Fruit Ninja remake featuring Kinect Controls and Grammar Recognition

Neil Byrne – G00343624

Colin Roche – G00349215

GitHub Repository - https://github.com/colinroche/GestureBasedUIProject

Purpose of the Application

Gem Ninja is a Kinect and grammar recognition game reproduction of the Fruit Ninja game series [1] running on the Unity engine. While there are two Fruit Ninja Kinect game available, both are merely a carbon copy of the mobile version with the same 2D graphics and static camera angle. Gem Ninja is more akin to Fruit Ninja VR which features 3D environments and a dynamic camera that follows the user's head movements. Grammar recognition enables menu navigation and hot-swapping weapons in-game.

Gestures identified as appropriate for this application

Navigation through menus can be controlled with keyboard and mouse, voice controls or motions controls separately or in conjunction with each other.

Kinect Motion Controls

In most Kinect games, navigating through menus is done with a point gesture. A cursor would track your hand movements and select the menu optioned pointed at [2]. During development, we began with this approach. However, moving from a pointing gesture to slashing gestures in game felt disjointed. Instead the Kinect game model avatar is used to interact with Menu objects in the game. This gives the user some time to feel adjusted to the motion controls before they start the game. After experimenting with the different interaction options found during our research. Making the game world "virtual space" was the best fit. This means that content is always attached to the user no matter where they go or what scene they're in [3].

Main Menu Motion Controls

The user is always presented with at least two Menu Gems in front of them with text hovering above them. Hitting a Menu Gem three times will perform the desired action. Although the user may not know three hits is required to start the action, when a Menu Gem is first hit it will spin and play an audio clip. This feedback encourages the user to hit it again which will spin the Menu Gem faster again and the third time will perform the action.

In Game Motion Controls

After learning how Main Menu gems react, the user has a feel for how the hitting game gems works. With a field of view set tor 90, the user can move their head around to see what the environment around them but not too far so that they can't see their screen. A 3D model of the Kinect avatar in the scene is important so the user is not disillusioned by the positioning of the arms.

Grammar Recognition

The Speech Recognition Grammar Specification is used to invoke different methods. Below is the table for the voice commands in the main menu and in game.

Main Menu Grammar Recognition

RuleRef	Action	Method
Play	Play	Sets the Main Menu Panel off Sets the Difficulty Menu Panel on
	GameUI	Toggles the UI Panel
	Volume	Sets the Main Menu Panel off Sets the Volume Panel on
	Back	Sets the Main Menu Panel on Sets the Volume Panel off Sets the Difficulty Panel off
	Quit	Calls Application.Quit()
Game	Arcade	Calls the SceneManager to load Arcade scene
	Classic	Calls the SceneManager to load Classic scene
	Zen	Calls the SceneManager to load Zen scene

In Game Grammar Recognition

Weapon	Hammer	Sets the named weapon in user's hand to Active Sets the named weapon for weapon wall to false
	Axe	" "
	Club	" "
	Dagger	" "
	Scythe	" "
	Spear	" "
	Sword	" "
Pause	Pause	Sets PauseMenu to Active true
	Start	Runs StartGame()
	Resume	Sets PauseMenu to Active false
	Menu	Loads MainMenu scene
	Quit	Calls Application.Quit()
Volume	Off	Sets volume slider to 0
	TwentyFive	Sets volume slider to 25
	Fifty	Sets volume slider to 50
	SeventyFive	Sets volume slider to 75
	OneHundred	Sets volume slider to 100

Game Hints

A non-intrusive method that teaches the user how to use an application is hint boxes. Information about gem values, voice commands and game hints are all conveyed with the boxes every one minute.

Hint examples :"Say The Name of A Weapon To Swap It Out", "In The Main Menu Say 'Arcade' to Play Arcade mode"

Hardware used in creating the application

Kinect V1

The Kinect V1 was released November 2010, featuring a 3D Depth sensor (IR Emitter + IR Camera / Depth Sensor), RGB Camera, Microphones and Tilt motor (for detecting floor and players in playspace). While the technology in modern VR systems is improved in every way compared to Kinect, buying a VR system solely for this project was nonsensical. Instead we bought a Kinect for €30 and challenged ourselves to make a Kinect game to the quality of a VR game. Microsoft have an SDK and window? available for developers with packages to be implemented with the Unity [4].

