Course Project Proposal – VR Cooking Game

Virtual reality application concept.

This virtual reality cooking game gives the user control of the kitchen, in order to fulfill meal orders for customers. The user will be given various customer orders, and it is up to them to make and assemble those orders for delivery. Points are awarded on successful delivery of orders, and the goal is to get as many points as possible within the given time limit.

The main setting for the game is in a virtual kitchen. As orders come in, the user has to find all the needed ingredients. Based on what the order is, the user may need to do something with the ingredient (such as cook meet or cut up lettuce), and then assemble the order on a dish. Once the order is done, the user can pick up the plate and take it over to the delivery window to finish that dish. The user will need to be able to grab specific ingredients, successfully perform operations on them, and assemble them together correctly (see Figure 1).

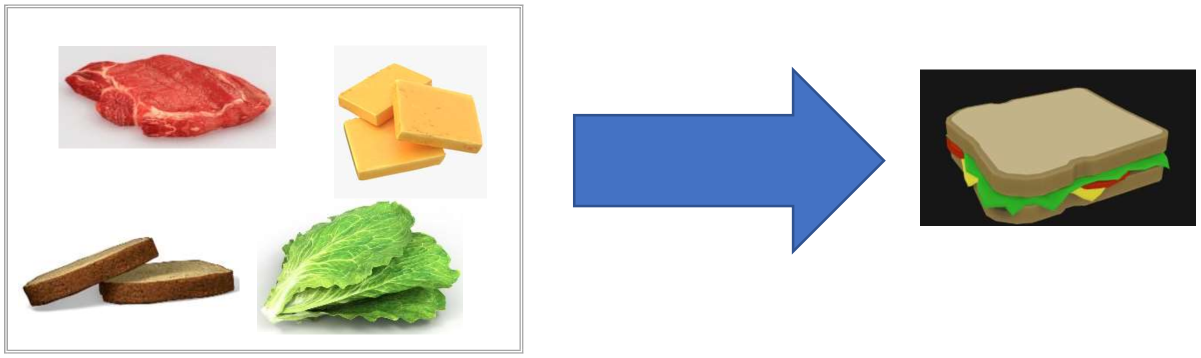

Figure 1: Assembly of ingredients into order, as described in the Virtual Reality Application concept

Main technical features.

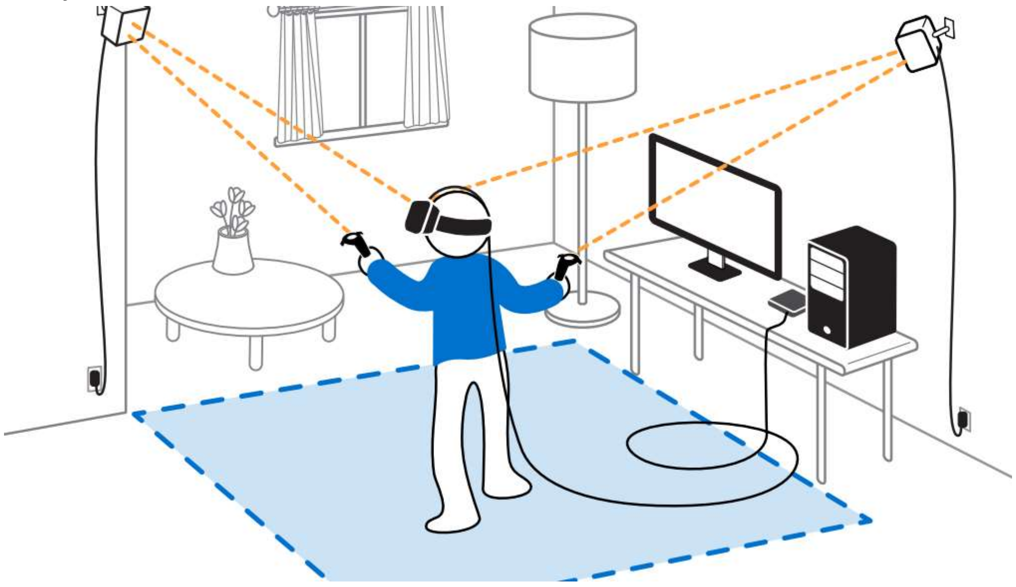

Figure 2. Setup for the cooking game includes the VR system, a headset, two controllers, and at least 2 cameras to monitor user movement. User will also need to clear out a small space in the room in order to play without hitting anything.

