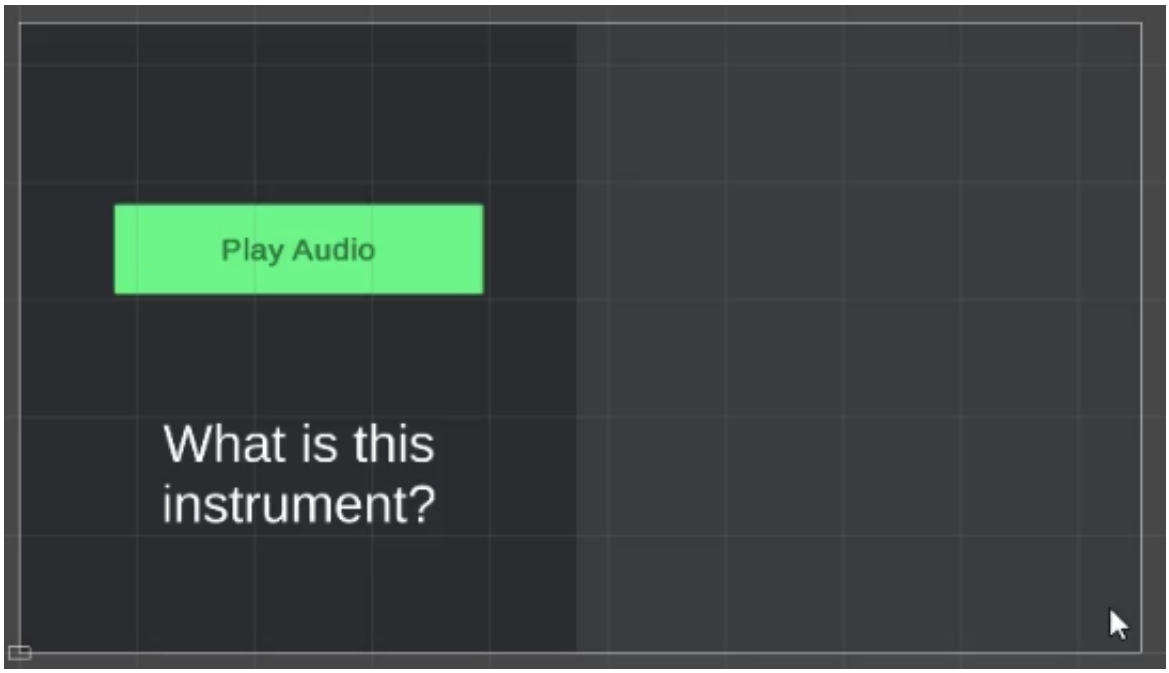
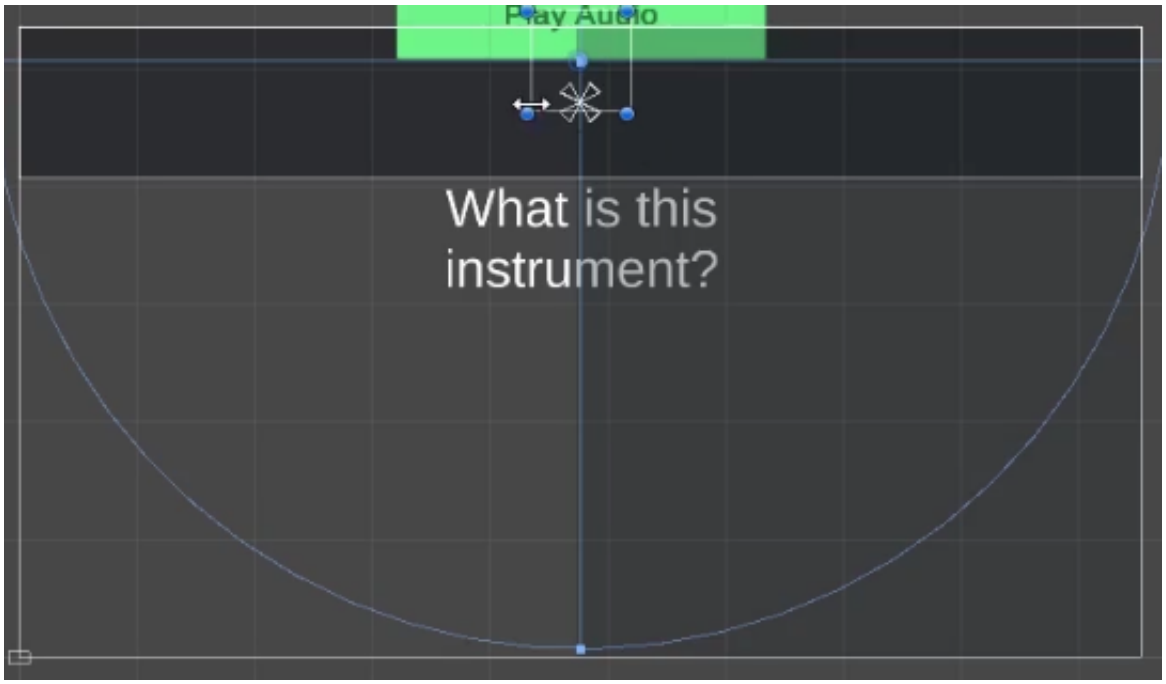
Welcome to **Zenva's Create a Spanish-Teaching Quiz** course. In this short course, we will create a Spanish language quiz in **Unity** based on the **Super Quiz game** made in the previous course. We will also add in nifty features such as the ability to use **sprites** for the answers and the ability to play sounds based on whether the answer is correct or incorrect. With this lesson, we will focus on designing our **UI**.

Changing the Question Panel UI

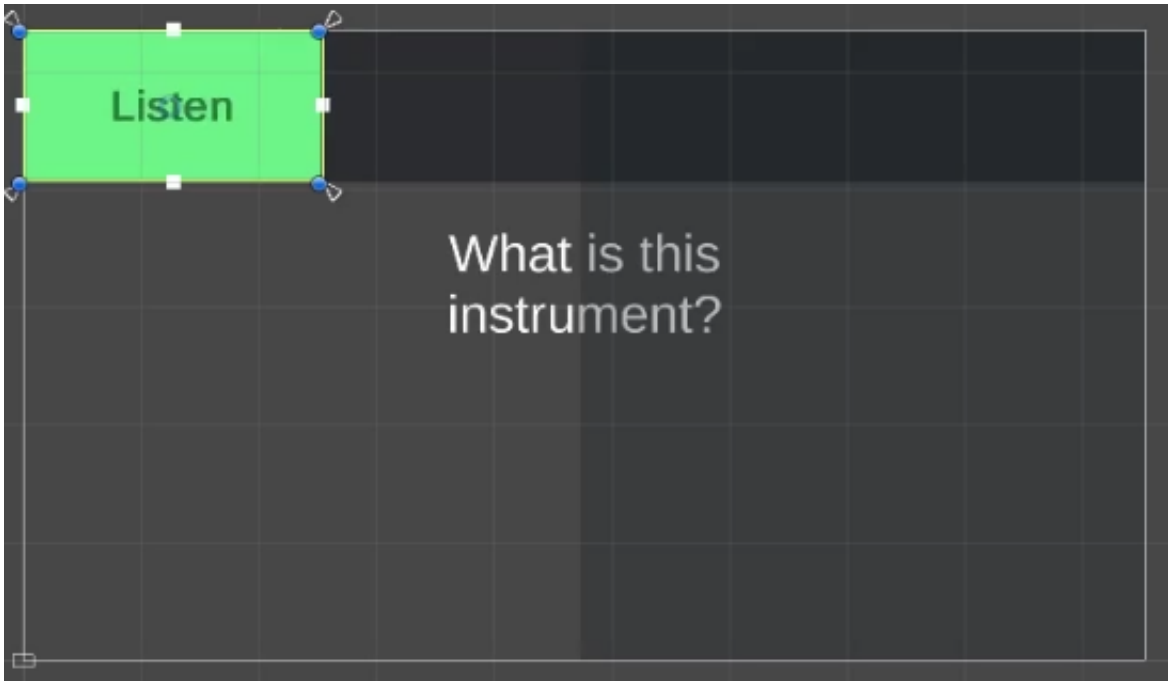
To begin, make sure you have the **Super Quiz game** from the last course open in **Unity**, as we will be reusing this template. We're going to start working in our **Question Scene** first and adjust the **UI**.

