Dish It Out

Artists: Arfa, Nelson
Musicians: Abdi, Shrey
Programmers: Linh, Sheryl

Overview:

A digital card game about cooking ingredients and recipes. Players compete against one another to collect ingredients cards from the market to make dishes from around the world and gain points from doing so. This strategic game allows players to carefully and quickly collect ingredients and create recipes so that they can reach 15 points first to win.

In this 2D game, players will select a cuisine style then collect ingredients accordingly to complete a series of dishes. The first player to collect a certain number of points wins. Points value vary based on the tier the recipes are in. Easier recipes require less ingredients but have fewer points while harder recipes require more ingredients and are valued with higher points. Ingredients will be presented as cards and the UI will resemble a board game where players sit across from each other and be able to observe others' cards/actions. This turn-based strategy card game forces players to strategize which ingredients to get and how that helps them get the recipes they want as their opponent could be competing for the same recipe.

Game Rules:

First, select a region to begin. Two players take turns to play. When it's your turn, you will have three options:

- select 3 different ingredient cards
- select 1 ingredient card to get 2 of the same ingredient
- or purchase a recipe card

To buy a recipe card, you must already have all of the ingredients that the recipe calls for, this will be listed at the bottom left of each recipe card

On the left of each recipe card, you'll find listed the ingredients you need to have in your inventory to be able to buy that recipe.

Each recipe card is worth a certain number of points. Keep an eye out for this on the top right corner of each card.

To win, you must reach a total of 10 points before your opponent does.

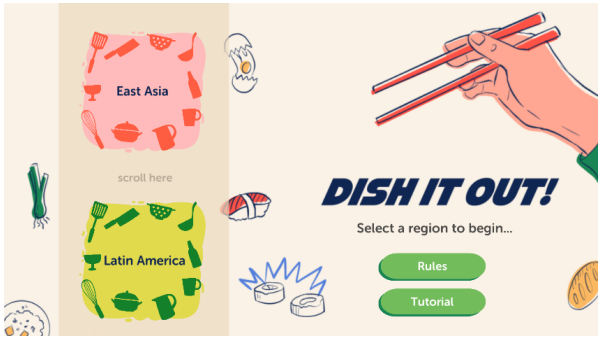
Code Structure:

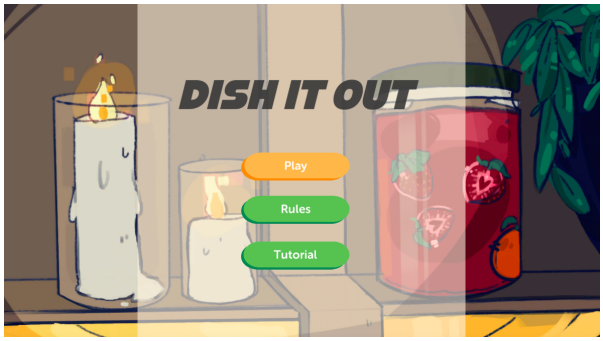
- Title screen scene
    - Region selection
    - Tutorial videos
- Game scene
    - Player turns
    - Shuffle recipe cards
    - Inventory
        - Player score
        - Player ingredients
        - Player recipes
    - Ingredient
        - Add ingredient to inventory function
    - Recipe
        - Purchase recipe
            - Check ingredients needed function
            - Deduct ingredients from inventory function
            - Add recipe to inventory function
    - Recipe Scriptable Objects
        - Properties of each recipe (name, tier, points, ingredients, etc)

Programming Goals:

Title Screen

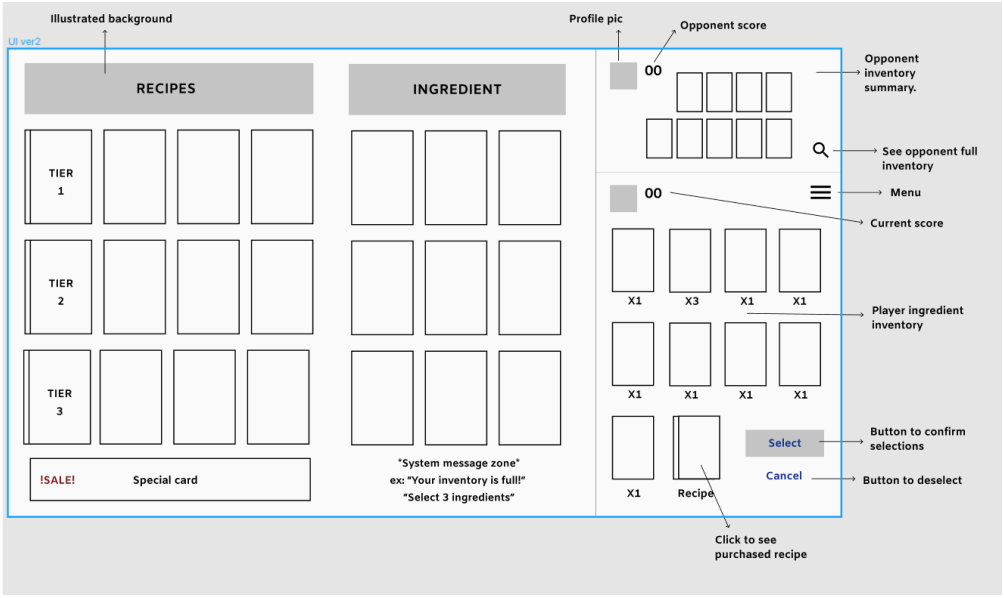
The minimum goal for the title screen was the region select, rules page, and tutorial. With some title screen reiteration, the region select scroll area became an animated panel for players to pick what region they're interested in playing. The minimalist style allowed for creative ways to juice up the title screen with animations. The title screen also had to go through different design iterations. The old format was more geared towards mobile as the scroll area allowed players to swipe through to see the different regions offered, however, with the placeholder art replaced, the screen had to be redesigned with download play in mind.





Game Scene

The technical goals for the game was a panel of selectable ingredients, a panel of selectable recipes, an interactive motherly cat guiding the player, inventory panel, special cards, and main menu. However, the project was overscoped and the special card feature had to be cut in order to prevent the quality from dropping. The game also had to go through multiple iterations of UI layout redesign since the game was originally intended to be for mobile. This meant cards had to be big for readability and the UI layout had to be simplified and compartmentalized instead of everything shown all at once which contributed to clutter confusion.



Code Callouts:

Vital scripts consist of inventory, UIController, Recipe, and RecipeInfo. The inventory was managed by two dictionaries: ingredients and recipes. The dictionaries were under the player class which we could make as many instances as needed. Simple dictionary methods were made to manage the inventory. The player class also kept track of scoring and the total number of recipes players have accumulated. This script is not is not groundbreaking, as it just uses simple dictionary and dictionary methods, but we utilized indices instead strings to prevent hardcoding ingredients and recipes names. We wanted to reuse and apply this script with other regional dishes so this was the best solution we came up with.

```
public class Player : MonoBehaviour
{
    //the player class is for managing each player's cards
    [Header("Ingredients")]
    public GameObject ing1;
    public GameObject ing2;
    public GameObject ing3;
    public GameObject ing4;
    public GameObject ing5;
    public GameObject ing6;
    public GameObject ing7;
    public GameObject ing8;
    public GameObject ing9;

    [Header("Dictionaries")]
    public Dictionary<int, int> ingredientDict = new Dictionary<int, int>();
    public Dictionary<GameObject, int> recipeDict = new Dictionary<GameObject, int>();

    [Header("Other")]
    public int score;
    public int numRecipes;

    // Start is called before the first frame update
    void Start()
    {
        //setup
        score = 0;
        numRecipes = 0;

        ingredientDict[0] = 0;
        ingredientDict[1] = 0;
        ingredientDict[2] = 0;
        ingredientDict[3] = 0;
        ingredientDict[4] = 0;
        ingredientDict[5] = 0;
        ingredientDict[6] = 0;
        ingredientDict[7] = 0;
        ingredientDict[8] = 0;
        ingredientDict[9] = 0; //for dough / tortilla
        ingredientDict[10] = 0; //for soup
    }
```

```
    //takes in the card index and adds to the total quantity
    public void selectIngredients(int card)
    {
        ingredientDict[card] += 1;
        //Debug.Log("Card val:" + ingredientDict[card]);
    }

    //dictionary method for checking ingredients
    public void containsIngredient(int card)
    {
        if (ingredientDict.ContainsKey(card) == true)
        {
            Debug.Log("Key is found.");
        }
        else
        {
            Debug.Log("Key is not found.");
        }
    }

    //dictionary method for checking recipe
    public void containsRecipe(GameObject card)
    {
        if (recipeDict.ContainsKey(card) == true)
        {
            Debug.Log("Key is found.");
        }
        else
        {
            Debug.Log("Key is not found.");
        }
    }
```