In this game, the most important features are the ability to see orders, select ingredients, and put those ingredients together. This requires the user to be able to pick out a specific ingredient from all the items on their screen, and pick it up using the trigger on their controller. The user allows needs to be able to put down the ingredient in the right place, to assemble it, and possibly pick it up and relocate it if they put it down on the wrong plate.

This game will be able to be played without the user needing to walk around everywhere. The ingredients will be located around the user, and the counter where they assemble the dish will be close to them. The user should be able to pick up ingredients, put them together, and even deliver them without needing to move more than a few steps from their standing location. Setup for this game includes a VR system, multiple cameras (at least 2), and a small clear area in which to play without hitting anything, as can be seen in Figure 2.

- **Order generation:** orders come in at designated times, and are displayed somewhere in the user's view [*required]*
- **Ingredient selection**: select the specific ingredient items needed, one at a time, to make the order *[required]*
- **Meal/Order Assembly**: put the required items together to make the order *[required]*
- **Level time limit**: restrict the amount of time given to the player in each level *[required]*
- **Game sounds:** music during play, and sound effects on certain actions to notify the user *[required]*
- **Delivering meal/order to customer:** deliver the order to the customer who ordered it by putting it in a designated delivery area *[optional]*
- **Scoring**: add or subtract from score value based on completed correct orders or incorrect orders *[optional]*
- **Meal/order variety**: orders can be from a variety of different meals with different options & add-ons or subtractions from the basic meal *[optional]*
- **Meal/order time limit**: each order has a time limit that starts when the order is placed *[optional]*
- **Tools and appliances**: using different tools and appliances (such as spatula, oven, frying pan, stove) to make different orders *[optional]*
- **Different levels**: different levels of difficulty and possibly location/appearance to add variety to the game *[optional]*

Course Project Design and Development

Virtual reality system-application design.

This game was designed to work in a virtual reality environment, both in initial testing phases and during the actual virtual reality implementation. To make up for the limited space a user would have in a virtual reality setup, the game was designed to work with minimal steps needed by the user. All items would be close to the user, and allow the user to get the full experience without worrying about room space or tripping over any cord.

Apart from the size of the gameplay area, this game also took a virtual reality based design in its controls. All the controls are very simple, and rely more on the user's movement, and the only button used is the trigger button. This was designed not only for ease, but to allow the user simple controls as if they were in reality picking up ingredients, and also ensure the user does not feel lost at first when they do not know the controls of the game.

Since a virtual reality system does not limit user movement or their ability to look around, this game made up for that by ensuring that the entirety of the gameplay area looked complete. This includes a furnished "restaurant" area outside the kitchen, where the user works, to make the game feel complete and full.

Immersion and presence

This game takes place in a simulated restaurant kitchen, facing a restaurant. The kitchen is small but complete, so that it appears to be an actual room. Since the movements needed by the user are limited, and the setting is a simple, non-moving room, sickness is not a large concern, as the user will know and see that they are in a room, although the room that they see is obviously different than the physical room they are in. The limited movements also allow the user to play the game without worrying about moving past the limit of the VR system, or moving into physical objects in the room. The only concern is the user's rotation, and getting tangled in the cord.

Immersion is increased by making most UI elements in-game. Dish orders come in on the window, like an actual order from a restaurant. The time is displayed on a clock that sits on the main assembly counter and counts down. The score is kept using a tip jar, and coins that appear in the tip jar upon order completion.

Controls were also evaluated to help with immersion. Instead of making use of multiple buttons on the user's controller, which would force the user to know the buttons or be able to see the buttons, only the trigger is used, to make it feel more like the user is actually in a kitchen, and less as though the user is simply playing a game.

Virtual user interface and interactions

Various information needs to be available to the user during each level, primarily the time left in the level, the user's current score, and the orders that are needed to be made. The orders appear in the window at the front of the restaurant, which is the main location the user will be facing. The timer is displayed as an alarm clock on the main assembly table. The score is given by a tip jar, that coins are dropped in when an order is completed successfully.