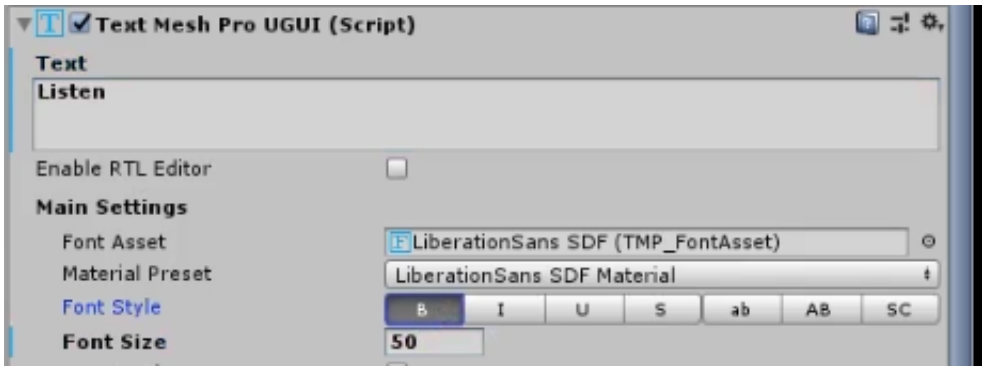


Unlike our previous **game**, we want our **question** to appear as a header at the top of the **canvas** and put all our answers horizontally at the bottom. Thus, our first step is to take the **Question panel** and resize it so it's on top fully spread across. Our **height** for the **Question panel** will be **173.1** if you'd like to have a similar size.

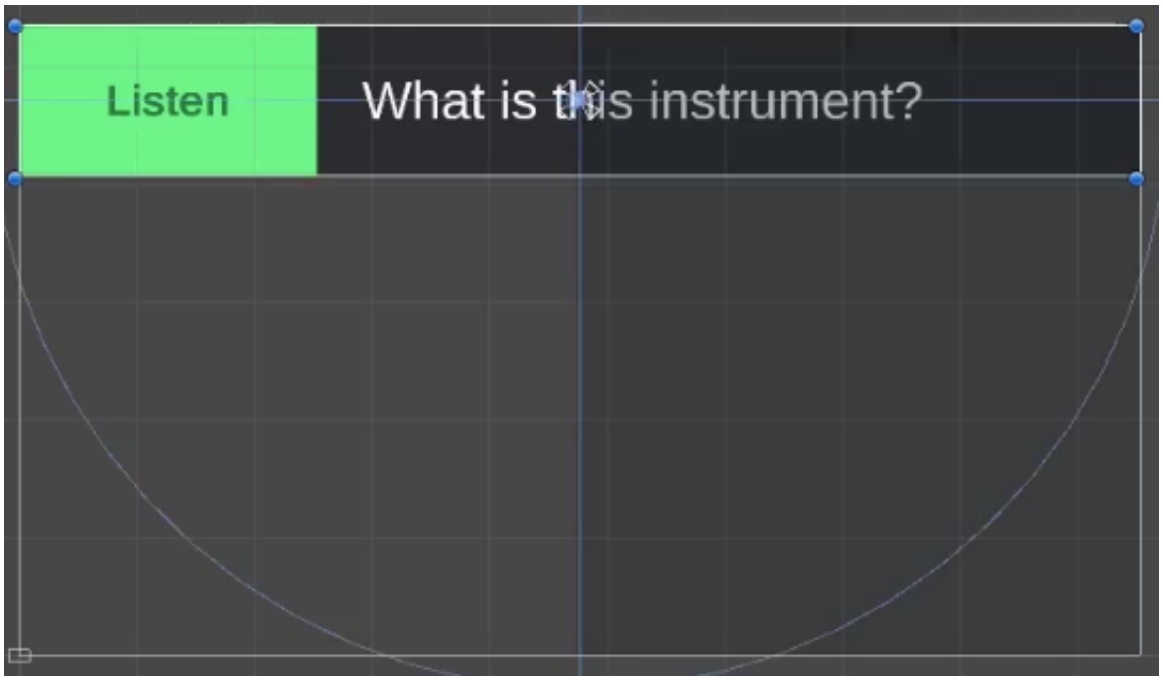


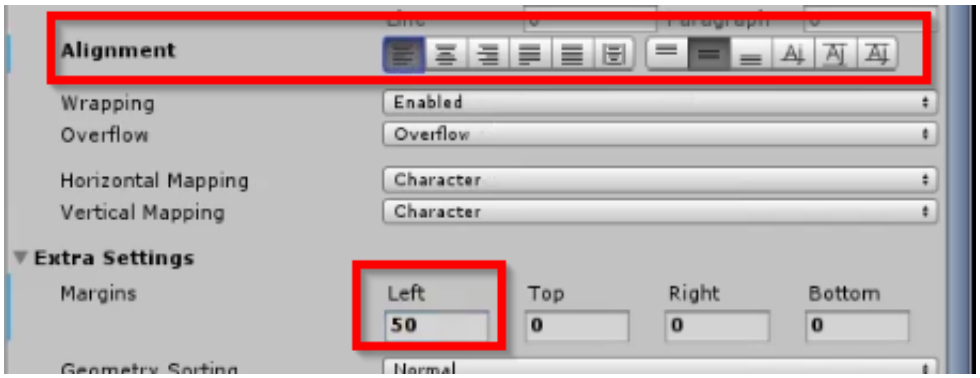
In a related fashion, we want to take the **Audio With Caption parent** object and resize it to match the entire **Question Panel**. Once done, we can move our **audio button** to the left side of the panel and resize it to fully stretch to the top and bottom of its panel. For good measure, we're going to change the **button's text** to be **Listen**, make it **bold**, and change the **font size** to **50**.



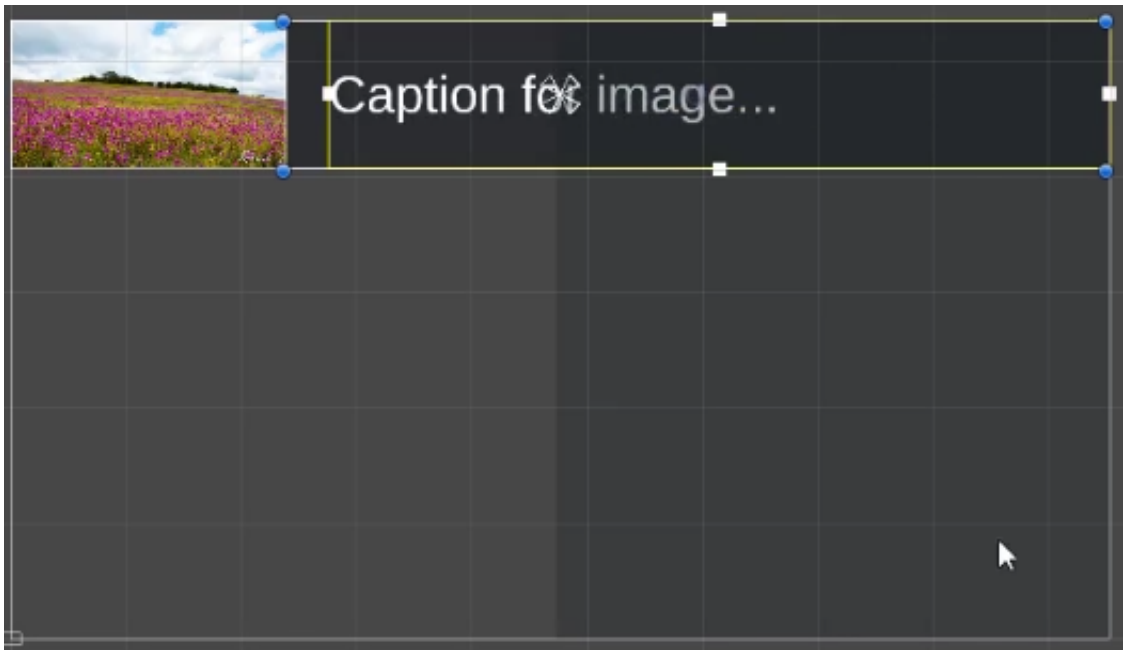


Last but not least, we want to take the **caption text** for the **audio** and snap it next to the button. From the **Extra Settings** in the **Inspector**, we can add **50** to the **left margin** and change the text's alignment to be to the **left** as well.

