COMP437/537 Final Project Submission Form

Project Description:

This project introduces a new interface for level design in Unity, by integrating voice commands, hand gestures and gamepad controls to work together. This combination was designed to alleviate the complexity of Unity's interface, by proposing an object focused method instead of using menus on all sides of the screen.

This approach could address the issues of losing focus while designing a level and failing to meet the initial vision of the designer by allowing to design from the observer's angle and making the UI require less distracting.

Also, it was designed to be applicable with VR level design which is not present for Unity at the moment of this project's development.

Technical Aspects:

User can give voice command, and natural language processing is used to retrieve the user's intent and relative entities. Microsoft's Cognitive Services were used to get text from the speech, and LUIS was used to get intents and entities from the text. Speech to text is based on Active Nick's Unity-MS-SpeechSDK. Since LUIS tutorials were all deprecated, code based on Creagines' UnityWebRequest tutorial was used to retrieve JSON from the query page. Serialized classes were defined to easily transfer data from JSON by using the built-in JsonUtility functions.

For natural language processing, a model had to be trained. Create Object and Edit Object were designed as the main intents, then Create Walls, Create Floor and Either Side was also added as they differed from the intents defined initially.

Create Object uses VerbCreate (place or create) and GameObject (furniture names) and depending on the use case, it can also have an int (numeric amount), Location Pointing (there or here), or a relative location entity. Optional entities determine the use case. If location pointing is present, raycast from right index finger's position and angle to floor is used to determine a location that will be used for the object. Relative location is a composite entity that takes relative position (like on top) which is applied to given object in the composite entity. If a numeric amount exits, multiple amounts of the given object is created. If none of these optional entities are present, a single object is created at the default location.

Edit Object takes VerbEdit (make or change) and ObjectAttribute (color, size or material) and applies them to an object. If the object is given verbally with its name, the program chooses the one that is closest to the center of the screen. If user calls out the last created object with LastObject (it or that), program looks up the last created object to apply the attributes. If ObjectPointing (this) is present, raycast from right index finger's position and angle to any objects within the LevelObjects layer and has a collider is done to determine the object that is being pointed at. Color

These commands support default size modifiers and color adjusters, and also support expandable list of user defined materials (textures) and prefabs (game objects). Expandable list was implemented by a string to game object dictionary defined using a Rotary Heart's Serialized Dictionary for extra flexibility and for making it visually accessible in the inspector window of Unity.

CreateFloor can take either one number and make a square floor, or take two numbers and make a rectangle floor. CreateWalls takes the height of the walls, and surround the floor that was determined by CreateFloor. Both CreateFloor and CreateWalls have default arguments in case user does not quantify the sizes.

EitherSide takes two object names (“Create chairs(1) on either side of the table(2)”), and searches the second object on the screen and picks the one that is closest to the center. Then it computes both widths of the object from its rendering. It picks the shorter width, and places the other object twice at the center of the first object then offsetting depending on the shorter width of both objects and rotating first objects to face center of the second object.

Using a gamepad inputs, user can navigate the camera, switch views, and control objects. For easy use, camera hovers around by moving it with left analog stick, and rotating it with right analog stick. This provides a quick and a familiar method to move around. For comparison, Unreal Engine 4 does not use a gamepad, instead movement is made by making a rope pulling gesture with HTC Vive.

Rotating left and right occurs often during design, therefore buttons dedicated to turning 90 degrees were added. L1 or Left Trigger is used to turn the camera the axis that is left of the camera. So, if it’s at 115 degrees, it will rotate to reach 180. Also, a tolerance variable was added, so if at 88 degrees, it rotates the came 92 degrees instead of 2 degrees.

Objects within the level need to be moved around too, so a button to select and unselect was added. When A button on Xbox controller, or X on PlayStation gamepad is pressed, the game object with the tag LevelObjects that is closest to the center is picked. After being selected, left analog stick is used to control X and Z axes of the object, and right analog stick is used to rotate the object. Unlike the camera, the movement of the object is relative to the world instead of itself. This makes it simpler to move objects only on X or Z axes.

There is also an auto select option that can be toggled on the screen. This makes it so that every placed object is selected, so their placements can be quickly adjusted.

During object placement, checking from multiple view angle is a common practice in level design. To achieve this quickly special view were introduced. By pressing the Y button on Xbox controller or Triangle on PlayStation gamepad, user can switch to bird's-eye view, which is at a certain height looking directly down. This can also be used in conjunction with a selected object to go bird's-eye view on the object, and keep track of the object from the above while it is being moved around.

There is also a side view option, but that is only available when an object is selected. It makes the camera look directly at the object from its left side.

Bird's-eye view and side view are considered as special views, and user can go back to normal view, with the same location and rotation it was in just before using a special view. This is done by pressing the X button on Xbox controller, or Square on PlayStation gamepad.

Camera movement and rotation, object movement and rotation, turning tolerance are available on the inspector so that user can adjust them easily from the Unity's UI.

By default, Unity discards any changes made in its testing environment called "game mode". But to make use Speech Services and Leap Motion, that mode needs to be used. Thus, running it in that mode but saving every change to a file and loading the changes after returning to the editor solves the problem. For this a code based on the Object Minder from Filmstorm was used, which saves every coordinate of position, rotation and size of every object in the scene to a file. Then when returned to editor mode, it loads and updates these values onto the objects. This code originally did not create objects while loading, so it had to be updated since object creation is a common feature. Separate list for created objects is kept, and as user is exiting game mode, these object names are also saved. Then in the editor, these object names are read and initialized to the scene again. Now their values that was saved can be applied onto them.