The UIController is the main skeleton of the game. Because the game is all screen interactions, the UIController takes care of all calls relating to the UI elements. All the cards on screen are toggles and are part of toggle groups. Since the interactions are from buttons, the turn-based structure is also located in this script. The structure ensures that the player makes a valid move before changing to the other player's turn.

```
// Update is called once per frame
void Update()
{
    //turn-base structure
    if (yourTurn == true)
    {
        //player 1
        if (moveMade1 == true)
        {
            //Debug.Log("you moved");
            yourTurn = false;
            moveMade1 = false;
            player2Turn.SetTrigger("popUp");
            invplayer1Toggle.isOn = false;
            invplayer2Toggle.isOn = true;
            turnIndicator.text = "P2";
        }
    }
    else
    {
        //player 2
        if (moveMade2 == true)
        {
            //Debug.Log("opponent moved");
            yourTurn = true;
            moveMade2 = false;
            player1Turn.SetTrigger("popUp");
            invplayer2Toggle.isOn = false;
            invplayer1Toggle.isOn = true;
            turnIndicator.text = "P1";
        }
    }
    //Debug.Log(yourTurn);
    //yourTurn = true;
}
```

SelectButtonPressed function is called from the onClick of the Select Button in the ingredients panel. It first checks whose turn it is before calling the SelectIngredient function which returns a boolean to denote the ending of the turn.

```
//when button is pressed, pass into correct parameters to update correct player's ingredient inventory
public void SelectButtonPressed()
{
    if (yourTurn == true)
    {
        //if p1 ingredients are selected, your move was made
        bool result = SelectIngredients(player1, p1IngCount);
        if (result)
        {
            moveMade1 = true;
        }
    }
    else
    {
        //if p2 ingredients are selected, their move was made
        bool result = SelectIngredients(player2, p2IngCount);
        if (result)
        {
            moveMade2 = true;
        }
    }
}
```

SelectIngredients function first checks what ingredients are selected and validates if it's a legal move.

```
//this function updates selected ingredients into respective player's inventory
public bool SelectIngredients(Player play, Text[] ingredientCountList)
{
    selected = 0;

    //check what ingredient toggles were selected
    Toggle[] ingToggles = ingredientGroup.GetComponentsInChildren<Toggle>();
    for (int i = 0; i < ingToggles.Length; i++)
    {
        //if tog is on, check if player selected over 3 ingredients (illegal move)
        if (ingToggles[i].isOn)
        {
            selected += 1;
            if (selected >= 4)
            {
                wrongAudio.Play();
                mochiText.text = "Slow down little chef! You can only select 3 or 1 ingredient card each turn.";
                return false;
            }
        }
    }
```

```
//if 3 ingredients are selected, add to inventory and update ui text
if (selected == 3)
{
    actionAudio.Play();
    for (int i = 0; i < ingToggles.Length; i++)
    {
        if (ingToggles[i].isOn)
        {
            play.selectIngredients(i);
            ingredientCountList[i].text = play.ingredientDict[i].ToString();
        }
    }
    mochiText.text = "3 ingredients added to your inventory!";
} else if (selected == 2) //if 2 selected, prompt that either 1 or 3 should be selected
{
    wrongAudio.Play();
    mochiText.text = "Hmmm...you only selected 2. Pick 3 or 1 ingredient!";
    return false;
} else if (selected == 1) //if 1 selected, add 2 of that ingredient to the inventory and update ui text
{
    actionAudio.Play();
    for (int i = 0; i < ingToggles.Length; i++)
    {
        if (ingToggles[i].isOn)
        {
            play.selectIngredients(i);
            play.selectIngredients(i);
            ingredientCountList[i].text = play.ingredientDict[i].ToString();
        }
    }
    mochiText.text = "2 of the same ingredient added to your inventory!";
}

//deselect all toggles (reset for next turn)
for (int i = 0; i < ingToggles.Length; i++)
{
    ingToggles[i].isOn = false;
}
selected = 0;
return true;
```

```
//general function that takes in a tier of toggles and its respective recipes to shuffle
public void ShuffleAllCards(Recipe[] tierList, RecipeInfo[] recipeList, int totalCards)
{
    foreach (Recipe tog in tierList)
    {
        int randomRecipe = Random.Range(0, totalCards);
        tog.infoScript = recipeList[randomRecipe];
    }
}

//general function to replace the old card with a new one
public void ReplaceCard(Recipe togRecipe, int tier)
{
    if (tier == 1)
    {
        int randomRecipe = Random.Range(0, 2);
        togRecipe.infoScript = tier1Recipes[randomRecipe];
        Debug.Log("Changed to " + tier1Recipes[randomRecipe].name);
    }
    else if (tier == 2)
    {
        int randomRecipe = Random.Range(0, 3);
        togRecipe.infoScript = tier2Recipes[randomRecipe];
        Debug.Log("Changed to " + tier2Recipes[randomRecipe].name);
    }
    else if (tier == 3)
    {
        int randomRecipe = Random.Range(0, 3);
        togRecipe.infoScript = tier3Recipes[randomRecipe];
        Debug.Log("Changed to " + tier3Recipes[randomRecipe].name);
    }

    togRecipe.UpdateCardToggle();
}
```