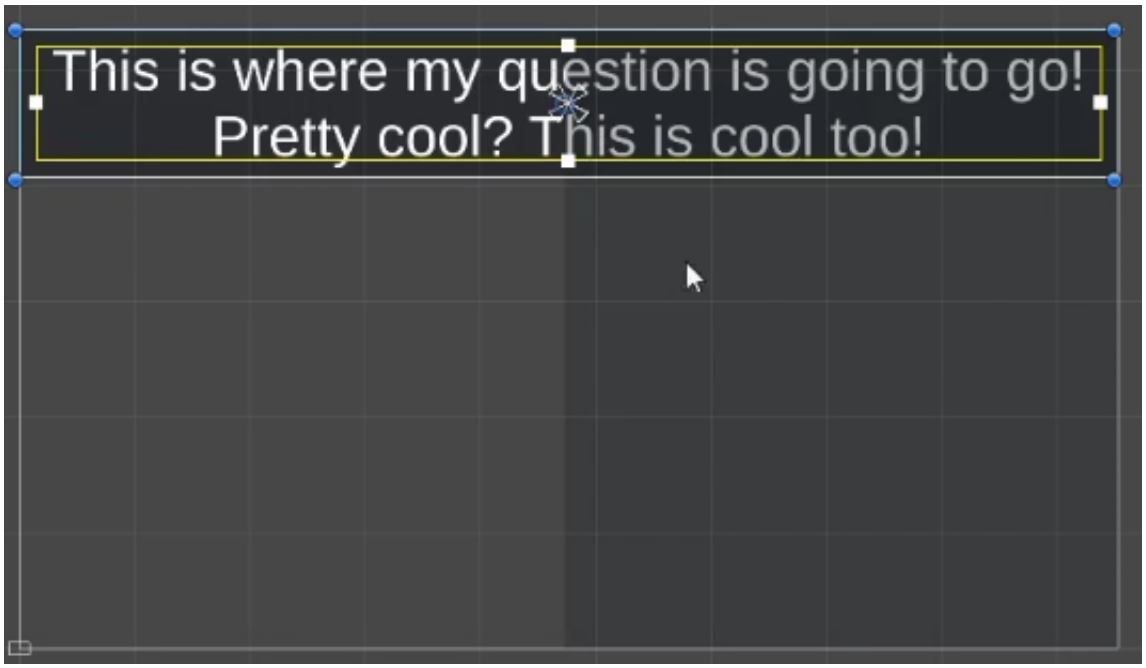


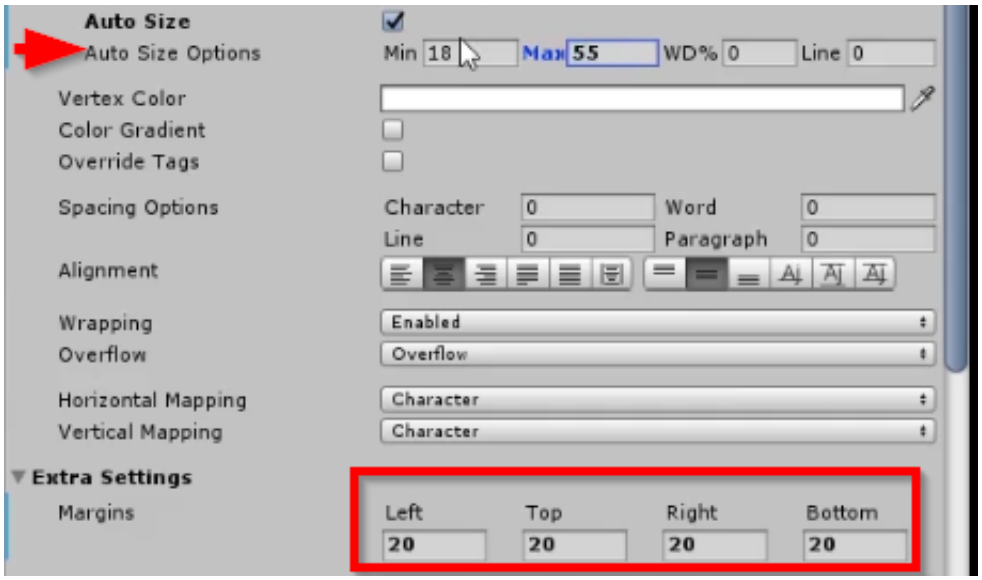


After these changes, we can **turn off** the **Audio With Caption parent** and **enable** the **Image With Caption parent**. Following this, we will repeat the process above by resizing the **Image With Caption panel** to match the **Game panel**, moving and resizing the **Image**, and moving and resizing the **caption text**.



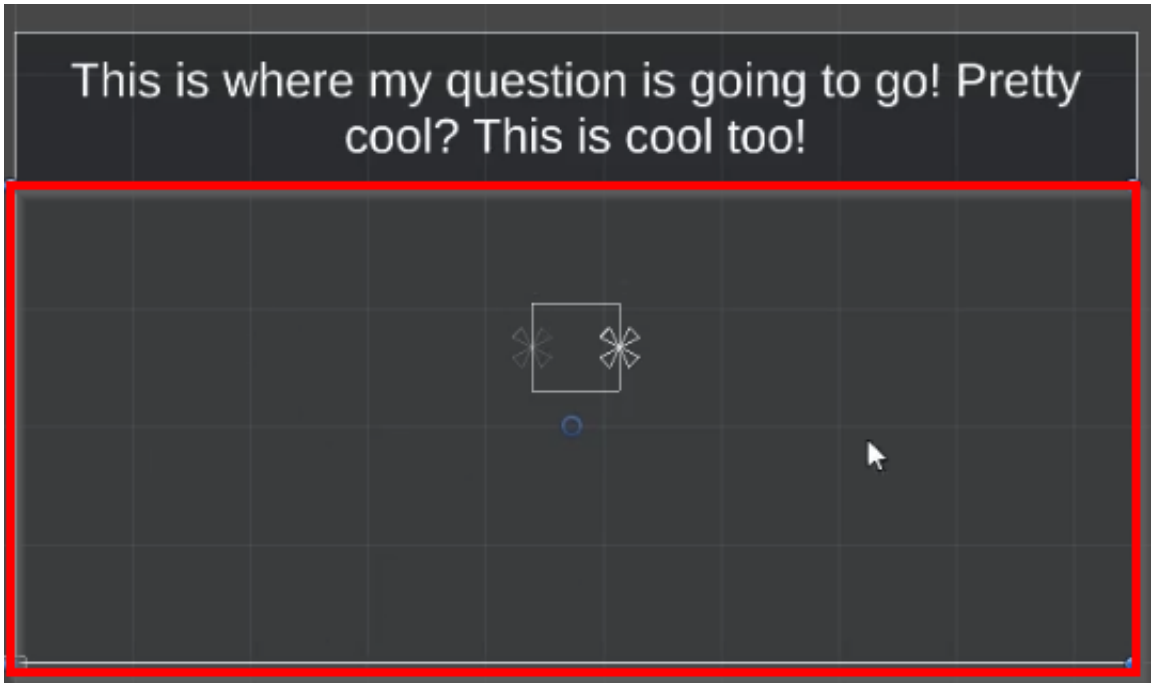
Finally, we can take our **Text Question** object and resize it to match the **Question panel** as well. For this one, we will add **Margins** of **20** to all the sides. We also want to make the **Max Auto Size option 55** so the text resizes nicely.



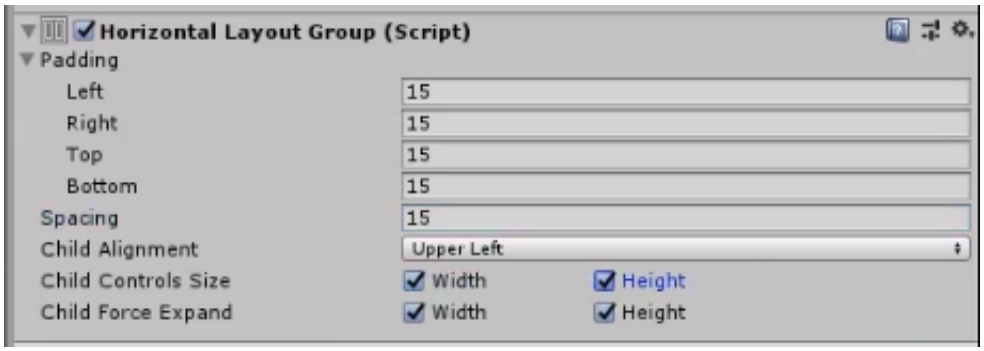


Changing the Answer Panel UI

We'll next alter our **Answers panel**. To start off, we want to take the **panel** itself and resize it so it takes up the entire rest of the **canvas** under the **header** we created for the **question**.

