Elements

Gesture Based URI Project

Tomas Brazas G00349242

Link: https://github.com/tomasbrazas97/Elements-Kinect

Video: https://youtu.be/K02i6exWX2Q

This repository contains a project completed for GMIT's Gesture Based URI Development. The project stated that we must *Develop and application with a natural user interface*. The contents contain the source code and documentation for my voice-controlled application "Elements". This document aims to give the reader an insight into the design and development process under the following categories.

- **Purpose of the application**
- **Gestures used**
- **Hardware/Software**
- **Architecture of the solution**
- **Conclusions and Recommendations**

Elements is a voice recognition game that can be fully controlled using your voice. A mouse can also be used as an alternative.

Purpose of the application:

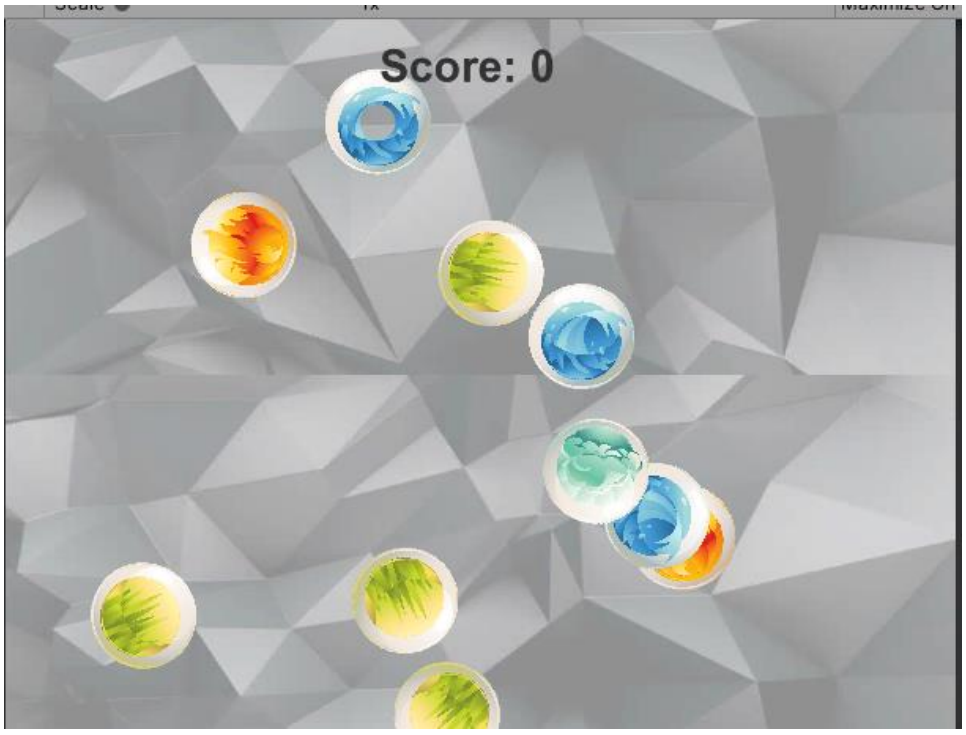
The intention of the application was to design and develop a game for users who cannot use the classic mouse and keyboard to play. That is the target audience for this application. The application uses

Windows Speech Recognition, chose to use this as it would not require any additional expensive hardware setup.

The application is inspired by the four main core elements: Earth, Water, Fire and Air. Aimed for the game to be accessible, easy to understand and easy to use. The game is aimed to be not over complicated, envisioning sitting a relaxing environment while playing this game with the minimal setup that is required.

How to play:

The player controls an orb, distinguished with a transparent mesh in the middle of the sprite to destroy orbs of the same element on the play area. The player element orb can be controlled by voice



recognition or the traditional mouse and keyboard. Upon successful collision the score counter increases, if the orbs are not the same and collision happens then the user is presented with a game over.

Picture of the play area with player orb and randomly spawning orbs.

Gestures used:

The game has been configured to be fully functional with just user's voice. Implementation of compatibility with a mouse is also present so that game is playable at any circumstance.

The voice commands are as follows:

Main Menu:

"Start/Play/Begin" – Begins the game scene from the main menu.
"Quit" – Closes the application.

Game Scene:

"Turn fire" – turns the player orb into Fire element.
"Turn water" – turns the player orb into Water element.
"Turn earth" – turns the player orb into Earth element.
"Turn air" – turns the player orb into Air element.