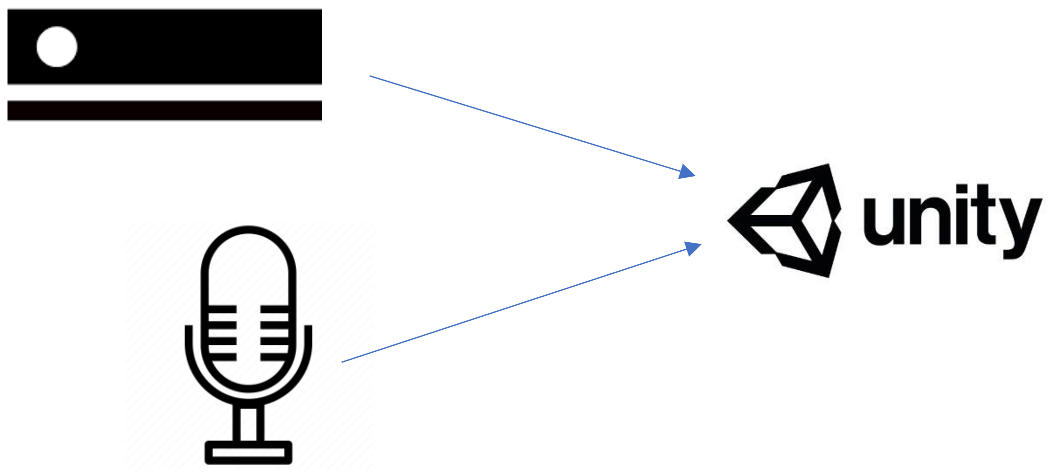
The most recent version of the Microsoft SDK is version 2.0, we worked with this for quite some time, but we were not able get it fully functional. After some further reach we realised this version is supposed to be used on the Kinect 2.0 which is the Kinect for the XBOX One not the XBOX 360, which is the Kinect we are using. We were then able to download the Microsoft SDK version 1.8 which was able to be used with this version of the Kinect.

Using the version of the SDK we were able to use the Kinect with MS-SDK on the Unity store [5]. There was virtually no set up required after this apart from some errors which required some modification of the provided scripts. This was mostly fixing the affected scripts with updated unity code. The package also contained a series of scenes demonstrating what the Kinect is capable of. Playing around and manipulating these scenes is what inspired the kind of game we were to develop and let us brainstorm how we would achieve it.

Microphone

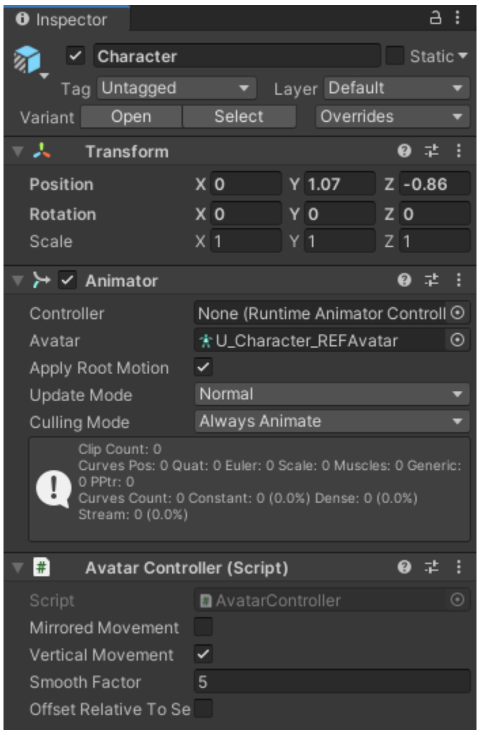
The computer microphone is used to detected voice commands to allow the user to control the game with the various voice commands described in the grammar recognition tables.