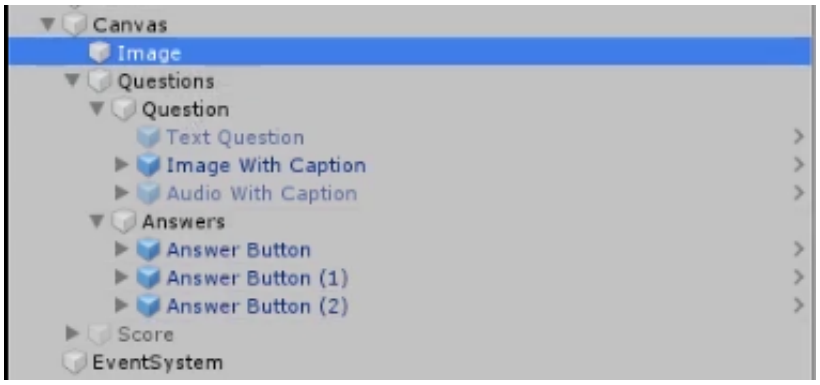


At the moment our **Answers panel** uses the **Vertical Layout Group** and displays any prefabs vertically. We want them to display **horizontally** for this version of our quiz game. As such, we're going to **remove** our **Vertical Layout Group Component** and replace it with a **Horizontal Layout Group Component**. Once added, we can add **15** to all the spacing and **padding** and make sure **Child Controls Size** is **enabled**.



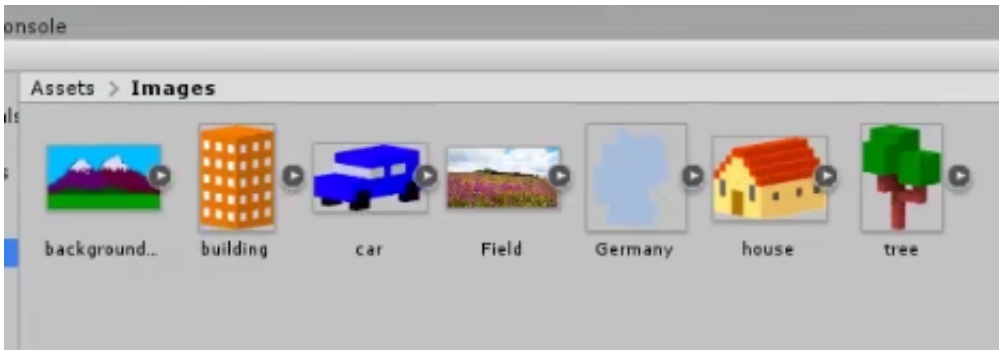
Canvas Background

With the **Answers** squared away, we want to add a **background image** to our **Canvas**. By right-clicking on the **Canvas**, we can select **Image** from the **UI menu**. After its creation, we want to make sure to move the **Image** to the top of the **Canvas Hierarchy** so that is displays behind all our other panels. We also want to **resize** the **image** so it takes up all of the **canvas**.

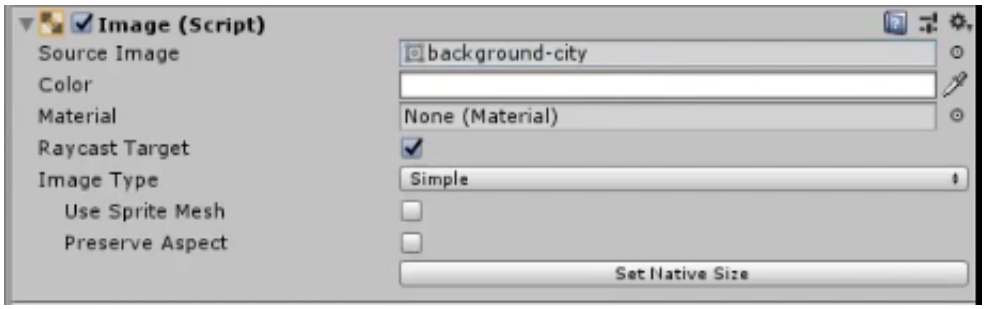


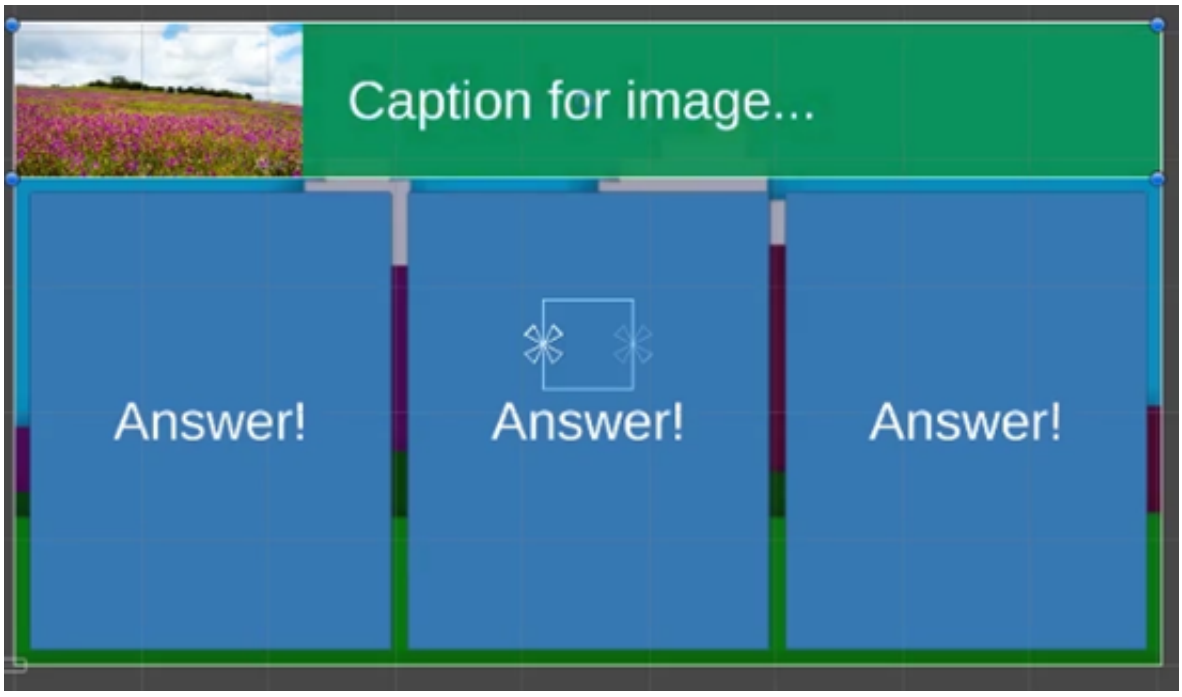
At this point, you should make sure to **download** the **Course Files** for this project. From this **download**, you can head into **Super Spanish Quiz/Assets/Images** and locate several images we will be using for this project. We want to take the **images** and **drag** them into our **Assets/Image**

folder in our **Unity project**.



Now that we have our **images**, we want to return to our **background image** for the **canvas** and assign the **background-city sprite** to it. This will in turn give us the background we're looking for.





At this point, we encourage you to play around with the panel colors so that they look nice against the background. In the next lesson, we're going to work on our answer options.

In this lesson, we're going to set up a new **answer structure** so we can use sprites or buttons for the **answers**.