"Up/ Go up" – moves the player orb up the y axis.
"Down/ Go down" – moves the player orb down the y axis.
"Left/ Go Left" – moves the player orb left on the x axis.
"Right/ Go Right" – moves the player orb right on the x axis.

```
actions.Add("turn fire", Fire);
actions.Add("turn earth", Earth);
actions.Add("turn water", Water);
actions.Add("turn air", Air);
actions.Add("pause", Pause);
actions.Add("continue", Continue);
actions.Add("restart", Restart);
actions.Add("go up", Up);
actions.Add("go down", Down);
actions.Add("go Left", Left);
actions.Add("go Right", Right);
actions.Add("up", Up);
actions.Add("down", Down);
actions.Add("Left", Left);
actions.Add("Right", Right);
actions.Add("Quit", Quit);
```

“Pause” – Pauses the game.
“Continue” – After pause player can continue.
“Restart” – When game over menu is displayed player can restart the game.
“Quit” – Closes the application.

Hardware/Software:

Only necessary hardware for the application is mouse or a microphone. This was done to make the game accessible as originally intended. The two components are standard piece of hardware for a computer. The microphone is needed for speech recognition part of the game.

The software I used to complete this project was software that was available to me for numerous tasks.

- Unity 2019 – Game Engine to build the game.
- Visual Studio 2017 – My personal choice of IDE. Another common IDE for Unity is MonoDevelop.
- Adobe Photoshop – Used to tweak and design sprites.
- GitHub – Source control.

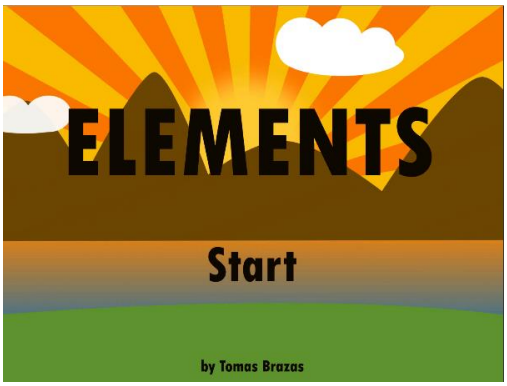
Architecture of the solution:

Libraries used in the project

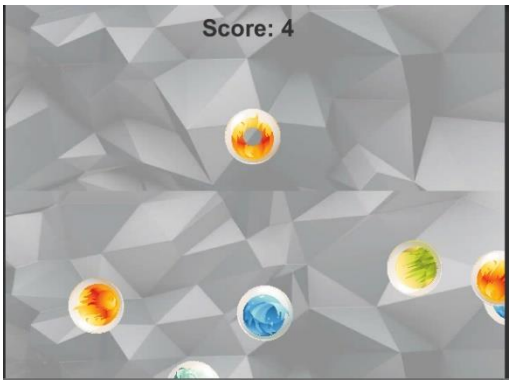
```
using UnityEngine.Windows.Speech;
using System.Collections;
using System.Collections.Generic;
using System.IO;
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.SceneManagement;
using System.Linq;
```

Game Screens

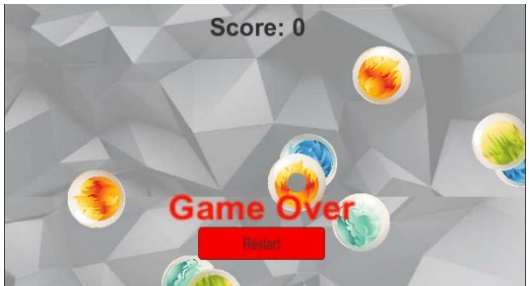
Main Menu



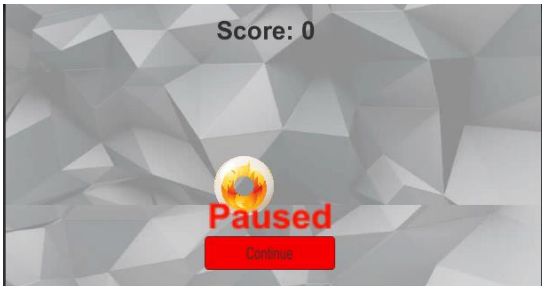
Game Scene



Game Over



Pause



Sound System in the game

The sound system has been developed to avoid silence at all times. Two scripts, first script is DontDestroyAudio which is loaded at the start of the Main Menu. This is attached to an empty gameobject with AudioSource component and handles the background music.

```
public class DontDestroyAudio : MonoBehaviour
{
   void Awake()
    {
        DontDestroyOnLoad(transform.gameObject);
    }
}
```

