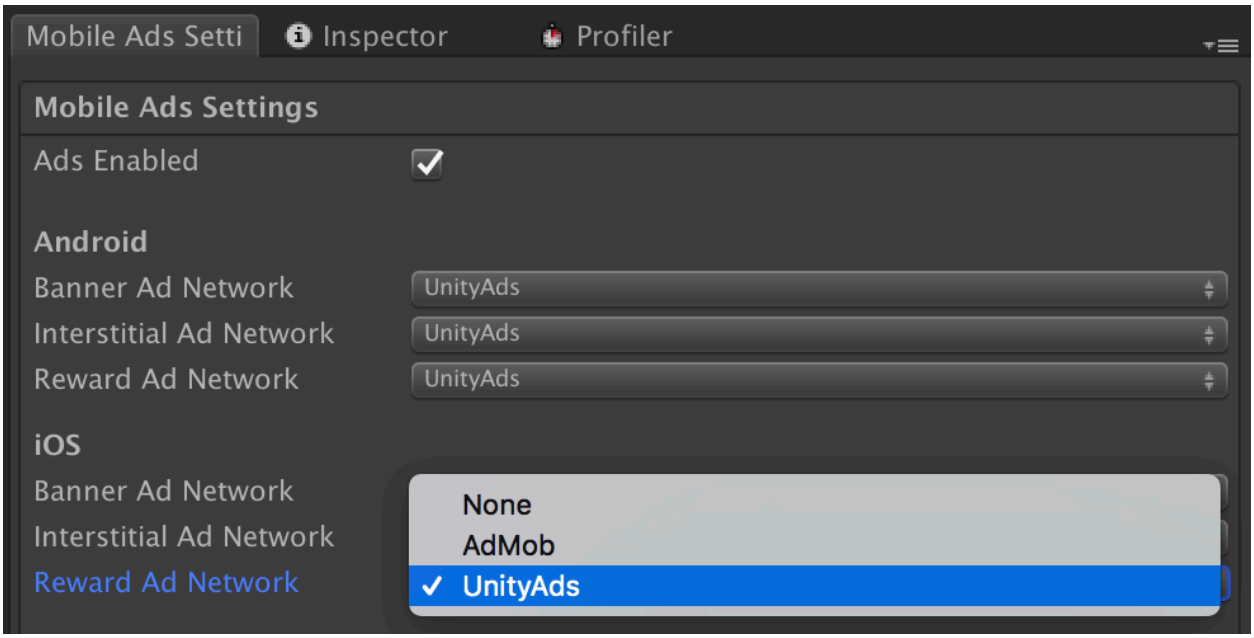
```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.google.unity.ads"
    android:versionName="1.0"
    android:versionCode="1">
  <uses-sdk android:minSdkVersion="14"
      android:targetSdkVersion="19" />
  <application>
    <meta-data
        android:name="com.google.android.gms.ads.APPLICATION_ID"
        android:value="ca-app-pub-3644762853449491~2999379837"/>
  </application>
</manifest>
```

Step6 [Android Only]. Make sure the play services resolver that comes with the GoogleMobileAds plugin has executed by selecting the menu item **Assets -> Play Services Resolver -> Android Resolver -> Resolve**.

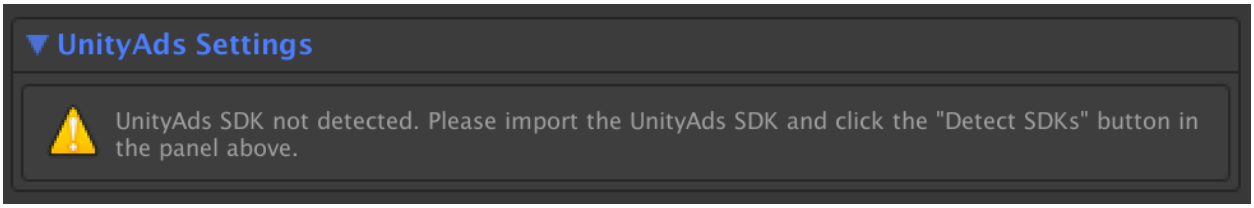
Thats it! AdMob ads should appear in the game.

Unity Ads Setup

Step1. Select AdMob in one or more of the drop downs.

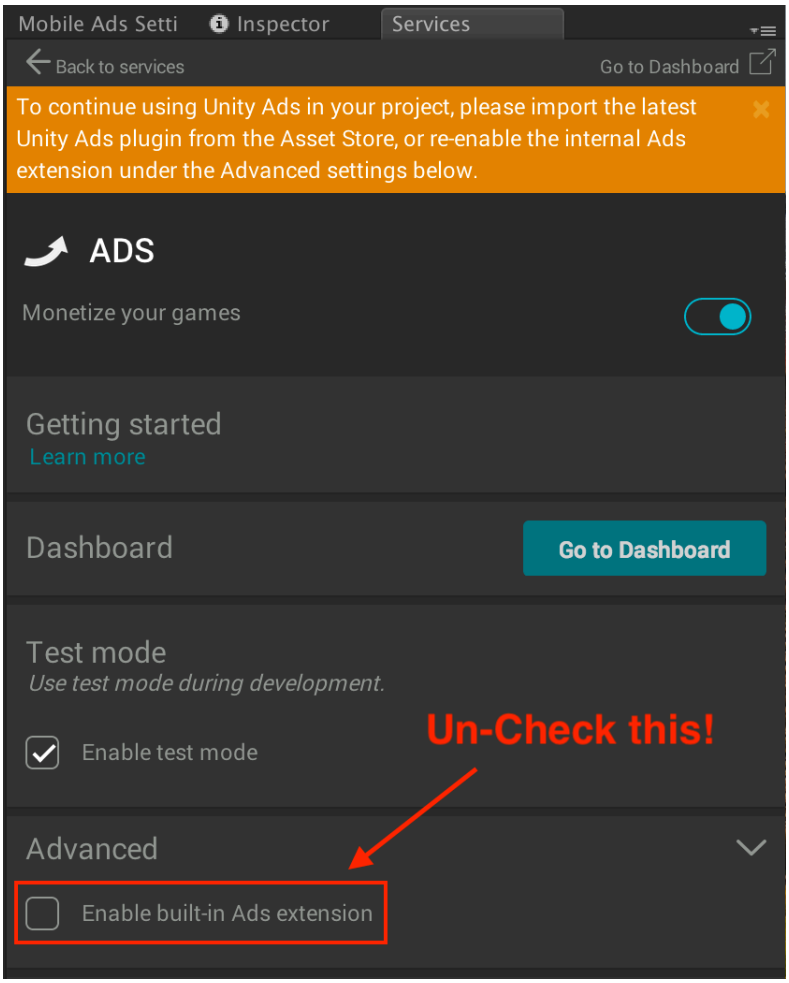


A new section will appear at the bottom of the window called **UnityAds Settings**. Expanding it now will display the following warning:

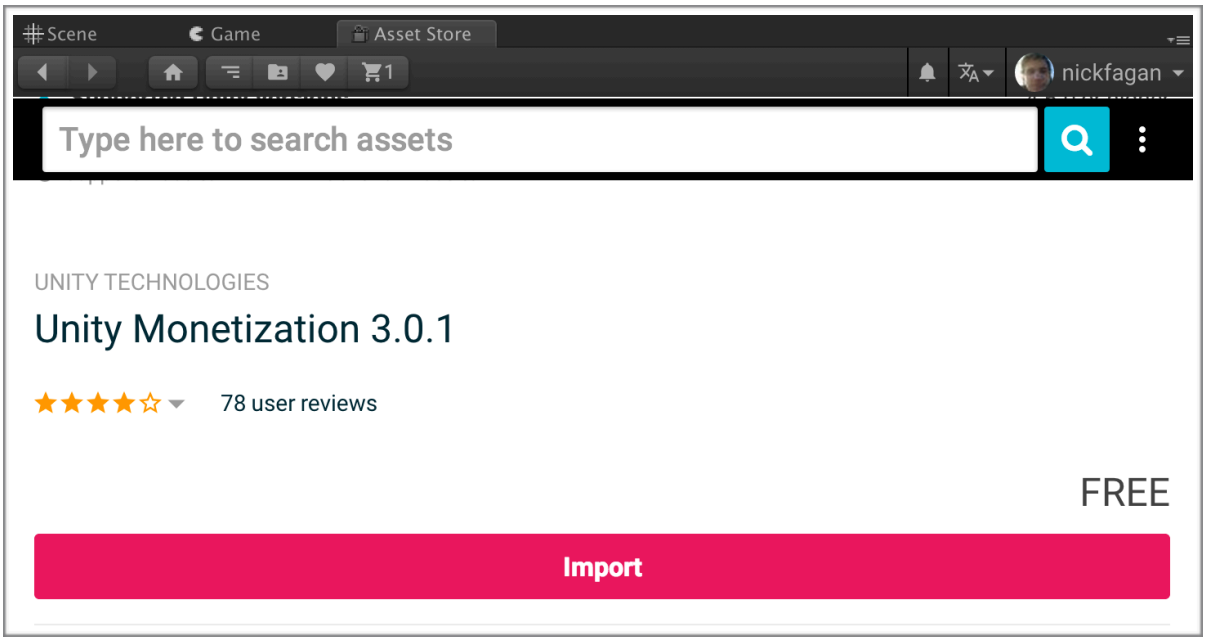


Step2. Enable Ads in the Services window:

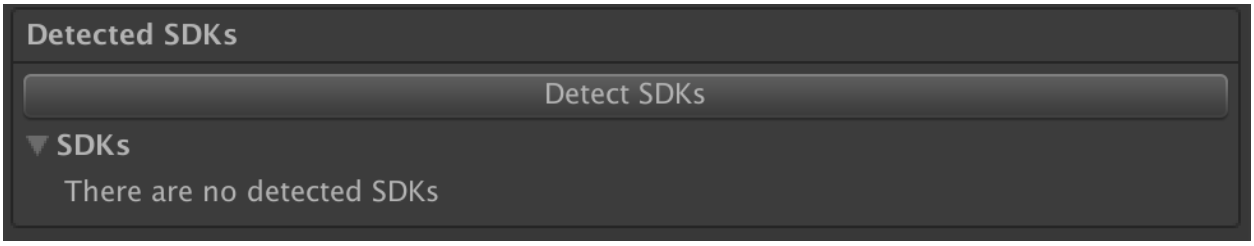
***** IMPORTANT ***** Make sure “Enable built-in Ads Extension” is disabled or it will collide with the Monetization plugin you will import in the next step. To do so expand the **Advanced** section and un-check the field if it is checked



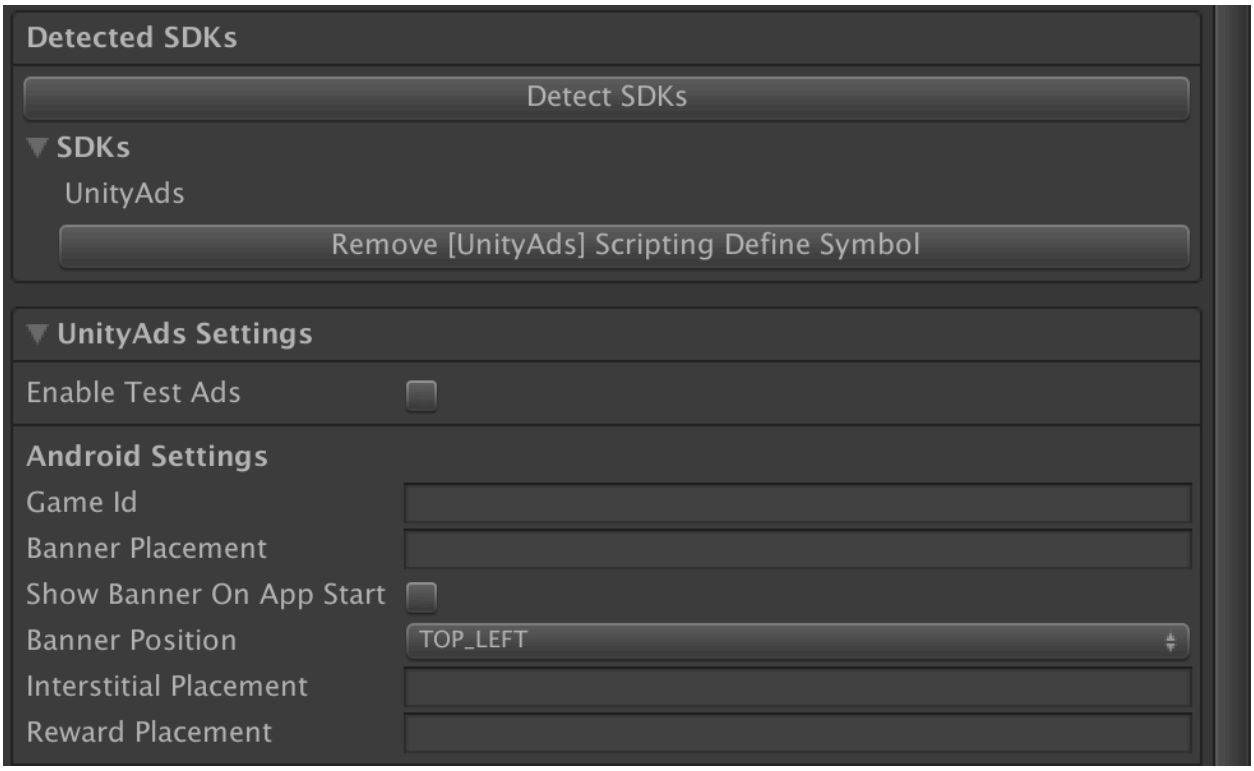
Step2. Open the Asset Store window and Download/Import the Unity Monetization asset:



Step3. Click the **Detect SDKs** button on the Mobile Ads Settings window:



After Unity finishes compiling, UnityAds should appear under the SDKs list and the UnityAds fields should appear under UnityAds Settings:

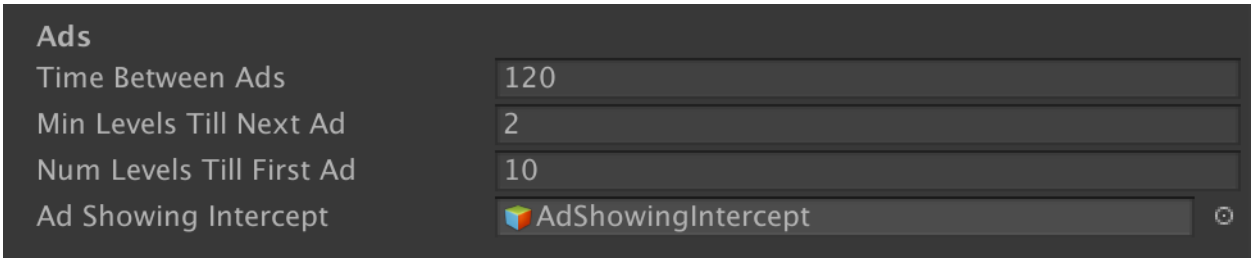


Step4. Add you Game Id and Placement Ids to the fields under UnityAds Settings.