Creating the Answer Struct

To begin, we want to open up our **Question script** up in **Visual Studio**. Similarly to how we created a **QuestionSet struct** in the **QuestionDatabase**, we want to create one for our **answers**. Outside of the **Question class**, we can create our **Answer struct**. This **struct** will have three properties: the image, the answer key, and the answer button text which is used for the UI. We also want to make sure these elements are **serialized** to the **Inspector**.

```
[System.Serializable]
public struct Answer
{
    public Sprite answerSprite;
    public string answerKey;
    public string answerButtonText;
}
```

Following this, we want to look to our **Question class** and change our **answerChoices** variable to be an **Array** of our **Answer struct** items. This way we can add all the elements we need in the **Inspector** to each question.

```
public class Question
{
    public enum QuestionType { Text = 0, ImageWithCaption = 1, Audio = 2 }
    public QuestionType questionType;
    public string questionText;
    public Sprite questionImage;
    public AudioClip questionAudio;
    public string correctAnswerKey;
    public Answer[] answers;
    //public string[] answerChoices;
}
```

Datatype Fixes

However, since we changed the **datatype**, we need to open up the **QuestionUI script** to fix an issue. In our **foreach loop**, our **answers** are no longer **strings**, so we need to change it so it's **looping** through our **Answer object**.

```
foreach(Answer answer in question.answers)
{
    Transform answerButtonInstance = Instantiate(answerButton, answerPanel).transform
;
    answerButtonInstance.GetComponent<AnswerButton>().SetAnswerButton(answer);
}
```

Once we make this change, however, the **script** flags the fact that the button is looking for a

string and not an **Answer object** to display its **text**. Since we only want this text to even display when there is no image, we need to do some major modification to our **AnswerButton script**.

In the **AnswerButton script**, we need to first change our **private answer variable** to be of an **Answer object** type.

```
private Answer answer;
```

Next, we want to modify our **SetAnswerButton function** to take in an **Answer object** instead of a **string**. Since we now have multiple elements stored as our **answer** that's passed in, we also have to make sure the **text component** grab knows to use the **answerButtonText**.

```
public void SetAnswerButton(Answer answer)
{
    this.answer = answer;
    transform.GetChild(0).GetComponent<TMPro.TextMeshProUGUI>().text = answer.answerB
uttonText;
}
```

After this, we also need to change our **Start method** to use the **answerKey** when it performs **CheckAnswer**, as this is the only element from the **Answer object** it needs to check.

```
void Start()
{
    game = FindObjectOfType<Game>();
    GetComponent<UnityEngine.UI.Button>().onClick.AddListener(() => game.CheckAnswer(
answer.answerKey));
}
```

Dynamic Display

The next feature we want to add is the ability for our **answer button** to only display text if there is no **image** to display. Within the **SetAnswerButton function**, we can do this by adding an **if statement** to the mix. However, in order to access the components we need, we need to get access to the **button** (as we did in the **Start method**). Since we will do this several times, we're going save it as a **variable**. First, at the top of the file though, we're going to add in a **using** statement for the **UI namespace**.

```
using UnityEngine.UI;
```

Following this, we can then add the **variable** that will save our **button** reference.

```
private Button button;
```

Finally, using our **Start method**, we can assign this **variable** using **GetComponent** to find the **UI**

button. Since this variable will be saved, we can also use it to where we **add** the **listener** to shorten our code some.

```
void Start()
{
    button = GetComponent<Button>();
    game = FindObjectOfType<Game>();
    button.onClick.AddListener(() => game.CheckAnswer(answer.answerKey));
}
```

At last, we can create our **if statement** in **SetAnswerButton**. With this **if statement**, we will check if the **answer sprite isn't null**. Since this code is subsequently run when there is an **image**, then, we can set the **button's image sprite** to be our **answer sprite**.

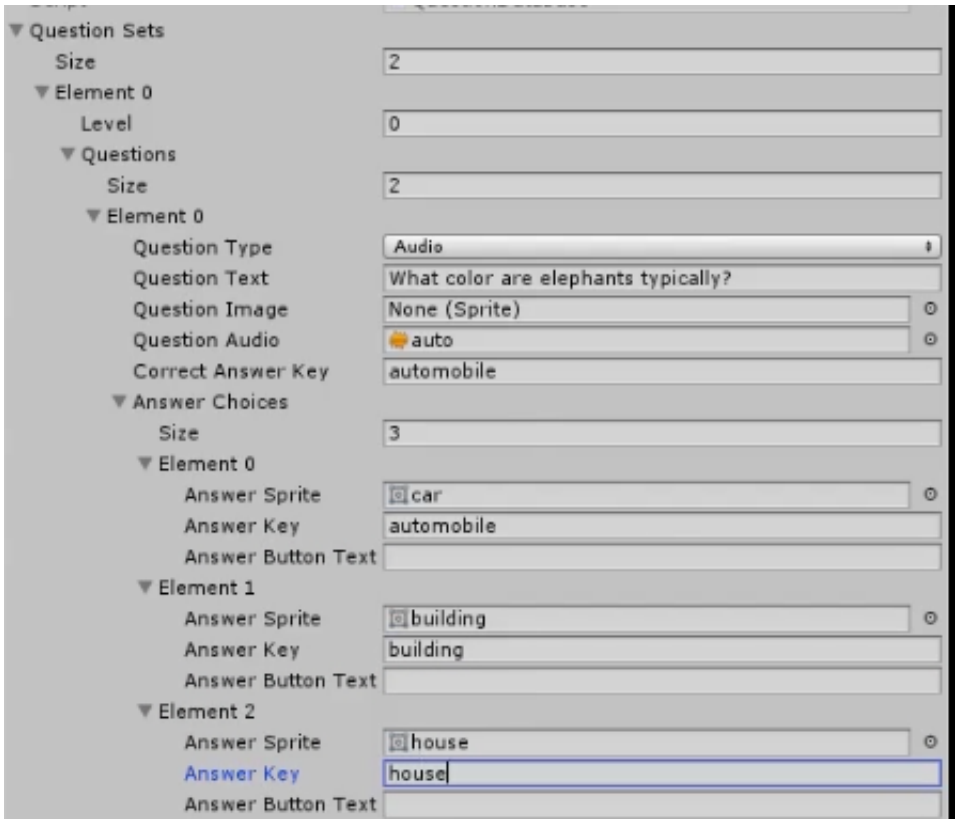
```
public void SetAnswerButton(Answer answer)
{
    this.answer = answer;
    transform.GetChild(0).GetComponent<TMPro.TextMeshProUGUI>().text = answer.answerB
uttonText;
    if (answer.answerSprite != null)
    {
        button.image.sprite = answer.answerSprite;
    }

}
```

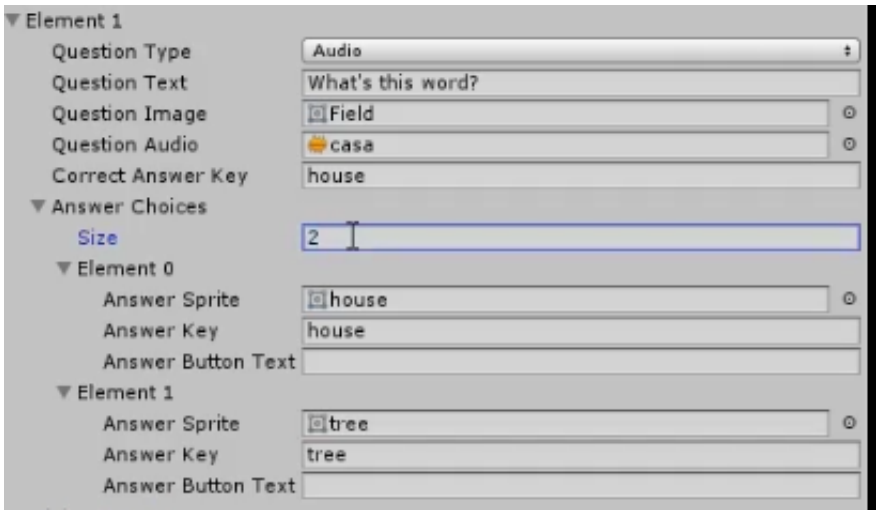
Setting up Answers in Unity

Let's now do some testing in **Unity**. First off, from our **Course Files**, we want to locate the audio files in the **Super Spanish Quiz/Assets/Audio folder**. After which, we can drag those files into our **Assets/Sounds** folder.

After, we want to open up our **Questions set database** to look at our **questions**. For the first question, we will change the **Question Type** to **Audio** and add the **auto audio** to the **Question Audio** option. To supplement this, we can also make the **correct answer key** be **automobile**. For our **Answer choices**, we will make them be **3** in size. You can add whatever options you want, but at least one should use the **car sprite** and have the **automobile answer key**.