Sounds are used as part of the user interactions. When a recipe is completed, and the assembled dishes turn into the correct dish. This transformation is also accompanied by a small particle system, which, while it may somewhat affect the immersion, gives it a more natural game feel. After the user delivers a dish, if the dish matches an order, a cash register sound plays, accompanied by coins dropping into the tip jar and another particle system. If the dish does not match any current order, a buzzer sounds and the particle system is red, showing the user that the order was not correct.

When the level is ended, which occurs when the timer runs out, the user is notified by an alarm clock going off. A UI canvas then appears, in real world space so the user can look away, with the score needed to win, the user's score, and then a statement underneath saying if they passed or failed.

Technical implementation

- **Order generation:** orders will come in at random times, given constraints of minimum and maximum times between orders, maximum orders possible at a time, and whether or not there are orders currently available (see **Code 1**). The order will appear to the user as an order ticket, similar to one seen in a restaurant setting, but large enough so that the user can see it. The tickets will appear in the center of the main view so that it will be easy for the user to refer to, and will be generated when a new order is generated (see **Code 2**).

Code 1: Unity C# script that generates new orders for the user. Order generation is based on minimum and maximum time between orders, as well as whether or not there are current orders available for the user.

```
// Generate a random index between 0 and recipeList size
// and choose the next order based on random number
void generateNextOrder()
{
    int recipeIndex = rnd.Next(0, 2);
    nextOrder = recipeList[recipeIndex];
    Type recipeType = Type.GetType(nextOrder);
    Recipe newOrder = (Recipe) Activator.CreateInstance(recipeType);
    orderQueue.Add(newOrder);
    outputOrder(currentOrders);

    currentOrders++;
}
```

Code 2: C# Script that sends new orders to the virtual environment so that they are available to the user.

```
// Sends out order to user
void outputOrder(int index)
{
    Transform newTicket = Instantiate(ticket, firstTicketPosition,
Quaternion.Euler(ticketRotation));
    if (index != 0)
    {
        newTicket.position = new Vector3(ticketQueue[index - 1].position.x -
0.647F,
                ticketQueue[index - 1].position.y, ticketQueue[index -
1].position.z);
    }
    ticketQueue.Insert(index, newTicket);
    Transform canvas = ticketQueue[index].GetChild(0);
    Text orderTitle =
canvas.Find("OrderTitle").gameObject.GetComponent<Text>();
    Text orderContents =
canvas.Find("OrderContents").gameObject.GetComponent<Text>();

    string title = orderQueue[index].getRecipeName();
    string contents = orderQueue[index].ingredientMenu();
    orderTitle.text = title;
    orderContents.text = contents;
}
```

- **Ingredient selection:** ingredients are placed on a counter, the ingredient counter, in a row so the user can see all available ingredients and select what they need. Ingredients automatically spawn and regenerate when picked up so that the level never runs out of ingredients. This respawn works by continuously checking to see if the ingredient still exists as a child of the ingredient spawn location, and once it is picked up it no longer is a child of the spawn location and the spawn point generates a new ingredient (see **Code 3**).

Code 3: Unity C# script spawning ingredients on ingredient removal

```csharp
public Transform spawnLoc;
public Transform ingredient;
public Vector3 localSpawnPosition = new Vector3(0.0F, 0.0F, 0.0F);
public Vector3 localSpawnRotation = new Vector3(0.0F, 0.0F, 0.0F);
private String ingredientName;

void Update()
{
    if (spawnLoc.childCount == 0)
    {
        Transform newIngredient = Instantiate(ingredient, spawnLoc.position, spawnLoc.rotation);
        newIngredient.SetParent(spawnLoc);

        newIngredient.localPosition = localSpawnPosition;
        newIngredient.rotation = Quaternion.Euler(localSpawnRotation);
        newIngredient.localScale = new Vector3(0.5F, 0.5F, 0.5F);

        // Freeze all movement of new ingredient (only moves when picked up by user)
        Rigidbody rbIngredient = newIngredient.gameObject.GetComponent<Rigidbody>();
        rbIngredient.constraints = RigidbodyConstraints.FreezePositionX | RigidbodyConstraints.FreezeRotationX
                                 | RigidbodyConstraints.FreezePositionY | RigidbodyConstraints.FreezeRotationY
                                 | RigidbodyConstraints.FreezePositionZ | RigidbodyConstraints.FreezeRotationZ;

        newIngredient.gameObject.name = ingredientName;
    }
}
```

- **Meal/Order Assembly:** meal assembly occurs as the user brings the required ingredients to the assembly counter and places it on a plate. When all ingredients for a dish have been gathered have been gathered (included in the right form, so uncut lettuce does not replace cut lettuce if cut lettuce is required), then the dish is made and should change into the correct form/model (see **Code *4***).