Thats it! Unity Ads will now appear in your game.

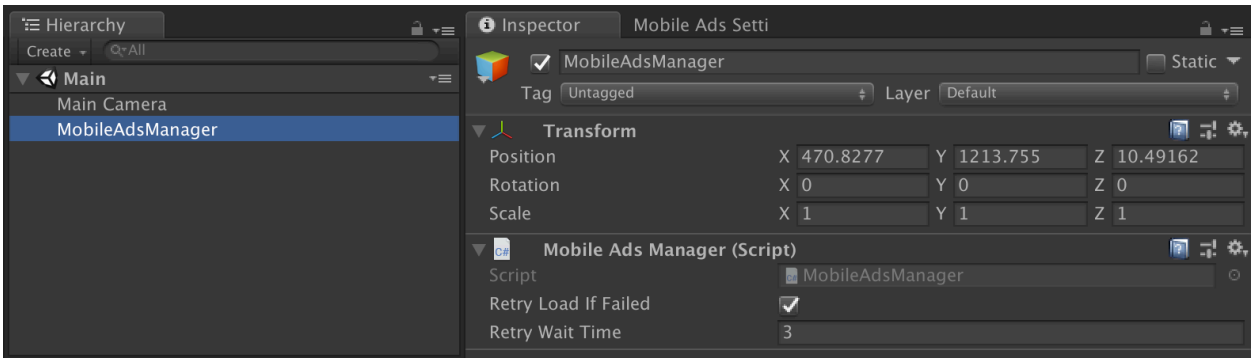
Showing Ads

The game is already setup to show Banner ads on every screen, Interstitial ads when the player starts a level, and a Reward ad button on the Game Screen. The frequency of Interstitial ads can be controlled on the GameManagers inspector under the Ads header:



Time Between Ads is the amount of seconds that must pass since the last ad was shown before a new ad will be shown.
Min Levels Till Next Ad is the minimum number of levels that must be started before an interstitial ad is shown.
Num Levels Till First Ad is the number of levels that must be started before the user sees their first ad. This is so when the user first downloads the app they can play a few levels before being presented with an ad.
Ad Shown Intercept is the GameObject that will be shown when an ad is about to show but hasn't been displayed on the screen yet. This is so the UI is blocked and the user cant click anything when an ad is about to show.

The MobileAdsManager script is used to show ads in your game.



The MobileAdsManager can be accessed anywhere in you projects scripts using the static **Instance** property like so: **MobileAdsManager.Instance**

If the **Retry Load If Failed** is checked then when an ad fails to load for any reason (no internet connection, no ad fill, etc) the the MobileAdsManager will wait the amount of seconds specified in **Retry Wait Time** then it will try and load the ad again. If Retry Load If Failed is un-check then when an ad fails to load you will have to manually call one of the load methods.

NOTE: If one of the show methods (ShowInterstitialAd, ShowRewardAd) is called and there is no pre-loaded ad then the MobileAdsManager will attempt to load an ad so the next time the show method is called there will be an ad ready to show.

Showing Banner Ads

To show banner ads call the **ShowBannerAd** method on the MobileAdsManager Instance:

```
MobileAdsManager.Instance.ShowBannerAd();
```

You can hide the banner ad by calling the **HideBannerAd** method.

Showing Interstitial Ads

To show interstitial Ads call the **ShowInterstitialAd** method on the MobileAdsManager Instance:

```
MobileAdsManager.Instance.ShowInterstitialAd();
```

The ShowInterstitialAd method can be passed a callback method that is invoked when the interstitial ad closes and is no longer displayed on the screen.

The ShowInterstitialAd method returns a boolean indicating if an interstitial ad has been pre-loaded and is going to show. If the method returns false then there was no interstitial ad loaded so no ad will be displayed to the user. If it returns true then an interstitial ad has been pre-loaded an is about to show.

Showing Reward Ads

To show interstitial Ads call the **ShowRewardAd** method on the MobileAdsManager Instance:

```
MobileAdsManager.Instance.ShowRewardAd();
```

The ShowReward method can be passed two callback methods:

onClosedCallback - Invoked when the reward ad closes and is no longer displayed on the screen.
onRewardGrantedCallback - Invoked when the user should be granted a reward for watching the reward ad. This method is a reward id and reward amount.

The onClosedCallback will always be called after a reward ad is shown and is called when the reward ad is no longer displayed on the screen. The onRewardGrantedCallback will only be called if the user watched the ad and should be granted the reward, if the user closed the ad in any way then onRewadGrantedCallback will not be invoked. (onRewadGrantedCallback is always invoked before onClosedCallback).

The ShowReward method returns a boolean indicating if a reward ad has been pre-loaded and is going to show. If the method returns false then there was no reward ad loaded so no ad will be displayed to the user. If it returns true then an reward ad has been pre-loaded an is about to show.

Events

There are a number of ad events you can listen to on the MobileAdsManager script. To listen for one of the events add the following code anywhere in your project and replace OnEvent with the event you wish to listen for:

```
MobileAdsManager.Instance.OnEvent += YourEventMethod;
```

Banner Events

OnBannerAdLoading - Invoked when a banner ad starts loading.
OnBannerAdLoaded - Invoked when a banner ad has loaded successfully.
OnBannerAdFailedToLoad - Invoked when a banner ad fails to load.
OnBannerAdShown - Invoked when the banner ad is shown on the screen.
OnBannerAdHidden - Invoked when the banner ad is hidden from the screen.

Interstitial Events

OnInterstitialAdLoading - Invoked when an interstitial ad starts loading.
OnInterstitialAdLoaded - Invoked when an interstitial ad has successfully loaded.
OnInterstitialAdFailedToLoad - Invoked when an interstitial ad failed to load.
OnInterstitialAdShowing - Invoked when an interstitial ad is about to show on the screen.
OnInterstitialAdShown - Invoked when an interstitial ad has been displayed on the screen.
OnInterstitialAdClosed - Invoked when an interstitial ad has closed and is no longer displayed on the screen.

Reward Events

OnRewardAdLoading - Invoked when an reward ad starts loading.
OnRewardAdLoaded - Invoked when an reward ad has successfully loaded.
OnRewardAdFailedToLoad - Invoked when an reward ad failed to load.
OnRewardAdShowing - Invoked when an reward ad is about to show on the screen.
OnRewardAdShown - Invoked when an reward ad has been displayed on the screen.
OnRewardAdClosed - Invoked when an reward ad has closed and is no longer displayed on the screen.
OnRewardAdGranted - Invoked when a reward ad has been watched and the reward should be granted to the user. This event is passed the reward id and reward amount that is set on the ad networks dashboard.

Consent

Consent can be required before any ads are loaded by setting the **Consent Setting** on the Mobile Ads Manager. There are three types you can set the consent setting to:

Not Required - Consent is not required for ads to be loaded.

Required Only In EEA - Consent is only required for users in the European Economic Area. When the app starts it first attempts to determine if the user is located in the EEA and if so ads will not be loaded until the consent status has been set to either Personalized or Non-Personalized. If the user location can not be determined for any reason then it errs on the side of caution and requires consent before ads are loaded.