The second script handles sounds such as GameOverSound, ClickSound, ExplosionSound upon correct matching. This script is attached to an empty gameObject, the sounds are loaded from the Resources folder and only one occurrence of the sound is played when triggered with: `SoundManagerScript.PlaySound("Explosion");`

```
public class SoundManagerScript : MonoBehaviour
{
    public static AudioClip deathSound, clickSound, explosionSound;
    static AudioSource audioSrc;

    // Start is called before the first frame update
    void Start()
    {
        deathSound = Resources.Load<AudioClip>("DeathSound");
        clickSound = Resources.Load<AudioClip>("ClickSound");
        explosionSound = Resources.Load<AudioClip>("Explosion");

        audioSrc = GetComponent<AudioSource>();
    }

    public static void PlaySound(string clip)
    {
        switch (clip)
        {
            case "DeathSound":
                audioSrc.PlayOneShot(deathSound);
                break;
            case "ClickSound":
                audioSrc.PlayOneShot(clickSound);
                break;
            case "Explosion":
                audioSrc.PlayOneShot(explosionSound);
                break;
        }
    }
}
```

Voice Recognition Aspect

The VoiceMenu script takes care of all Voice Recognition in the main menu scene. The VoiceHand script takes care of all Voice Recognition in the game scene. The KeywordRecognizer recognises the inputs from the microphone if there is an action added to the phrase. Speech was implemented in the game wherever possible.

```
keywordRecognizer = new KeywordRecognizer(actions.Keys.ToArray());
keywordRecognizer.OnPhraseRecognized += RecognizedSpeech;
keywordRecognizer.Start();
```

RecognizedSpeech method is used to recognise spoken phrase and invoke an action corresponding to the phrase. Fire() is an example of a method that changes the player orb element with “turn fire”.

```
private void RecognizedSpeech(PhraseRecognizedEventArgs speech)
{
	Debug.Log("Keyword: "+ speech.text);
	actions[speech.text].Invoke();
}

private void Fire()
{
	spriteRenderer.sprite = spriteArray[0];
	transform.gameObject.tag = "Fire";

}
```

Game Play

The aim of the game is to rack up as much score points as possible. The player orb is controlled by the Hand script. If no microphone is present the DragObject script takes care of the mouse allowing the player orb to be dragged around.

```
else if (collision.gameObject.tag == "Air" && gameObject.tag != "Water"
&& gameObject.tag != "Fire" && gameObject.tag != "Earth")
{
	Orb orb = collision.gameObject.GetComponent<Orb>();
	StartCoroutine(orb.Pop());
	return;
}
else
{
	ScoreScript.scoreValue = 0;
	SoundManagerScript.PlaySound("DeathSound");
	m_Rigidbody.constraints = RigidbodyConstraints2D.FreezePosition;
	menuContainer.SetActive(true);
}
```

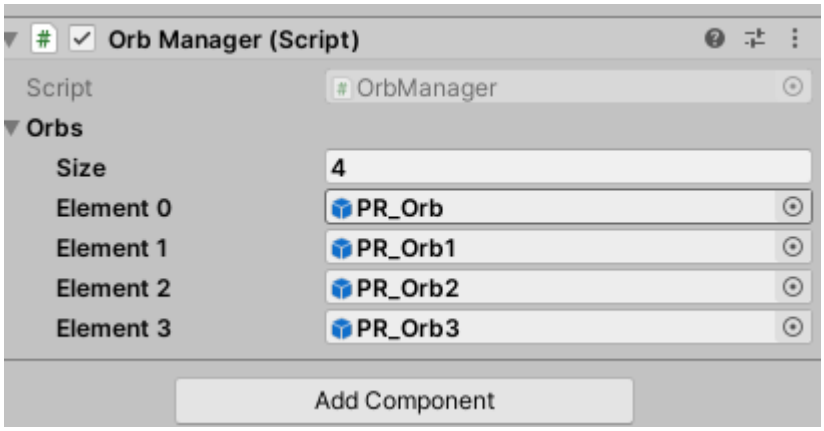
The code snippet above shows how the tags are compared when a collision occurs and if the tags are not the same of the random orb and the player orb then *else* part of the *if* statement kicks in and results in the game over menu.

The random orbs are spawned in by an OrbManager gameobject which has the OrbManager script.

```
public Vector3 GetPlanePosition()
    {
        // Random Position
        float targetX = Random.Range(mBottomLeft.x, mTopRight.x);
        float targetY = Random.Range(mBottomLeft.y, mTopRight.y);
```

```
        return new Vector3(targetX, targetY, 0);

    }

    private IEnumerator CreateOrbs()
    {
        while (mAllOrbs.Count < 10)
        {
            // Create and add
            // ChangeSprite();
            GameObject newOrbObject = Instantiate(orbs[UnityEngine.Random.Range(0,
4)], GetPlanePosition(), Quaternion.identity, transform);
            Orb newOrb = newOrbObject.GetComponent<Orb>();

            // Setup orb
            newOrb.mOrbManager = this;
            mAllOrbs.Add(newOrb);

            yield return new WaitForSeconds(0.5f);
        }


    }
```

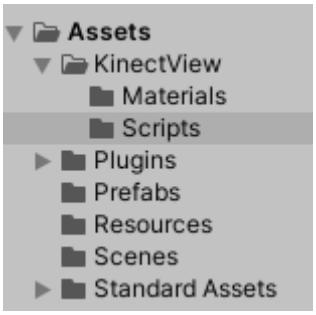
The GetPlanePosition method is used to create an area where the orbs are allowed to spawn in. The numerator method CreateOrbs is in charge of instantiating orb prefabs that are attached to the OrbManager in a random range 0-4.