Code 4: Unity C# script changing a completed order into the correct model (in this case, it is a sandwich model). Note that at this point in development, the code only works with one dish type

```csharp
public Transform tSnap;
 public Transform sandwich;
```

```csharp

    public GameObject plate;
    private bool isAssbembled = false;
    LinkedList<GameObject> assembledIngredients = new LinkedList<GameObject>();
    ArrayList orderList;

    void Update()
    {
        if(assembledIngredients.Count == orderList.Count)
        {
            ArrayList checkOrderCompletion = new ArrayList(); // Verifies that the order has all needed ingredients
            checkOrderCompletion = (ArrayList)orderList.Clone();

            foreach (GameObject current in assembledIngredients)
            {
                for (int ingIndex = checkOrderCompletion.Count - 1; ingIndex >= 0; ingIndex--)
                {
                    if (String.Equals(checkOrderCompletion[ingIndex], current.name))
                    {
                        checkOrderCompletion.RemoveAt(ingIndex);
                        break;
                    }
                }
            }
            if (checkOrderCompletion.Count != 0)
            {
                return;
            }

            int ingCount = assembledIngredients.Count;
            for (int index = 0; index < ingCount; index++)
            {
                GameObject current = assembledIngredients.First.Value;
                Debug.Log(current.name);
                Debug.Log(index);
                Destroy(current);
                assembledIngredients.Remove(assembledIngredients.First);
            }
            assembledIngredients.Clear();

            Transform plateLoc = tSnap;
            Transform transformSandwich = Instantiate(sandwich, plateLoc.position, plateLoc.rotation);

            transformSandwich.SetParent(plateLoc);
            transformSandwich.localPosition = new Vector3(-0.004F, 0.028F, 0.01899996F);

            Destroy(transformSandwich.gameObject.GetComponent<Rigidbody>());
            Destroy(transformSandwich.gameObject.GetComponent<BoxCollider>());


            isAssbembled = true;
        }
    }
```

- **Game sounds:** there are various game sounds, and two levels of audio sources: game music, which plays all the time, and reaction music, which plays after a certain action has occurred. The
- **Level time limit:** the timer determines the duration of a level. This timer works by first setting a fixed amount for the level, and then using a thread to decrement that amount every second, and displaying the current time every frame (see **Code *5***). When the time reaches 0, the timer disappears and text displays on the user interface letting the user know that the level time is up. When the time is up, the user is no longer able to operate and make dishes. The scene should also fade away and display the user's final score and a summary for that level, and let the user know if they have passed the level.

Code 5: Unity C# script that sets and controls the timer for each level

```
public class Timer : MonoBehaviour
{
    private int startTime;
    public int timeLimit = 60; // Seconds in level
    public Text countdown_text;
    public GameObject gameElements;
    public AudioSource alarm;

    // Start is called before the first frame update
    void Start()
    {
        StartCoroutine("CountDown");
        Time.timeScale = 1;
    }

    // Update is called once per frame
    void Update()
    {
        if (timeLimit >= 0)
        {
            //countdown_text.text = ("" + timeLimit);
            int minutes = timeLimit / 60;
            int seconds = timeLimit % 60;
            string minutes_str = minutes.ToString("00");
            string seconds_str = seconds.ToString("00");

            countdown_text.text = ("" + minutes_str + ":" + seconds_str);
        }
        if (timeLimit <= 0)
        {
            gameElements.GetComponent<ScoreKeeper>().gameOver();
            gameElements.GetComponent<OrderManager>().endAll();
            alarm.Play();
            timeLimit = -5;
        }

    }

    IEnumerator CountDown()
    {
        while (timeLimit > -1)
        {
            yield return new WaitForSeconds(1);
            timeLimit--;
        }
    }
}
```

- **Delivering meal/order to customer:** when the order has been completed, the user will place the dish in a designated location to “deliver” the order to the customer. When the user picks up the dish to complete, the dish on the assembly counter automatically respawns (see **Code *6***). Once the dish is in the delivery location, the game will check to see if the dish matches any current orders. If it does match an order, the order is removed from the list of upcoming orders, and the user is awarded points. If the order doesn’t match any preexisting orders, the order simply disappears and nothing happens.