Architecture for the solution

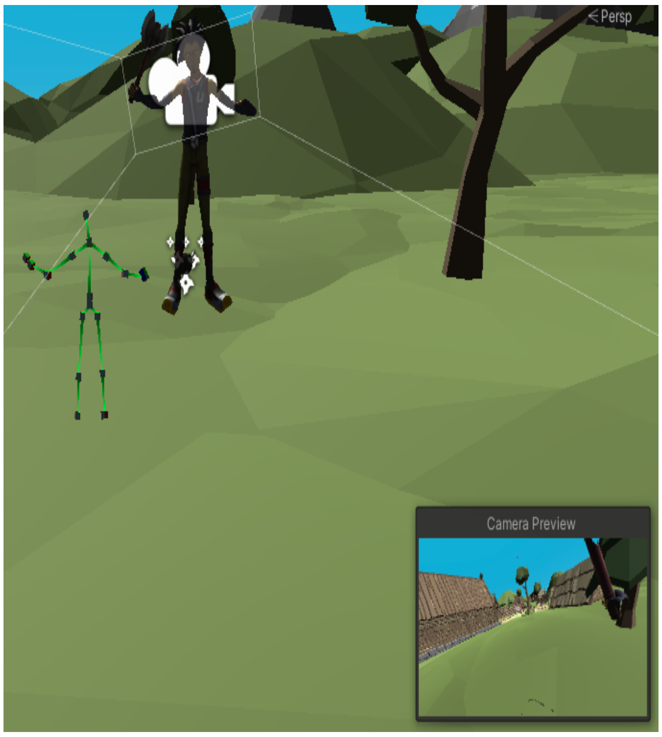


Motion Controls

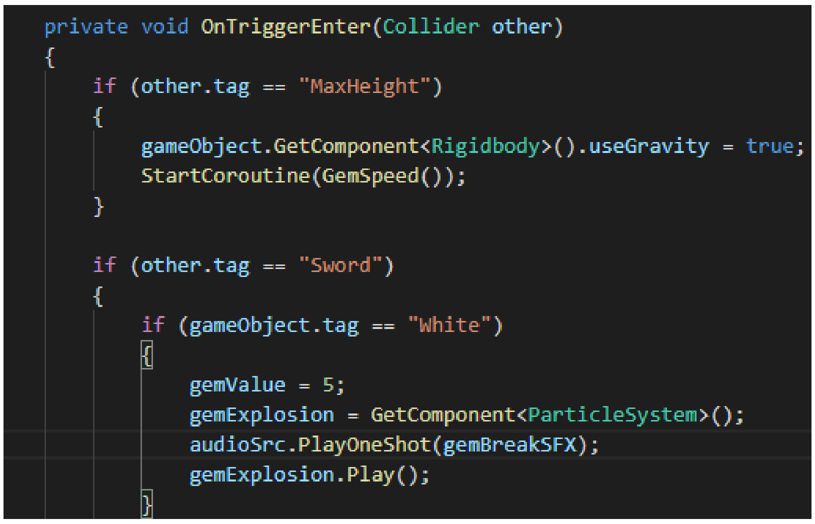
The Kinect SDK provided the code for the Kinect to map the user from the camera array to a 3D model avatar.



```
Inspector
Character                         Static
Tag Untagged          Layer Default
Variant   Open   Select   Overrides
Transform
Position     X 0     Y 1.07    Z -0.86
Rotation     X 0     Y 0       Z 0
Scale        X 1     Y 1       Z 1
Animator
Controller          None (Runtime Animator Controll
Avatar              U_Character_REFAvatar
Apply Root Motion   ✓
Update Mode         Normal
Culling Mode        Always Animate
    Clip Count: 0
    Curves Pos: 0 Quat: 0 Euler: 0 Scale: 0 Muscles: 0 Generic:
    0 PPtr: 0
    Curves Count: 0 Constant: 0 (0.0%) Dense: 0 (0.0%)
    Stream: 0 (0.0%)
Avatar Controller (Script)
Script                    AvatarController
Mirrored Movement         ☐
Vertical Movement         ✓
Smooth Factor             5
Offset Relative To Se     ☐
```



Scripts can now be applied to the model avatar. However, the only script added is the game hint script. Since all we care about is the position of the user, a box collider added to the model will give us the necessary feedback. The weapons attached to the model hand all have the “sword” tag. Each gem will check this collision and only react to the weapon.



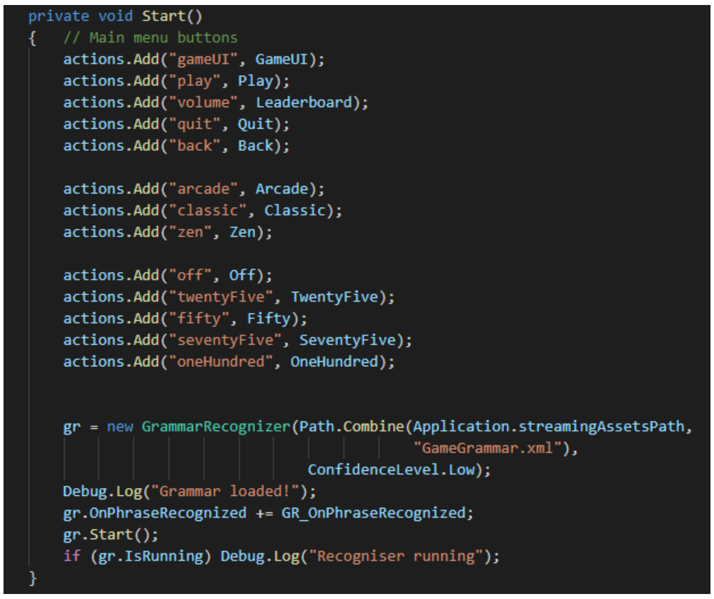
```
private void OnTriggerEnter(Collider other)
{
    if (other.tag == "MaxHeight")
    {
        gameObject.GetComponent<Rigidbody>().useGravity = true;
        StartCoroutine(GemSpeed());
    }

    if (other.tag == "Sword")
    {
        if (gameObject.tag == "White")
        {
            gemValue = 5;
            gemExplosion = GetComponent<ParticleSystem>();
            audioSrc.PlayOneShot(gemBreakSFX);
            gemExplosion.Play();
        }
```