The Orb script takes care of the rotation of the orbs and the destroy of the gameobjects when successful collision occurs. The rotation and movement of the orbs is randomized.

```
public IEnumerator Pop()
    {
        SoundManagerScript.PlaySound("Explosion");
        mSpriteRenderer.sprite = mPopSprite;
        StopCoroutine(mCurrentChanger);
        mMovementDirection = Vector3.zero;
        ScoreScript.scoreValue += 1;

        yield return new WaitForSeconds(0.5f);

        transform.position = mOrbManager.GetPlanePosition();

        mSpriteRenderer.sprite = mOrbSprite;
        mCurrentChanger = StartCoroutine(DirectionChanger());
```

```
    }
    private IEnumerator DirectionChanger()
    {
        while (gameObject.activeSelf)
        {
            mMovementDirection = new Vector2(Random.Range(-100, 100) * 0.01f,
Random.Range(0, 100) * 0.01f);

            yield return new WaitForSeconds(5.0f);
        }
    }
```



Layout of the folder system.

Test Cases

Main Menu

Test	Expected	Result	Conclusion
Saying “Start” should launch the game scene.	Game Scene is loaded.	Game scene is loaded.	**Pass**
Clicking “Start” button with your mouse should launch the game scene.	Game Scene is loaded.	Game scene is loaded.	**Pass**

Game Scene

Test	Expected	Result	Conclusion
Saying “Turn air/water/earth/fire” to change player orb.	Player orb sprite is changed.	Player orb sprite is changed.	**Pass**
Saying “Pause” to prompt pause menu.	Pause menu container is made visible.	Pause menu container is made visible.	**Pass**
Saying “Continue” to return to the game when Pause menu.	Returns to the game.	Returns to the game.	**Pass**
Saying “Restart” to restart the game when game over menu.	Restarts the game scene.	Restarts the game scene.	**Pass**
Collisions occur correctly, same orbs don’t result in game over.	Score is tracked properly, orbs are compared correctly.	Score is tracked properly, orbs are compared correctly.	**Pass**

Dragging player orb with mouse.	Successful movement.	Successful movement.	**Pass**

Research
What is Speech Recognition?

Speech recognition is the ability of a machine or program to identify words and phrases in spoken language and convert them to a machine-readable format. Rudimentary speech recognition software has a limited vocabulary of words and phrases, and it may only identify these if they are spoken very clearly. More sophisticated software has the ability to accept natural speech.

I focused the project on Windows Speech Recognition, which was initially designed for Windows Vista to control the desktop interface, keyboard shortcuts, send emails and other functions. WSR is a locally processed speech recognition platform; it does not rely on cloud computing for accuracy, dictation, or recognition, but adapts based on contexts, grammars, speech samples, training sessions, and vocabularies.

After thorough experimentation from the hardware that was available in class such as Myo Armbands and Kinects, I chose to use Voice Recognition because it's accessible and easy to use, also allows people to play who cannot traditionally use mouse and keyboard. This was one of my main aims and target audience. It also does not exclude people who have difficulty moving.

Conclusion:

In terms of research, the vast number of technologies that are available to be used for user input was interesting to discover. The choice of games to develop are quite limited but implementations are still apparent. The voice recognition market for games is small and this is one reason why I primarily chose voice recognition to be the main aspect of the game.

I originally also chose to implement Kinect to control the player orb but had to scrap that do to no access to a Kinect for testing purposes when the colleges closed. The code is still available in the project and it should still work as it did in early days if a Kinect V2 is successfully connected. I tried with an Xbox 360 Kinect but due to adapters all being made for V2. I thought they were universal and the Kinect did not connect.

Looking back to the project, I have learnt a lot about Windows Speech Recognition for UWP systems and Kinect development. I also got more hands on the Unity Engine. Game development is something I am passionate about and hopefully hope to progress in it as the years go on. My C# programming skills have also increased.

Recommendations:
If I was to do this project again I would implement a more navigable main menu. Include the Kinect controllability of the player orb.

References:
https://en.wikipedia.org/wiki/Windows_Speech_Recognition
https://docs.unity3d.com/Manual/index.html
https://en.wikipedia.org/wiki/Speech_recognition