Code 6: Unity C# script spawning plates on plate/dish removal

```
    public Transform spawnLoc;
    public Transform spawn;
    public Vector3 localSpawnPosition = new Vector3(0.0F, 0.0F, 0.0F);
    public Vector3 localSpawnRotation = new Vector3(0.0F, 0.0F, 0.0F);
    public string spawnName;

    // Update is called once per frame
    void Update()
    {
        if (spawnLoc.childCount == 0)
        {
            Transform newSpawn = Instantiate(spawn, spawnLoc.position, spawnLoc.rotation);
            newSpawn.SetParent(spawnLoc);

            newSpawn.localPosition = localSpawnPosition;
            newSpawn.rotation = Quaternion.Euler(localSpawnRotation);
            //newSpawn.localScale = new Vector3()

            // Freeze plate
            Rigidbody rbIngredient = newSpawn.gameObject.GetComponent<Rigidbody>();
            rbIngredient.constraints = RigidbodyConstraints.FreezePositionX | RigidbodyConstraints.FreezeRotationX
                                     | RigidbodyConstraints.FreezePositionY | RigidbodyConstraints.FreezeRotationY
                                     | RigidbodyConstraints.FreezePositionZ | RigidbodyConstraints.FreezeRotationZ;

            newSpawn.gameObject.name = spawnName;
        }
    }
```

- **Scoring:** scoring happens on completing an order. For every completed order, +5 points are given. In the early stages of the game, no points will be taken away. Later in development, points will be taken away if orders are not fulfilled in a given time period. A score threshold is set for each level that specifies how many points the user needs to pass the level.

- **Meal/order variety:** meals will be randomly assigned to the user to fulfill. Different levels have different varieties of meals/orders, so that the amount of orders, and the variety of ingredients, are limited for levels. The order variety is determined pseudo-randomly when a new order is generated, based on the parameters in the level.
- **Tools and appliances:** currently there is no support for tools and appliances. But as development progresses, a knife and an oven/stove will be needed to complete orders. For the knife, the user will need to take the specific ingredient to the cutting board to cut the ingredient. Once the ingredient is cut, the user can take the newly prepared ingredient to the assembly counter. The stove/oven will be used for making the order, like soup or baked dishes. So instead of preparing individual ingredients, the user will need to take a nearly completed dish to the stove or oven to cook it before delivering it to the "customer."
- **Different levels:** currently there is no support for different levels. However, levels will be pre-created and increase in difficulty as the user progresses. Difficulty is set by time limit, the score threshold needed to pass the level, variety of dishes possible in each level, and the processing needed for those dishes (such as cutting ingredients or cooking them). Each level will have specific ingredients laid out, and only dishes requiring those ingredients will be requested in the level.

Testing and evaluation

Initial testing for this project took place on a singular laptop, tested in mechanics of the game in a non-VR environment. This allowed for testing to verify that the game elements that don't require movement, such as orders assembling when placed together, ingredients respawning, and points being awarded upon order delivery, worked before VR entered the game.

Once initial testing of the core, non-VR mechanics were tested, the VR mechanics were then tested. Most of the game elements remained unchanged, allowing for testing to focus mainly on the VR aspects, such as making sure the user could pick up items and assemble dishes, verifying the space was not too large, and the user could comfortably move around the environment and pick ingredients and dishes up, and making sure the game still worked. Through testing, some things changed, such as the room being made smaller, so that the user did not have to move too much from their starting position and risk hitting something in the real world or moving outside the limits of the VR system.

This game was also tested with various other people other than the developer, to ensure that the game made sense to people outside the project, and unfamiliar with VR. This allowed for minor changes, like fixing some aesthetics of the room, and making the colliders on the hands larger so that it was easier to pick up ingredients.

The game tested well with others, and due to the nature of its gameplay there were no issues with sickness. Future improvements to the game would be creating a start and end menu, allowing for dish preprocessing, and making some of the gameplay items, such as the Quit button and the score jar, more immersive by placing them into the game and not so obviously pieces made for a game.