With the second question in the **set**, we will make another **audio question** using the **casa audio**. For the question we will make it **What's this word**? And have the **answer key** be **house**. Once again, you can add in any answer choices you wish, but one should use the **house sprite** and have the **answer key** of **home**.



If you play test the game from the main menu, however, you will see we get an **error**! In the next lesson, we will fix this and get our answers fully functioning!

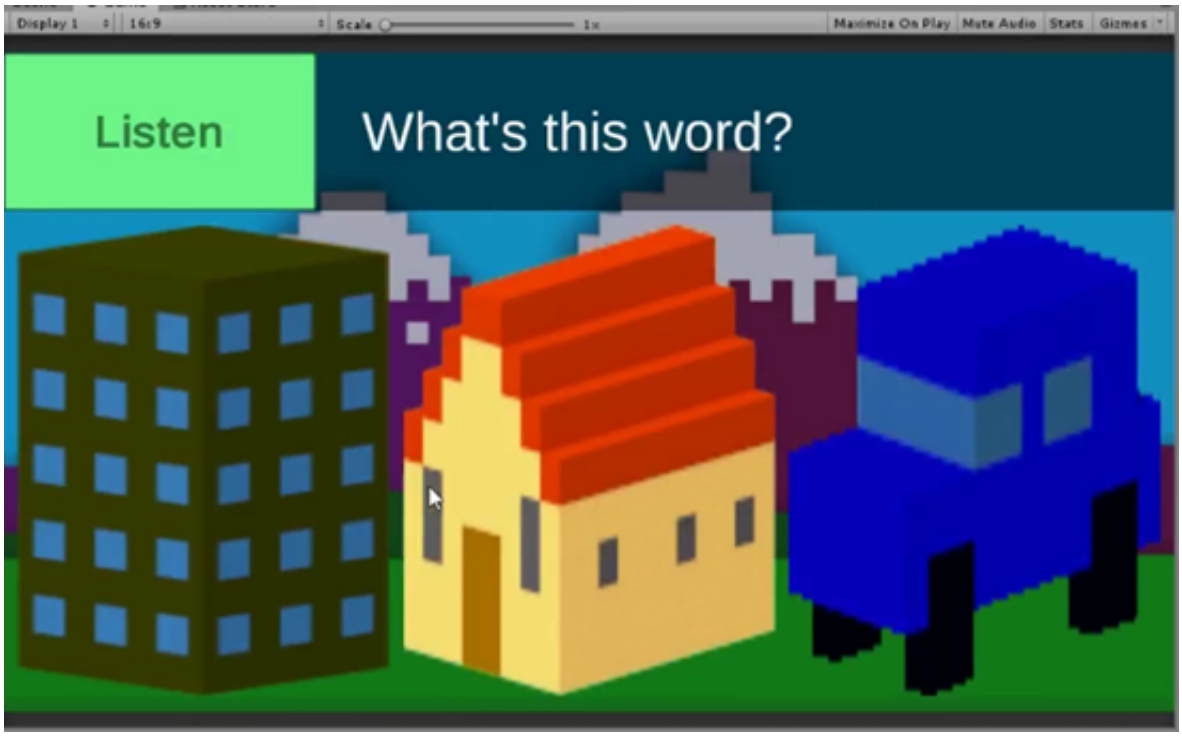
In this lesson, we're going to finish up our answers set up and fix some errors in our code from last lesson.

Error Fix and Testing

In regards to our initial error we ended the last lesson on, this was a **null reference exception**. When we get up our **GetComponent** for the **button**, we put it in **Start**. Unfortunately, the **answers** were **initialized** before this ran, causing the error. To make sure that **GetComponent** runs before this happens, we can change this **method** to **Awake** so it's the first thing run.

```
void Awake()
{
    button = GetComponent<Button>();
    game = FindObjectOfType<Game>();
    button.onClick.AddListener(() => game.CheckAnswer(answer.answerKey));
}
```

Back in **Unity**, we can test out our game now (make sure to delete any **prefabs** if you pulled them onto the scene for testing though)! This time, our quiz should play and render as expected.



Button Image Adjustments

However, as you might notice our images are far too large and stretched. While in the **Answer Button's Image option** we can change the **Image Type** to **simple** to fix this, we still need to use **sliced** for the regular buttons. We also want to be able to change our hover color as well. As such, we need to head back to our **Answer Button script** and add a few things to our **SetAnswerButton if statement**.

Within this **section**, we can first change the color of our button first. Unfortunately, **Unity** doesn't let

us change the **colors property** directly for individual **color types**. In order to apply the change we want, we first need to **create** a **color block variable**. From there, we can change the **normal color** in that **color block** to **white**. Finally, we can then use that **color block** to set the **buttons color**.

```
public void SetAnswerButton(Answer answer)
{
    this.answer = answer;
    transform.GetChild(0).GetComponent<TMPro.TextMeshProUGUI>().text = answer.answerB
uttonText;
    if (answer.answerSprite != null)
    {
        button.image.sprite = answer.answerSprite;
        ColorBlock colorBlock = button.colors;
        colorBlock.normalColor = Color.white;
        button.colors = colorBlock;
    }

}
```

Next, we can change the **Image Type**. This is a lot simpler since we can directly target the **Image Type property** and change it to **Simple**. This type change also gives us access to the **preserve aspect property**, which we can set to **true** so that our images don't stretch.

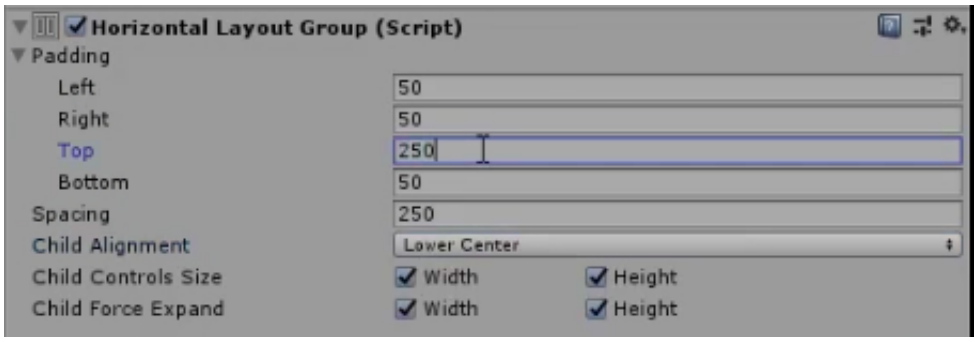
```
public void SetAnswerButton(Answer answer)
{
    this.answer = answer;
    transform.GetChild(0).GetComponent<TMPro.TextMeshProUGUI>().text = answer.answerB
uttonText;
    if (answer.answerSprite != null)
    {
        button.image.sprite = answer.answerSprite;
        ColorBlock colorBlock = button.colors;
        colorBlock.normalColor = Color.white;
        button.colors = colorBlock;

        button.image.type = Image.Type.Simple;
        button.image.preserverAspect = true;
    }

}
```

Once you **save** the file, you can head back into **Unity** and test again. This time, our images should look much better.

They're still a bit bigger than we would like though, but we can fix this more easily now. By selecting the **Answers Panel object** in the **Hierarchy**, we can look to our **Horizontal Layout Group Component** and change our **spacing** and **padding**. For the **padding**, we will put **250** on the **Top** and **50** for all the other sides. We will make our **spacing 250**. To place the objects more towards the ground of our background, we will change the **Child Alignment** to **Lower Center**.



We can once again play test the game from the **Main Menu**. Our quiz game should function, and our images should have much better placement.

In the next and final lesson, we will add some more features and finishing touches to our game.

In this final lesson, we're going to add the ability for questions to have multiple correct answers. Additionally, we're going to add sound effects for when an answer is right or wrong.