I had proposed the idea of creating an object to have the size of the distance between the palms when the voice command similar to “Create a cube this big” was given. This was pushed back due to being just a novel idea, while some features had much more priority to be implemented to be feature-complete.

I had promised an idea for editor to propose possible placements for a given object. As the feature selection and collecting data for each object was too complex, this feature was discarded in favor of implementing simpler but more impactful features for the program.

My initial propose was focused on giving simple commands with speech and Leap Motion. To be more like a proper editor that can be used I introduced gamepad controls to easily move around, and move objects around, also the special views to provide common editor functionality. My propose examples had simple create and edit commands, but I noticed relative placement was a huge time saver that could be introduced to challenge the traditional methods of level design. These additional futures push this project from being just a novel idea to a proper editor tool.

Novelties:

At the moment of this project's development, Unity does not have an interface for level design in VR. Only the main competitor engine of Unity, Unreal Engine 4, has a default VR level design support. All the components of this project work well within the VR environment.

Level design tools mainly work by keeping position, rotation and size relative to the its world. Unity game objects can also have a relative position, rotation and size relative to parent objects when they are placed under another object. But placing an object with relative position without being a child to another object is not possible. This project can natively place objects relative to each other. For further ease, certain relevant placement options, such as on top of another or either side of another, are provided.

Camera in the project can also act relative to object and keep tracking it like from a bird's-eye view. Unity natively can focus on an object and change perspective on it, but it does not keep tracking as object is moved around.

Technologies Used:

Program is an editor designed for Unity; therefore, it needs be run in Unity. For compatibility reasons, .Net 4.x needs to be installed.

Project mainly uses speech as input, which requires internet connection to reach Microsoft Azure's Cognitive Services to get text from speech and Language Understanding Intelligent Service (LUIS) to get intents and entities from the text. Both Cognitive Services and LUIS requires an API key with limited use.

Project has methods that make use of Leap Motion. These are not mandatory functionalities but they increase the intuitiveness and productivity of the product. Leap Motion is also computationally intensive, it may cause performance issues on laptops or old systems.

Project can manipulate the main camera and selected objects with a gamepad controller. Unity input is global, so most of the gamepads could work. But the project's controls were optimized for Xbox 360 and PlayStation gamepads. Without a gamepad, navigation would require interacting with the classic interface of Unity.

This project still does not have all the functionalities of a level design tool. Some of the missing important features include entering specific size and position numeric values, moving objects across y axis, and deleting objects.

Evaluation Conducted:

Evaluation was done by having a target design for a room, and using both this system and the classic methods to re-create this target room. Process of doing so was recorded to be checked to do analysis on time spent. Position and rotation values of the placed objects in the room was also recorded to compare them for accuracies. I was the subject for both tests. I have few hours of experience with this system, and few years with the classic method of Unity, and I made use of both systems shortcuts to fullest within my abilities. The target room was a simple room with 6 objects.

This method completed the room in 164 seconds, while the classic methods took 232 seconds. But this method had a total of 0.473-meter inaccuracies in positioning, whereas classic methods only had 0.172-meter inaccuracy. This method also had 1.036 degrees off while the rotation in classic method had perfect accuracy. (Details are in the excel file)

I also categorized the time spent to get meaningful results. Planning the next step in this system is less than half of the time in classic system despite my experience in the classic system. This may suggest intuitiveness or the simplicity of the interface. On the other hand, waiting for voice commands to be deciphered took three times the time spent navigating the interface of Unity assuming relevant objects were put in a single folder. Yet that time difference was overshadowed by the time of object placement. Pointing to a place and making adjustments with a controller was much faster than finding the right values and entering them (even though it was less accurate as numbers indicate). And since object placement is the major part of level design, it had the most impact on time.

It's important to denote that this was room that could make use of the features already present in the project. Their ease of use saved time, but that is not applicable to every room design.

It was concluded that the system needed a snap to degrees and position command, and an extra interface to see values on the screen rather than the inspector to do more accurate placements easier.

Related Works:

Codes used:

Speech to Text by Active Nick: https://github.com/ActiveNick/Unity-MS-SpeechSDK

Leap Motion SDK and Unity API by Leap Motion: https://www.leapmotion.com/

Object Minder by Kieren Hovasapian: https://filmstorm.net/blogs/news/keep-your-unity-scene-updated-after-exiting-game-mode-object-minder

UnityWebRequest by Creagines: https://www.youtube.com/watch?v=C0V3RhAVHac

Get Closest by bigmisterb: https://forum.unity.com/threads/find-the-nearest-enemy-center-of-the-screen.164689/

Unity's audio to Wav by deadlyfingers: https://github.com/deadlyfingers/UnityWav/blob/master/WavUtility.cs

Serialized Dictionary by Rotary Heart: https://assetstore.unity.com/packages/tools/utilities/serialized-dictionary-110992

3D Assets used:

Big Furniture Pack by Vertex Studios: https://assetstore.unity.com/packages/3d/props/furniture/big-furniture-pack-7717

Dining Set by FunFant: https://assetstore.unity.com/packages/3d/props/interior/dining-set-37029

Wispy Skybox by Mundus Limited: https://assetstore.unity.com/packages/2d/textures-materials/sky/wispy-skybox-21737