Require All - Consent is required for all users before ads are loaded.

Setting the Consent Status

If consent is required before ads are loaded then the **SetConsentStatus** method must be called on the MobileAdsManager to set the consent status to either Personalized or Non-Personalized ads.

To set the consent simply call the method like so:

```
MobileAdsManager.Instance.SetConsentStatus(consentStatus);
```

The consentStatus parameter is an integer value:

1 - Indicates the user has consented to receive personalized ads
0 - Indicates the user should only be shown non-personalized ads.

Testing

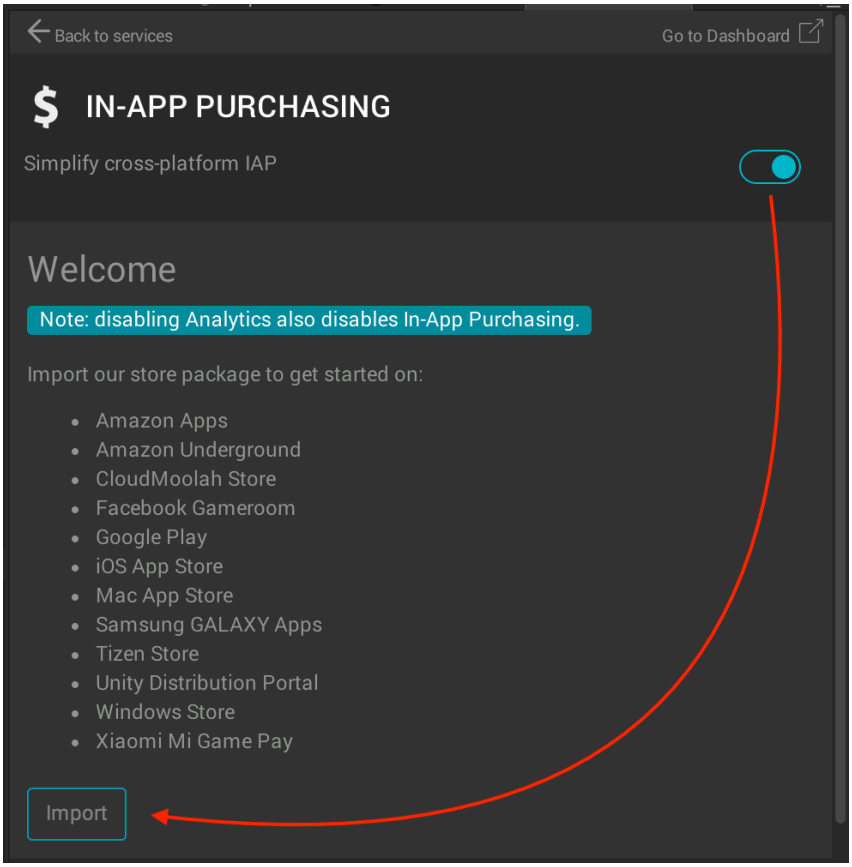
You should use the TestScene located in the Scenes folder to test if you have setup your ad networks properly. The test scene can be used to show ads and will print out log messages if there are any errors.

IAP

IAP is setup using the **IAP Settings** window which can be opened by selecting the menu item **Window -> IAP Settings** (Or clicking the button on the IAPManagers inspector).

Enable IAP

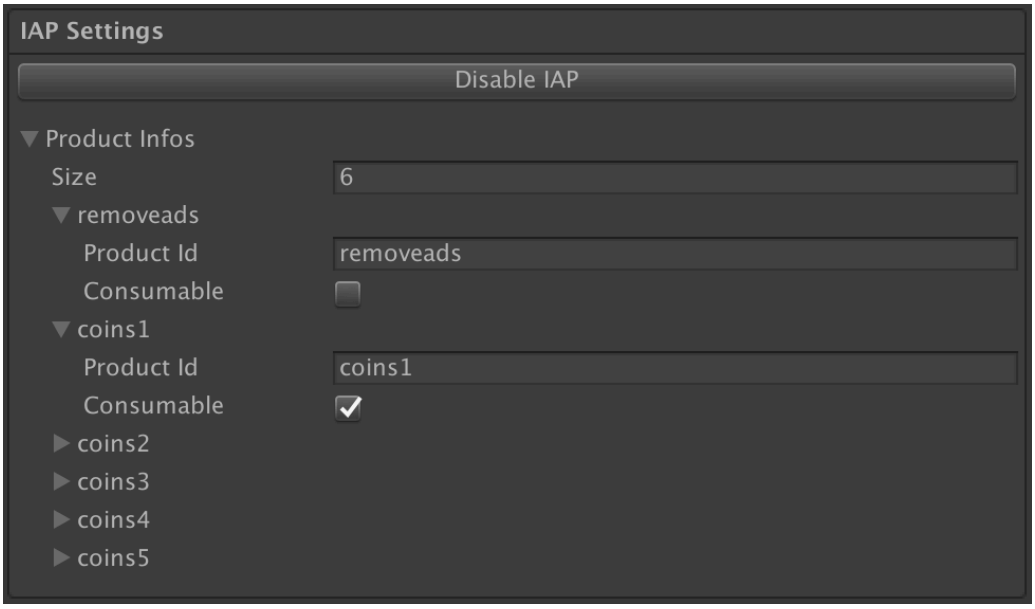
To enable IAP first you need to import the Unity plugin from the Services window. Open the Services window and turn on IAP then click the Import button:



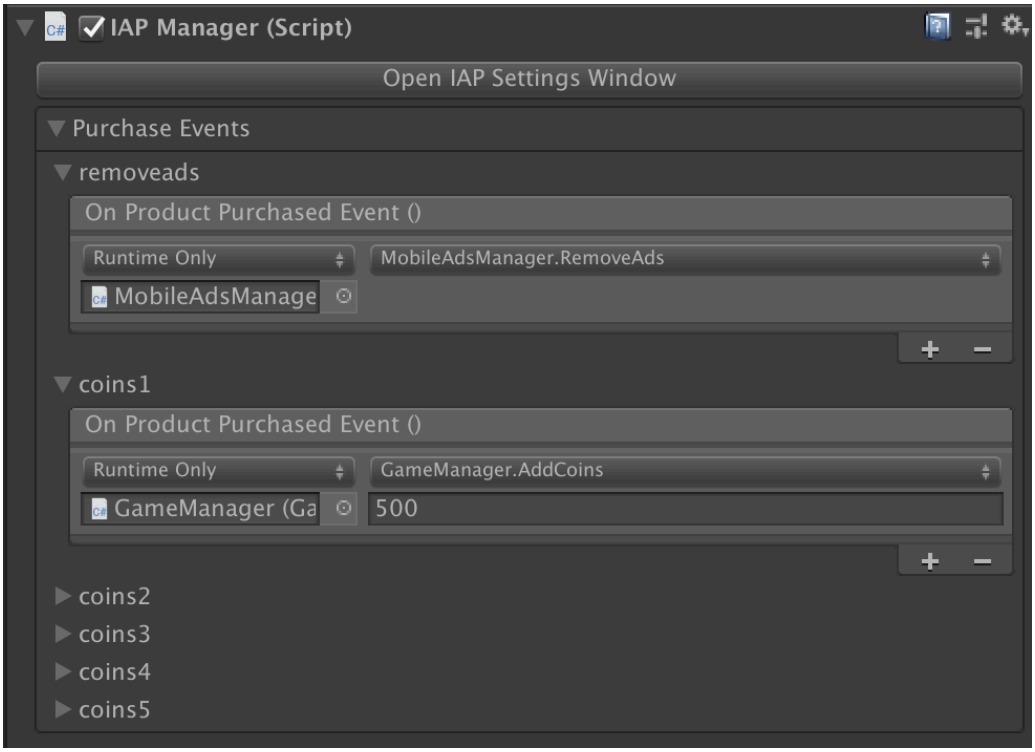
Once it has finished importing you can open the IAP Settings window and click the Enable IAP button which will enable the code in the project.