Adding Optional Correct Answers

Since we want multiple correct answers to be possible, we want to first head into the **Question script** and change the **Question class**. For the **correctAnswerKey variable**, we want to change it from **string** to **string[]** so an **array** is possible.

```
public string[] correctAnswerKeys;
```

However, we don't have a way to check the **arrat**, so we next need to open up our **Game script** and scroll down to our **CheckAnswer method**. Instead of one **correct answer**, we need to check for multiple. Thus, we need to **loop** through our **array** with a **foreach loop**. We can move our **if statement** to be within our **loop** so that we can check if the **answer** is equal to the **answerKey** that's being looked at in the current **iteration**. **If** it is, we can add points to our score and **break** out of the loop.

```
public void CheckAnswer(string answer)
{
    foreach (string answerKey in currentQuestion.correctAnswerKeys)
    {
        if (answer == answerKey)
        {
            correctAnswers++;
            Debug.Log("That's correct!");
            break;
        }
    }
    ClearAnswers();
    NextQuestion();
}
```

Adding Sound Effects

As mentioned, we also want to add sound effects for correct and wrong answers. We can't put this in our **for loop**, as this would cause the incorrect sound to play or cause multiple sounds to play. What we can do, however, is extend our **method**.

Right above our **loop**, we can create a **boolean variable** that tracks whether the **answer** is **correct** or not. If we **initialize** it as **false**, we can then change the **boolean** to be **true** within the **scope** of our **loop's if statement**. Outside of the **loop**, then, we can use an **if else statement** that decides which **sound clip** to play.

```
public void CheckAnswer(string answer)
{
    bool correct = false;
    foreach (string answerKey in currentQuestion.correctAnswerKeys)
    {
        if (answer == answerKey)
        {
```



```
            correctAnswers++;
            correct = true;
            Debug.Log("That's correct!");
            break;
        }
    }
    if (correct)
    {

    }
    else
    {

    }
    ClearAnswers();
    NextQuestion();

}
```

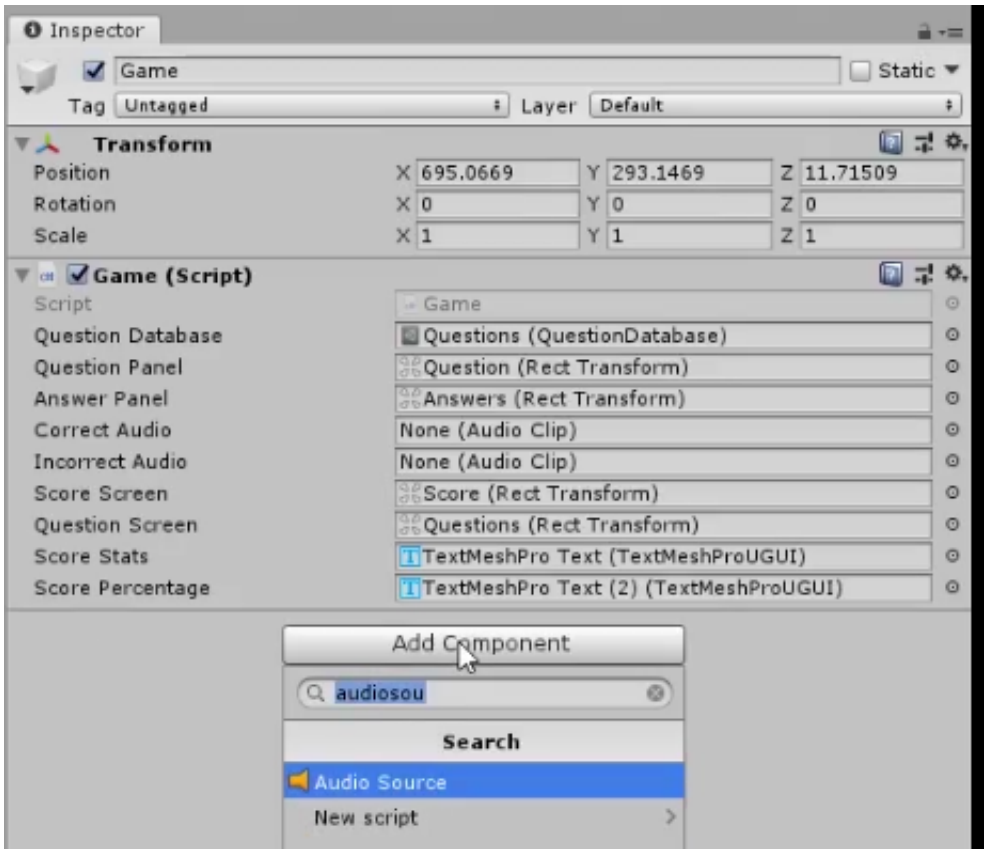
In order to play sounds, though, we need both an **audio source** and the **sounds** we want to play. At the top of the **Game script**, we will add **variables** for these. Since we need to assign the **sound clips** in the **Inspector**, we need to make sure they're **serialized**.

```
[SerializeField]
private AudioClip correctSound, incorrectSound;
private AudioSource audioSource;
```

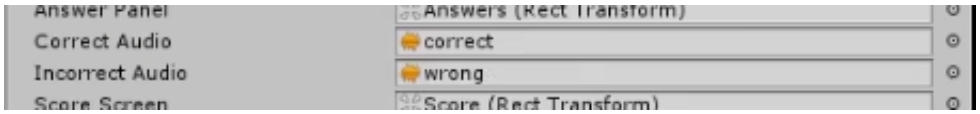
However, our **audio source** will be on the **Game object**, we can simply use our **Start method** to get that **Component**.

```
void Start()
{
    audioSource = GetComponent<AudioSource>();
    level = PlayerPrefs.GetInt("level", 0);
    LoadQuestionSet();
    UseQuestionTemplate(currentQuestion.questionType);
}
```

Back in **Unity**, we can now select the **Game object** and use the **Add Component** button in the **Inspector** to **add** an **audio source**.



We will also assign the **correct audio clip** from the assets to the **Correct Audio field** and the **wrong audio clip** to the **Incorrect Audio field**.



Once added, we can then head back to **Visual Studio** and finish our **if else statement** in the **Game script**. When the **answer** is correct, we will **play** the **correct sound once** from the **audio source**. **Else**, we will **play** the **wrong sound once**.

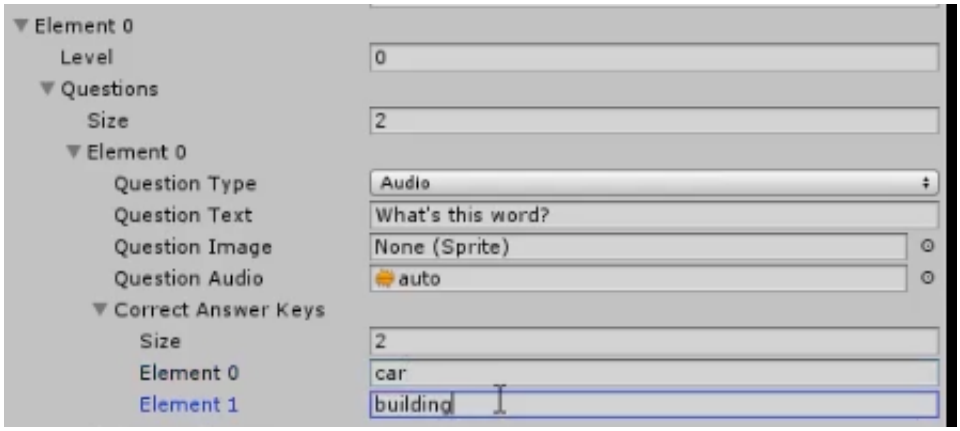
```
public void CheckAnswer(string answer)
{
    bool correct = false;
    foreach (string answerKey in currentQuestion.correctAnswerKeys)
    {
        if (answer == answerKey)
        {
            correctAnswers++;
            correct = true;
            Debug.Log("That's correct!");
            break;
        }
    }
    if (correct)
    {
        audioSource.PlayOneShot(correctSound);
    }
    else
```



```
    {
        audioSource.PlayOneShot(incorrectSound);
    }
    ClearAnswers();
    NextQuestion();

}
```