Grammar Recognition

The Speech Recognition Grammar Specification (SRGS) is used to incorporate an xml syntax file of a set of word patterns to expected to be used.

Grammar Recognition type is created with a low confidence level and pointed at the xml grammar file.



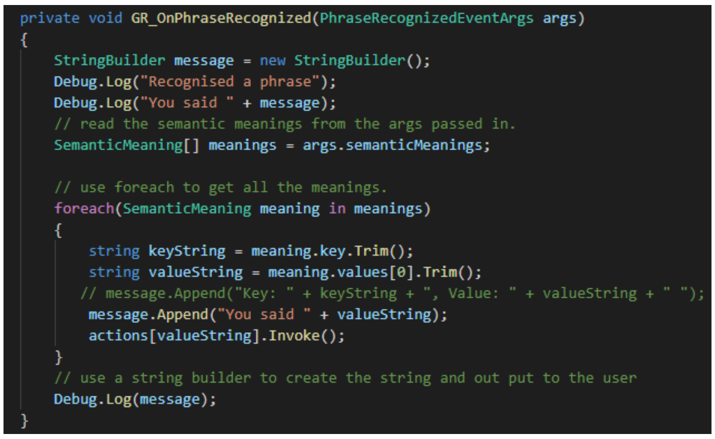
```
private void Start()
{   // Main menu buttons
    actions.Add("gameUI", GameUI);
    actions.Add("play", Play);
    actions.Add("volume", Leaderboard);
    actions.Add("quit", Quit);
    actions.Add("back", Back);

    actions.Add("arcade", Arcade);
    actions.Add("classic", Classic);
    actions.Add("zen", Zen);

    actions.Add("off", Off);
    actions.Add("twentyFive", TwentyFive);
    actions.Add("fifty", Fifty);
    actions.Add("seventyFive", SeventyFive);
    actions.Add("oneHundred", OneHundred);


    gr = new GrammarRecognizer(Path.Combine(Application.streamingAssetsPath,
                                    "GameGrammar.xml"),
                            ConfidenceLevel.Low);
    Debug.Log("Grammar loaded!");
    gr.OnPhraseRecognized += GR_OnPhraseRecognized;
    gr.Start();
    if (gr.IsRunning) Debug.Log("Recogniser running");
}
```

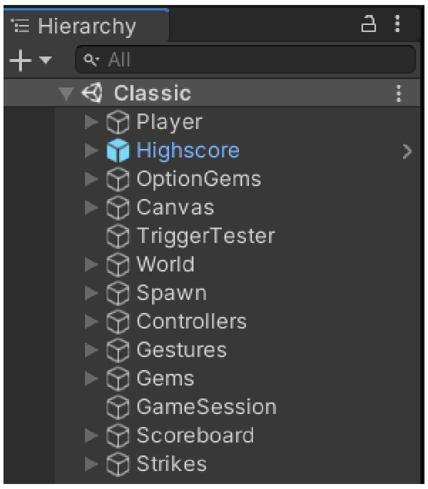
Once a phrase from the xml grammar file is detected the valueString is passed as an invocation to it's method counterpart. The tables above describe what action is performed by each phrase.



```
private void GR_OnPhraseRecognized(PhraseRecognizedEventArgs args)
{
    StringBuilder message = new StringBuilder();
    Debug.Log("Recognised a phrase");
    Debug.Log("You said " + message);
    // read the semantic meanings from the args passed in.
    SemanticMeaning[] meanings = args.semanticMeanings;

    // use foreach to get all the meanings.
    foreach(SemanticMeaning meaning in meanings)
    {
        string keyString = meaning.key.Trim();
        string valueString = meaning.values[0].Trim();
       // message.Append("Key: " + keyString + ", Value: " + valueString + " ");
        message.Append("You said " + valueString);
        actions[valueString].Invoke();
    }
    // use a string builder to create the string and out put to the user
    Debug.Log(message);
}
```

Unity Hierarchy



Classic
- Player
- Highscore
- OptionGems
- Canvas
- TriggerTester
- World
- Spawn
- Controllers
- Gestures
- Gems
- GameSession
- Scoreboard
- Strikes

Game Object	Script	Description
Player	Game Hint Changer	An array of texts animations and controllers to fade hint in and out are configured. An InvokeRepeating will call Tick() every 60 seconds and set the next hint in the list to active.
	Avatar Controller	Script provided by Kinect with MS-SDK package.
OptionsGems	Main Menu Gem behaviour	Tags are used to distinguish which gem has been hit and run the associated animation, audio clip and particle effect. Menu Gems each have panels which should be loaded in and out.
Canvas	Main Menu	Controls UI panels and scene management.
Spawn	Spawn Manager	Calls the GameSpawn Script.
	Game Spawn	Check the current scene that is loaded and calls the appropriate method in the SpawnLevel Script.
	Gem Spawner	Gets the transform for the gems/bombs/clock, clones them and positions the cloned object in a random position at appropriate distance from the player.
	Spawn Level	Spawns the relevant gems/bombs/clocks depending on current level. Set the time for delay on the loading of the gems. Calls the appropriate type of spawning for the level.
Controllers	Sound Manager	Plays configured menu audio clips onClick().
	Main Menu Speech Recognition	Initialisation for grammar recognition and functions for all voice commands.
Gems	Gem Behaviour	Using tags, checks which gem/bomb/clock has been hit by the player and give the relevant value to the object. Checks the max height for the gem and applies gravity while also stopping speed of the object so they fall. Calls the methods for the functionality of the bombs/clocks.
Game Session	Game Session	On Start, checks the level, sets up the relevant High Score object, sets the timer for Timeboard and the score for both the Scoreboard and HighScoreboard.
	AddScore()	Has the method for adding relevant score for the gem.
	BombCheck()	Checks how many bombs have been hit and enables the relevant strike. When three bombs have hit calls the EndGame().
	LevelCheck()	Checks the current Scene and sets the currentLevel to the current Scene.
	EndGame()	Checks the current level, if the player's score is greater than the high score sets score as new high score. Reloads the current scene.
	Countdown()	If the level contains a Timeboard, reduces the Timboard value by time. When the time is less

		then ten sets its colour to red. When it reaches zero calls the Endgame().
	StartTimer()	Sets the current time to the pre-set start time.
	AddTenSecs()	Adds ten seconds the value of the Timeboard. Called when the clock object is destroyed.

Relevant Libraries Used

UnityEngine.Windows.Speech – Dictation Recogniser listens to speech input and attempts to determine what phrase was uttered.

Microsoft Kinect SDK – Allows Unity to interface with the Kinect.

Conclusions & Recommendations

When given the project specification, the opportunity to develop a hand gesture-based application was realised. The Nintendo Wii remotes were first considered as we both believed it is underappreciated hardware and an underdog when it comes to developing a gesture-based application. Head and finger tracking is possible with remotes although it has not been supported in some time [6]. And a $25 price of the Wii Remote API made us reconsider our hardware [7].

The Kinect is a cheap and easy to use hardware to develop with. The big, expensive VR giants have left hardware like the Wii remotes and Kinect forgotten. For this project, we developed a remake of Fruit Ninja VR with all the same game modes that works using a Kinect. While the motion tracking is less accurate, and the absence of game controllers will lead to tricky workarounds in other remakes. We have learned that these old technologies are still powerful and should be investigated further as a low-cost alternative for gesture-based applications.

SRGS is an excellent for grammar recognition, linking voice commands to the game logic can be challenging. Before jumping into writing code for a feature, some consideration should be taken for how voice commands will be implemented. Due to the Kinect not having any hand controllers, a way to swap weapons (without UI controls) easily wasn't possible without voice commands. This gave the voice commands a practical reason to be featured.

In conclusion, we both enjoyed working gesture-based hardware and are more interested to work with legacy hardware in the future.

References

[1] – Fruit Ninja: https://www.halfbrick.com/games/fruit-ninja

[2] – Kinect Menu Gestures: https://www.gamepur.com/guides/how-to-navigate-xbox-one-dashboard-using-kinect-gestures

[3] – Designing Game Menus: https://medium.com/predict/the-4-most-important-observations-to-remember-when-designing-vr-menus-bf6ea35e5574

[4] – Kinect SDK: https://www.microsoft.com/en-us/download/details.aspx?id=44561

[5] - Kinect with MS-SDK: https://assetstore.unity.com/packages/tools/kinect-with-ms-sdk-7747

[6] – Head Tracking: http://johnnylee.net/projects/wii/

[7] – WiiBuddy: https://assetstore.unity.com/packages/tools/input-management/wiibuddy-4929#content