Add Product Ids

To add products open the IAP Settings window and add a Product Info for each product in your game. The Word Blocks asset comes with six products already configured in the game, you can replace the assets product ids for your own or you can remove them and add new ones

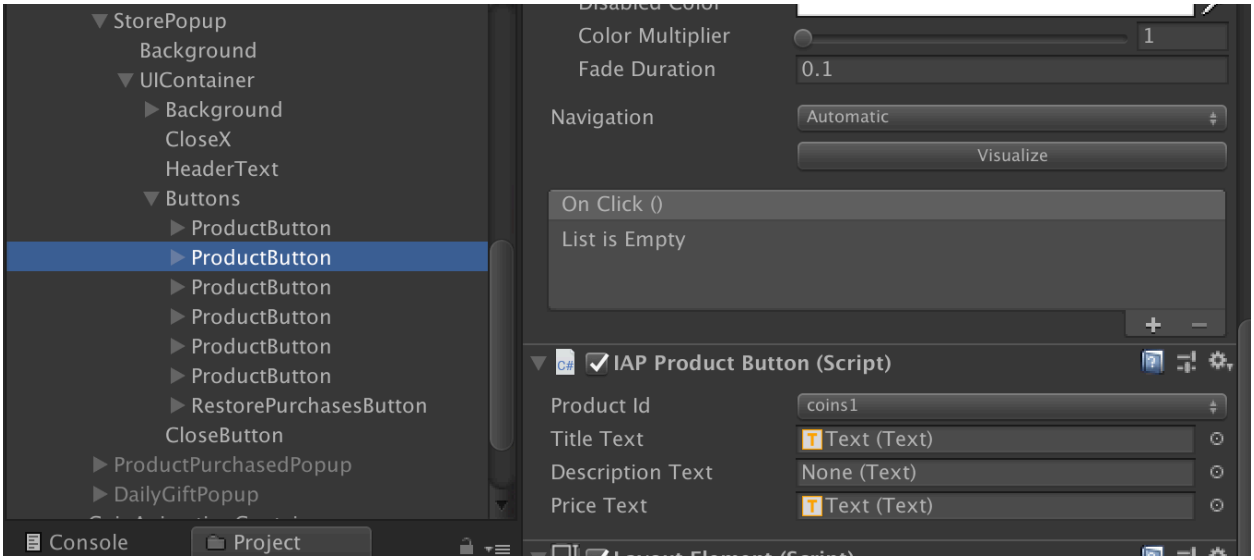


Once you add a new product an entry will appear on the IAPManager's inspector. Here you can add events for when the product is purchased:



Purchase A Product

To invoke the purchasing of a product you can use the **IAPProductButton** component.



The Product Id dropdown will contain all the products you created in the IAPSettings. When the button is clicked in the game the IAPManager's BuyProduct method will be called with selected Product Id.

You can also call the BuyProduct method manually like so:

IAPManager.Instance.BuyProduct(string productId);

You can also listen for successful product purchases with the OnProductPurchases action like so:

IAPManager.Instance.OnProductPurchases += YourMethod;

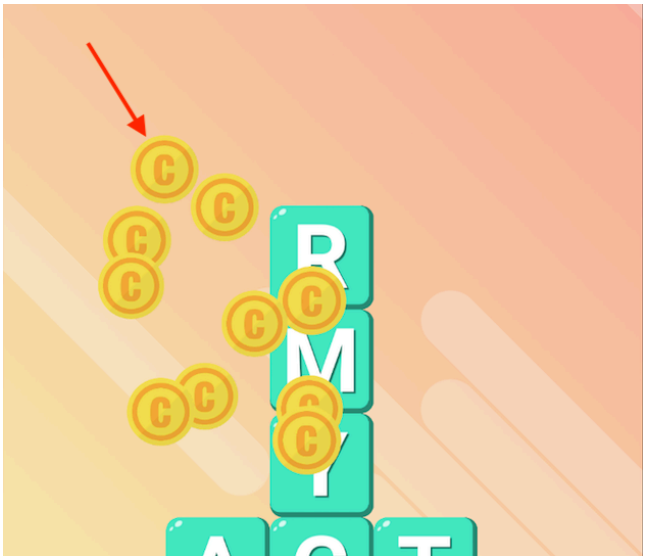
Testing

You should use the TestScene located in the Scenes folder to test if you have setup IAP correctly. The scene will create a button for each product in the IAPSettings window which you can click to purchase the product. Logs will output on the screen to show and errors that may occur.

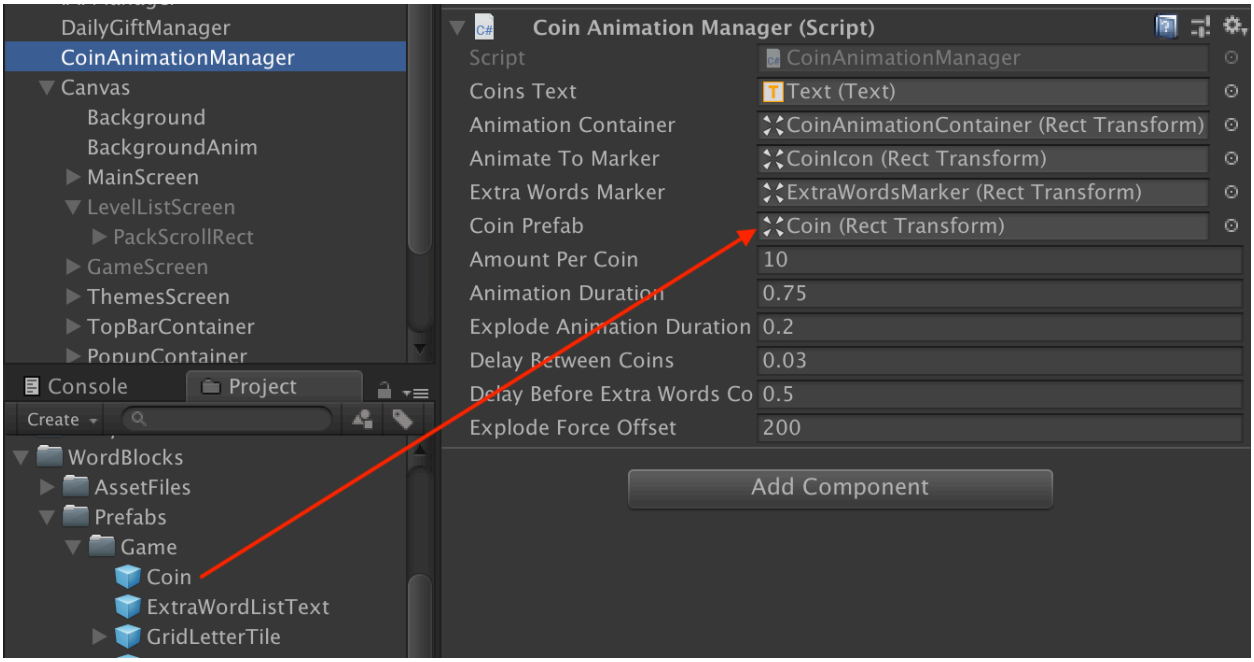
Project

In this section I will show you where you can find / edit some game elements that are instantiated when the game runs and are not located in the Scenes hierarchy.