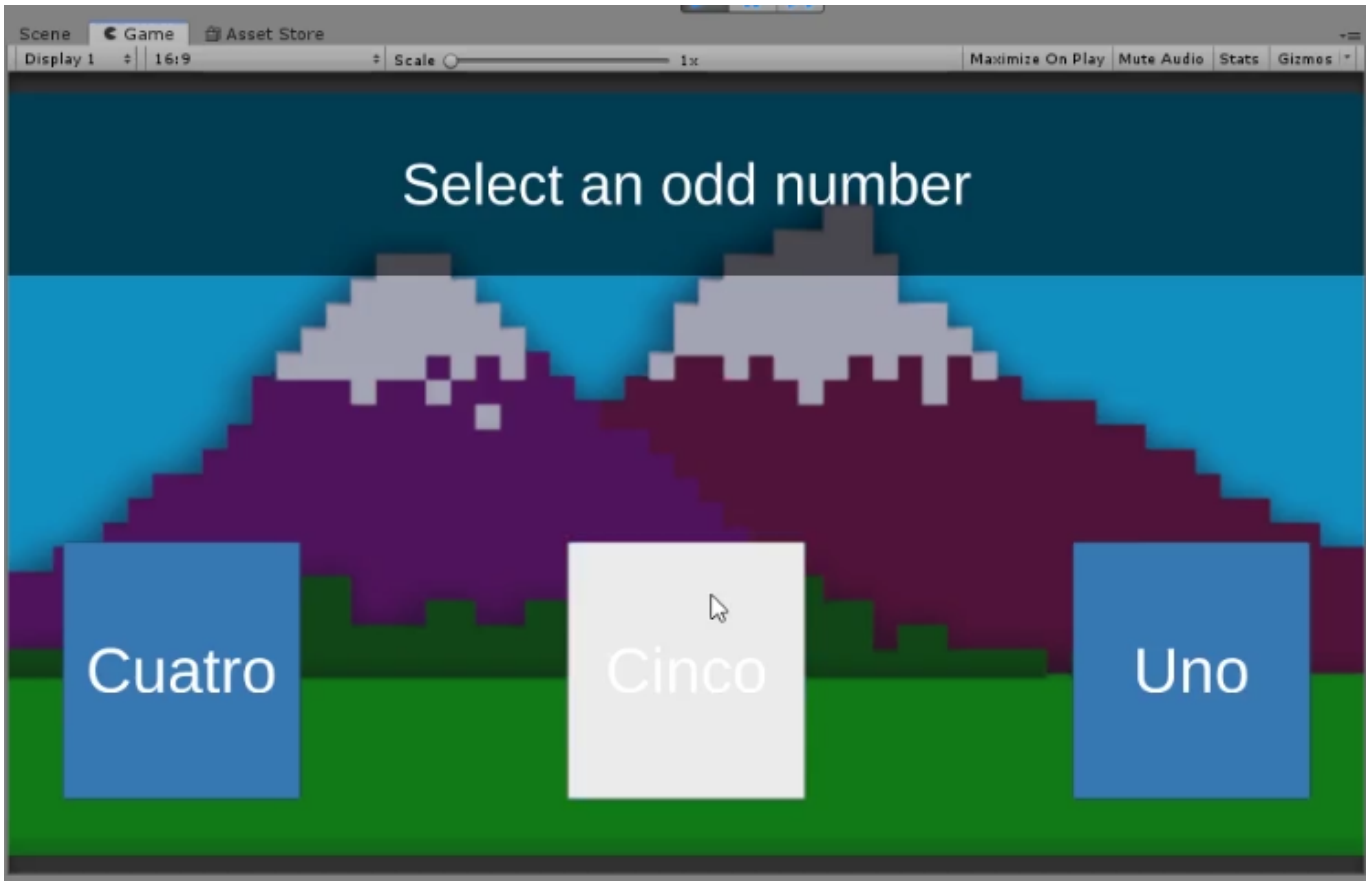
Returning to **Unity**, we can test our clips out. However, before doing so, we need to be sure to head into the **Questions set database** and add **correct answer keys** to our **database**. Remember that you can set multiple option answers now, so you can add whatever variations you'd like.



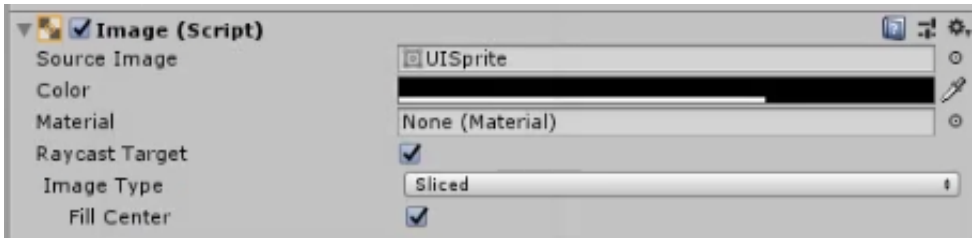
Once you do that, however, you can play around with your quiz and listen to the sounds played depending on whether the answer is right or wrong.

Finishing Touches

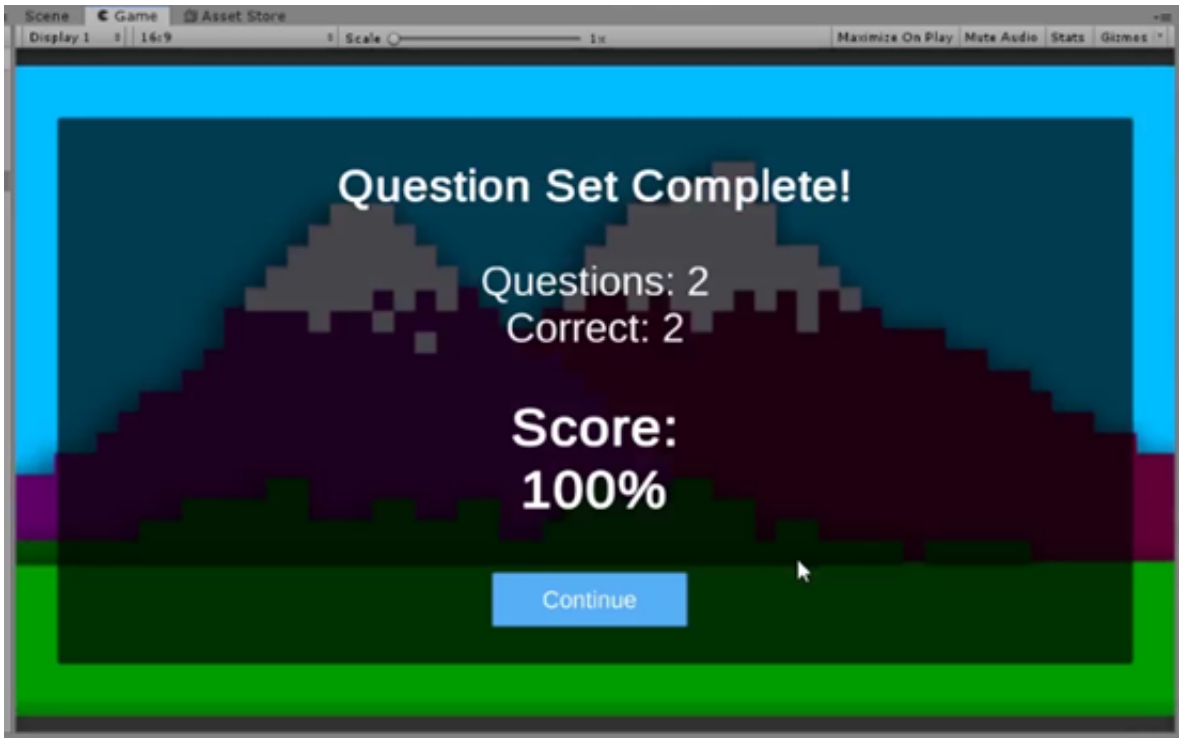
Let's finish our game up by adding some finishing touches. First, we're going to add more questions to our setup. You can make the questions whatever you see fit based on the **assets** you have available. Keep in mind as you fill out the **answer choices**, you are not obligated to fill in the **answer key field** for incorrect answers. If left blank, they will still evaluate to false. Once you add your questions, feel free to test the game from the **Main Menu** to verify everything is working.



The last thing we want to do before ending this lesson is to add a **background** to our **score panel**. By selecting **Score** in the **Hierarchy**, we can **Add** an **Image Component** to it in the **Inspector**. After which, we can add a **UI Sprite** to it and **change** the **color** to a **semi transparent black**. Make sure to resize the **panel** as well to fit better on the screen.



Now we'll be able to see our score better!



Congratulations! You've completed **Zenva's Complete a Spanish-Teaching Quiz** course! Now that we've expanded on the abilities of our **Super Quiz game**, we hope you're able to take it further and make your own fun, educational quiz game on different subjects!