Coins

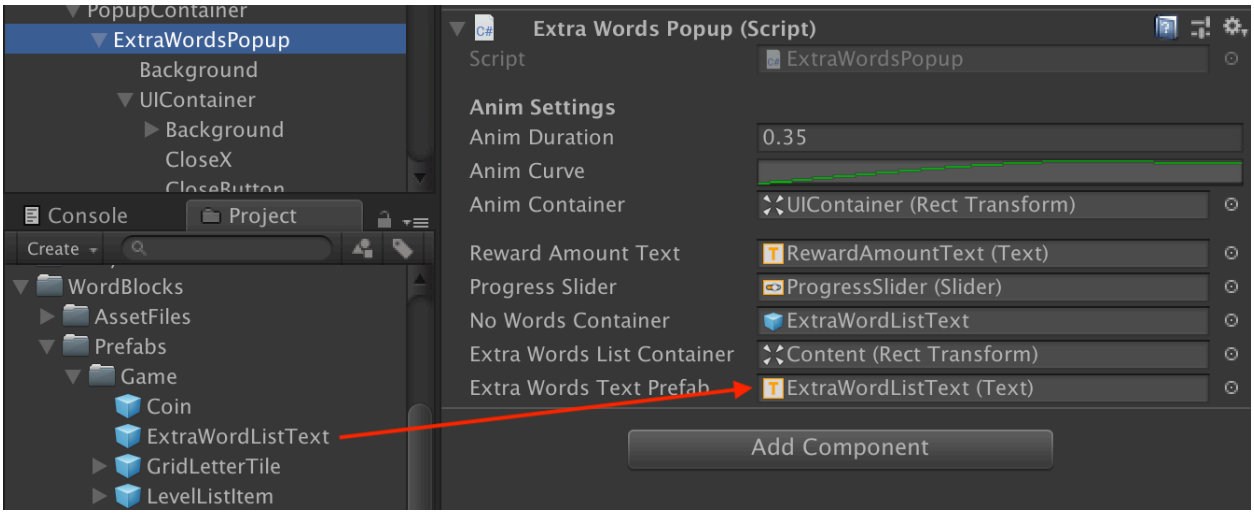


The flying coins are created / controlled by the CoinAnimationManager. The **CoinPrefab** is the prefab that is instantiated when a coin needs to animated across the screen. The **Coin** prefab in the Prefabs folder is the coin that is being used in the asset.



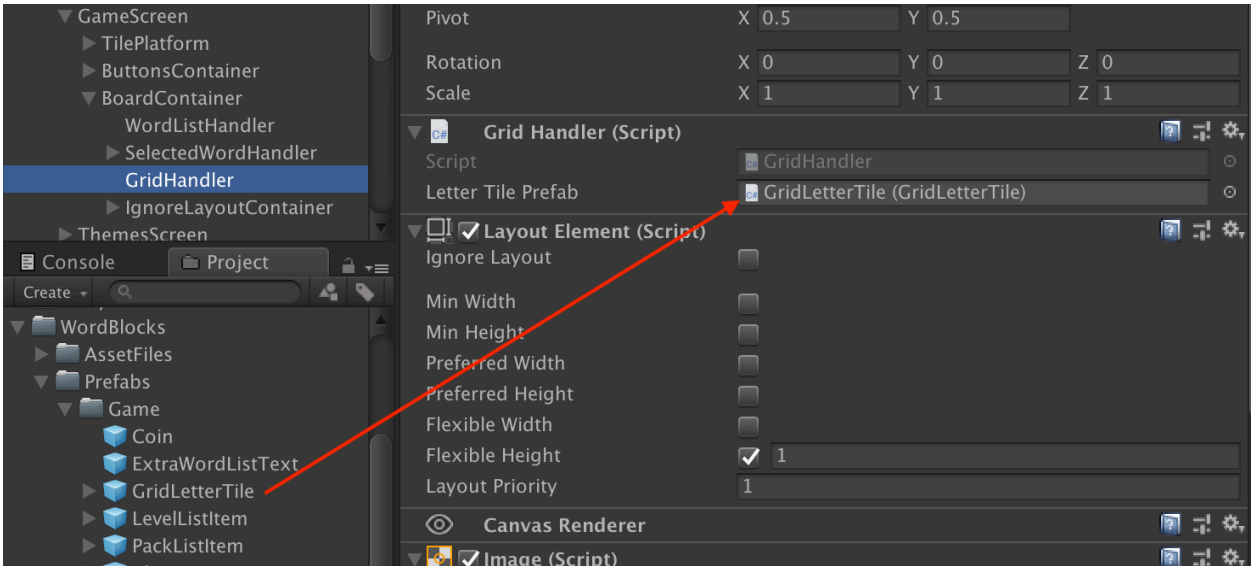
Extra Words Text

The extra words text is instantiated by the ExtraWordsPopup. The **Extra Words Text Prefab** is the prefab that is instantiated when the popup populates the list of found extra words. The ExtraWordListText prefab in the Prefabs folder is the one used in the asset:



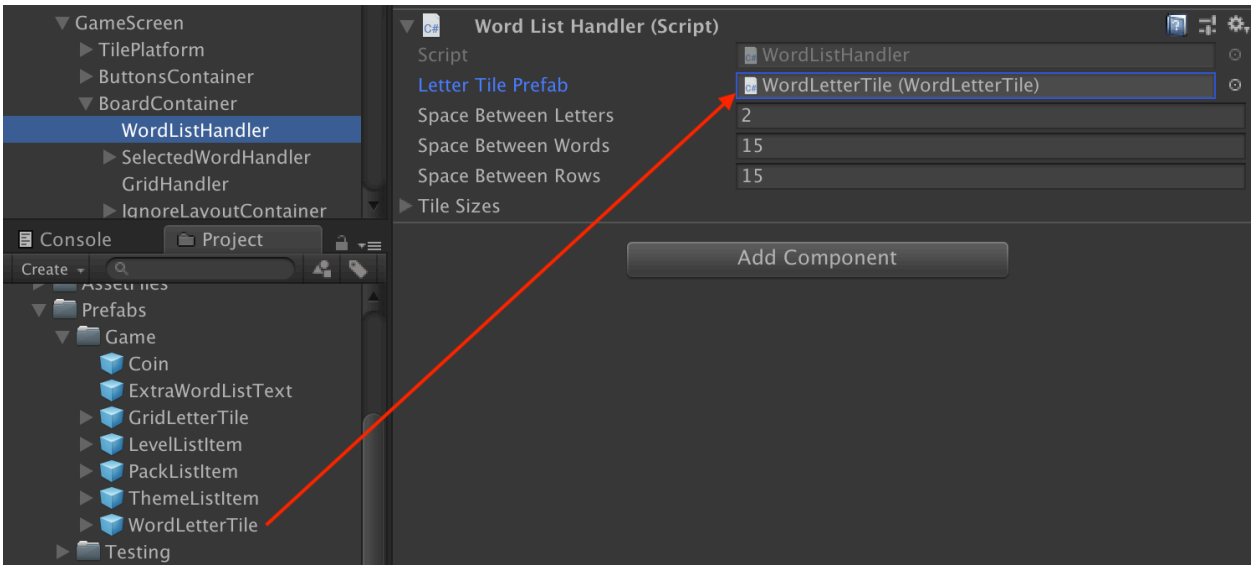
Grid Letter Tiles

The **GridHandler** is responsible for instantiating and positioning the tiles that the player selects. The **Letter Tile Prefab** is the prefab that is instantiated for each cell in the grid. The GridLetterTile in the Prefabs folder is the prefab used in the asset:

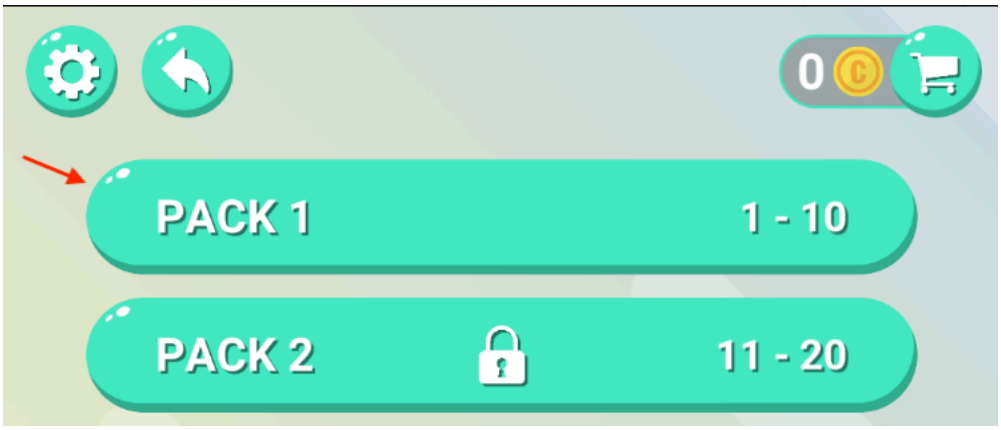


Word Letter Tile

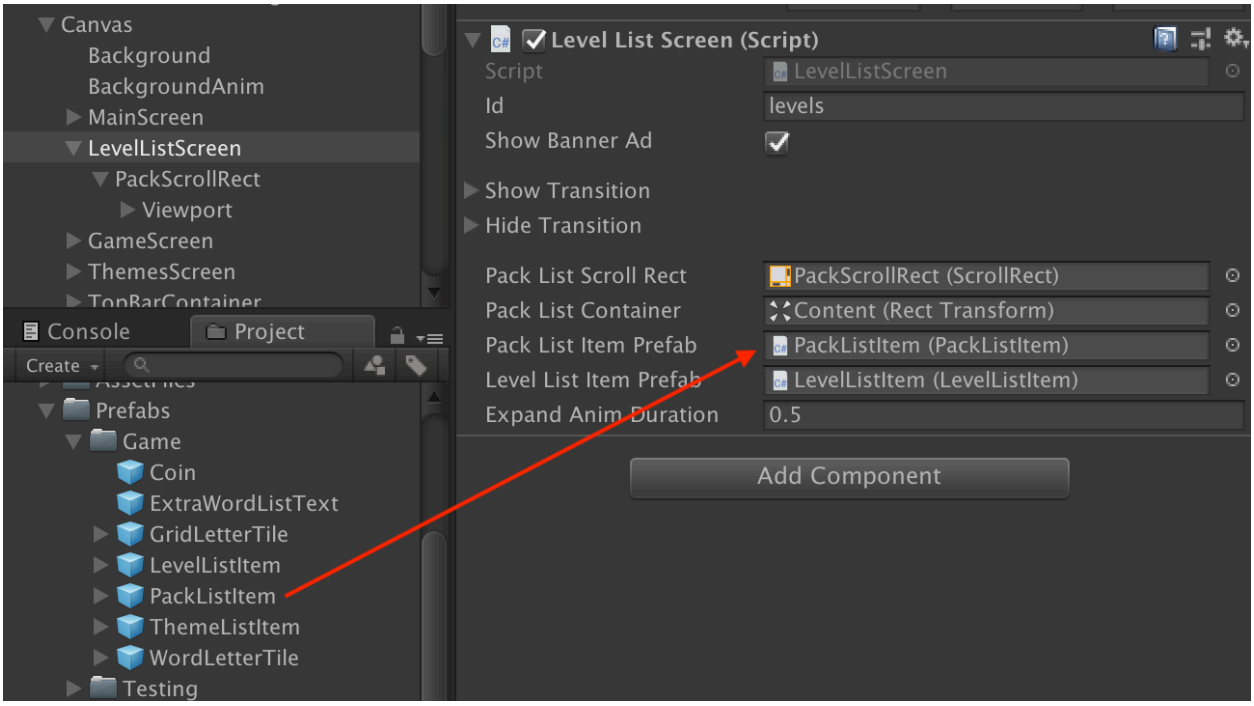
The **WordListHandler** is responsible for instantiating and positioning the tiles for the words at the top of the game screen. The **Letter Tile Prefab** is the prefab that is instantiated for each letter in each word. The WordLetterTile in the Prefabs folder is the prefab used in the asset:



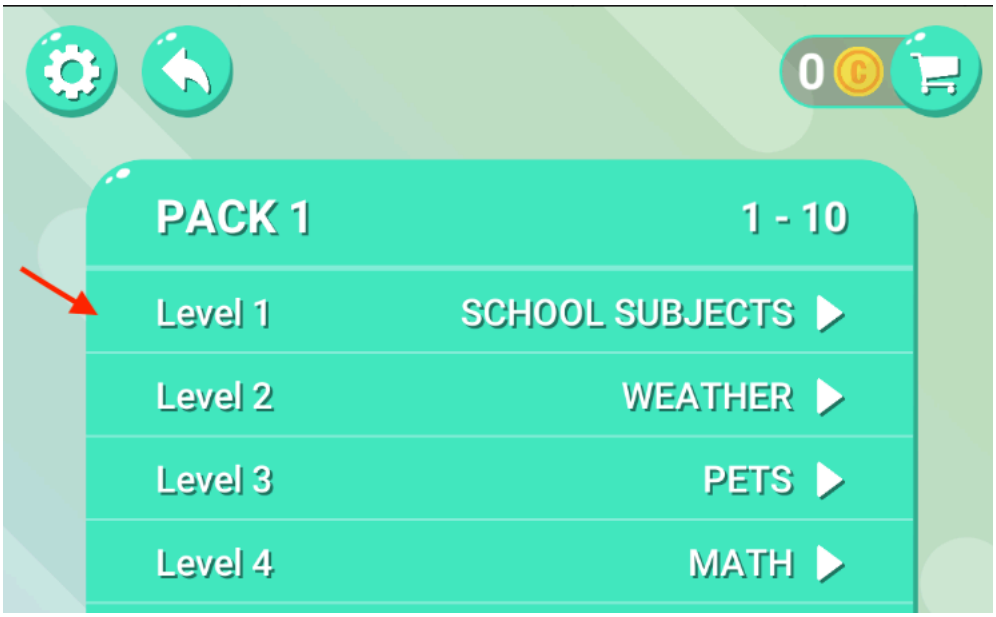
Pack List Item



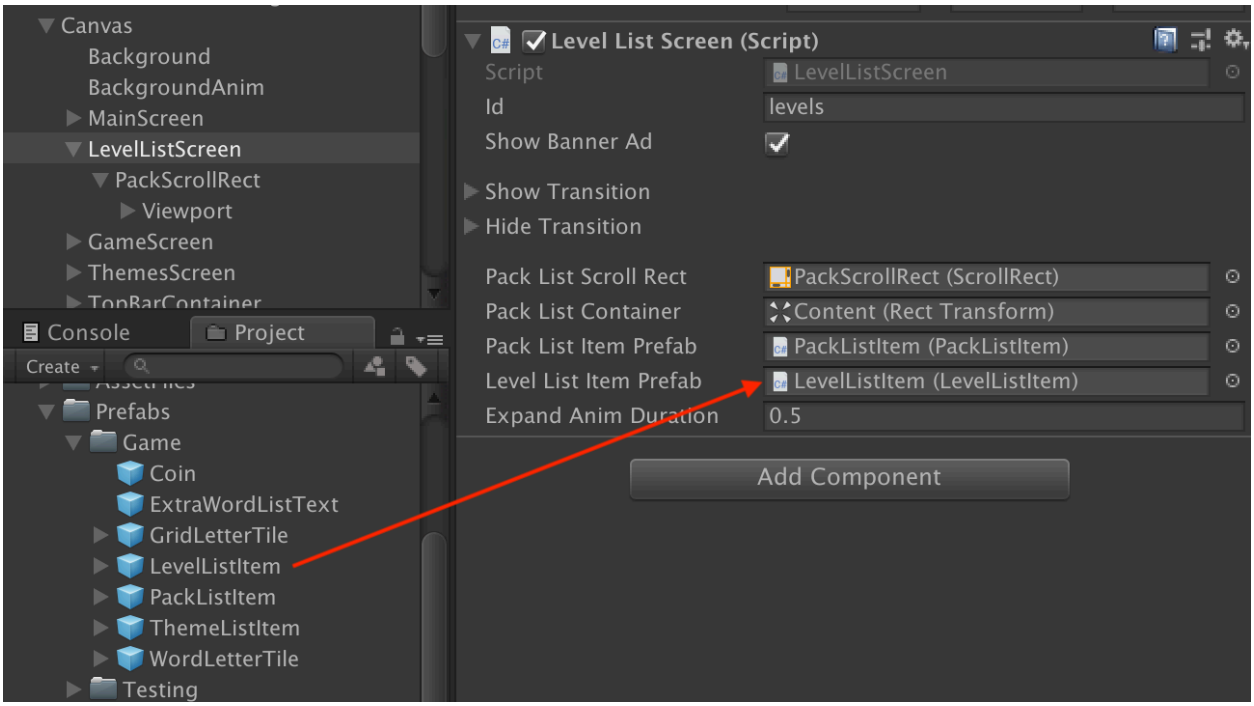
The **LevelListScreen** is responsible for instantiating the pack list items. The **Pack List Item Prefab** is the prefab that is instantiated for each Pack Info set on the GameManager. The PackListItem in the Prefabs folder is the prefab used in the asset:



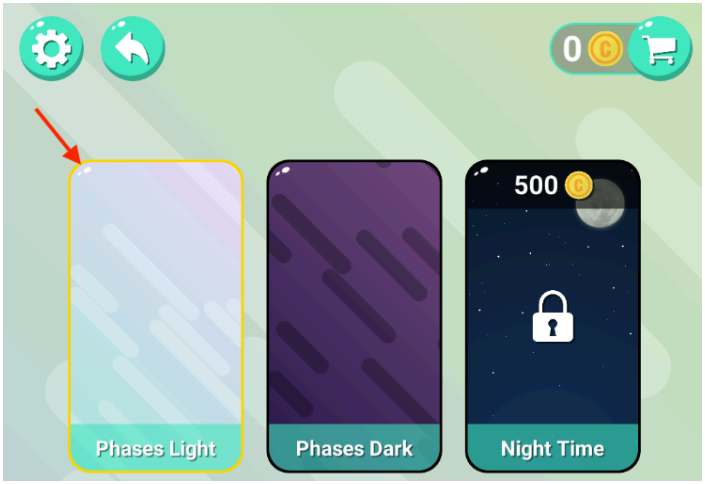
Level List Item



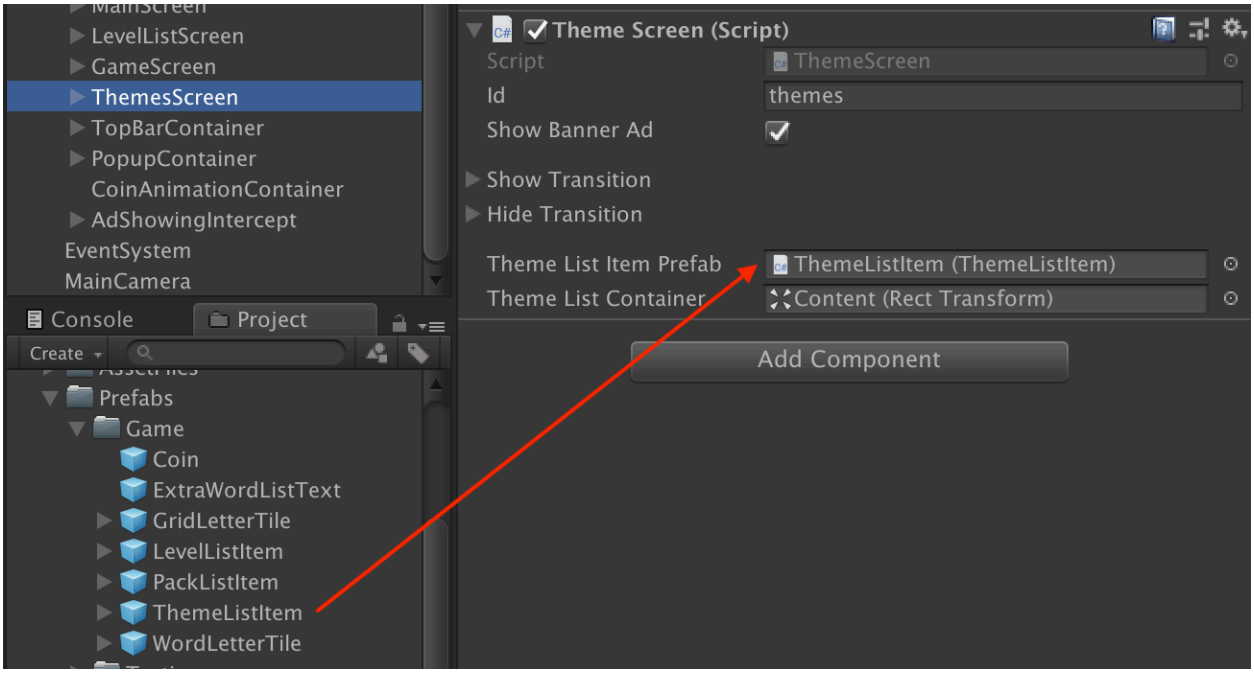
The **LevelListScreen** is responsible for setting up the ObjectPool which is used by individual PackListItems to instantiate copies of the **Level List Item Prefab**. The LevelListItem in the Prefabs folder is the prefab used in the asset:



Theme List Item



The **ThemesScreen** is responsible for instantiating the theme list items. The **Theme List Item Prefab** is the prefab that is instantiated for each Theme set in the ThemeManager. The ThemeListItem in the Prefabs folder is the prefab used in the asset:



The Sprite used as the image for the theme list item is set on the Theme in the ThemeManager: