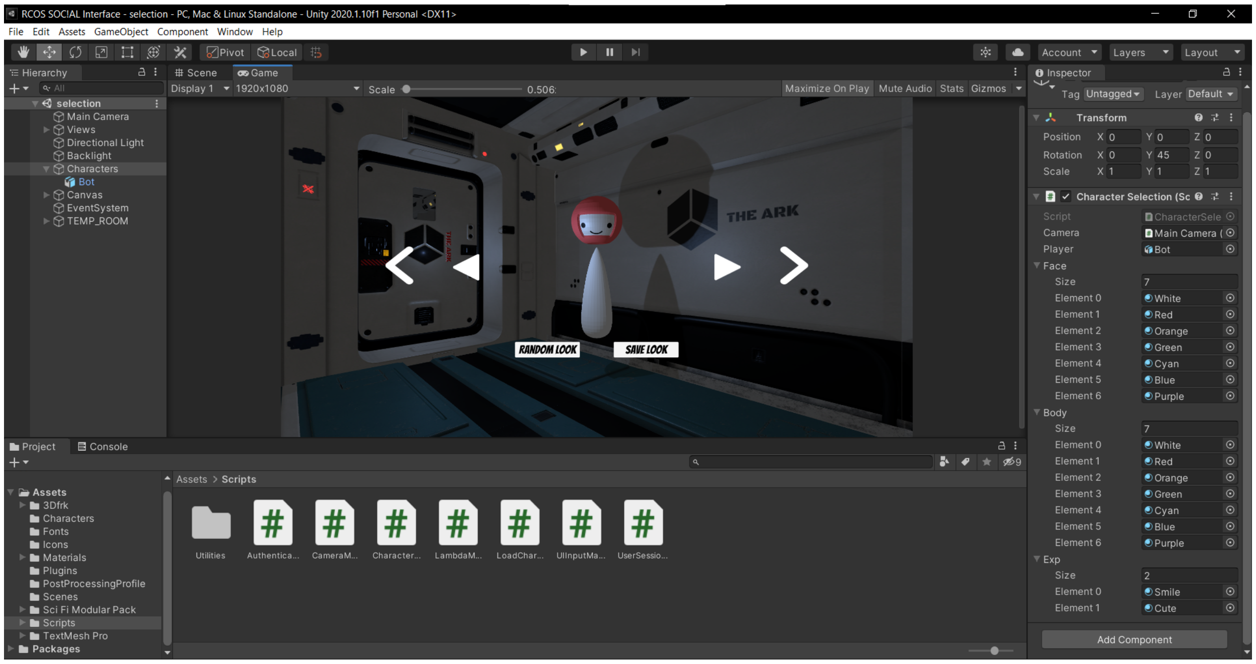
<u>User Selection Interface</u>

- Adding new textures:
    - Click “characters” inside the hierarchy
    - Increase the size of the corresponding section
    - Add the materials



Tag Untagged Layer Default

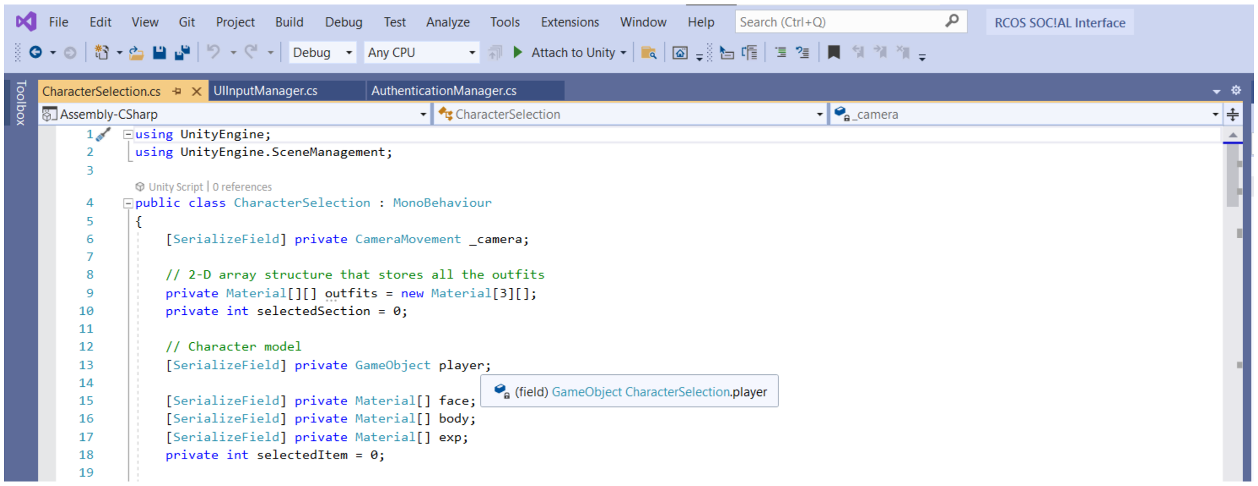
Transform

Position	X 0	Y 0	Z 0
Rotation	X 0	Y 45	Z 0
Scale	X 1	Y 1	Z 1

Character Selection (Sc

Script	CharacterSele
Camera	Main Camera (
Player	Bot
Face	
Size	7
Element 0	White
Element 1	Red
Element 2	Orange
Element 3	Green
Element 4	Cyan
Element 5	Blue
Element 6	Purple
Body	
Size	7
Element 0	White
Element 1	Red
Element 2	Orange
Element 3	Green
Element 4	Cyan
Element 5	Blue
Element 6	Purple
Exp	
Size	2
Element 0	Smile
Element 1	Cute

- Add new components:
    - Locate CharacterSelection.cs
    - Add `[SerializedField] private TYPE[] COMPONENTNAME;`
    - Do not forget initialize it in `Start()`
    - If the type is not simple textures (such as new props), the set-up process may look different compared to the existing codes (make it active/inactive might be one solution if they are `GameObjects`).



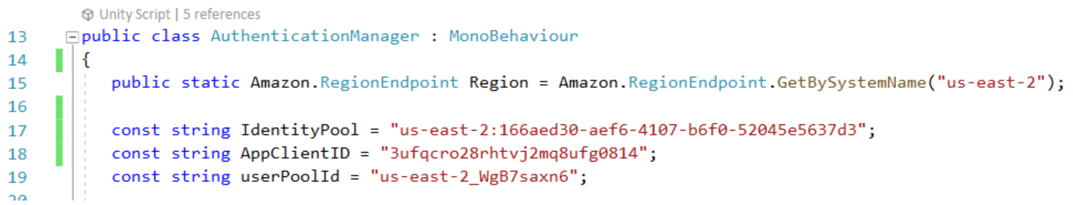
```
using UnityEngine;
using UnityEngine.SceneManagement;

public class CharacterSelection : MonoBehaviour
{
    [SerializeField] private CameraMovement _camera;

    // 2-D array structure that stores all the outfits
    private Material[][] outfits = new Material[3][];
    private int selectedSection = 0;

    // Character model
    [SerializeField] private GameObject player;

    [SerializeField] private Material[] face;
    [SerializeField] private Material[] body;
    [SerializeField] private Material[] exp;
    private int selectedItem = 0;

```

- Save Player Preferences
    - Inside CharacterSelection.cs, the preferences is saved as an array of integers each indicating the specific choice of each component.
    - Everytime the user changes the look, the saved data is also changed.
    - `PlayerData[selectedSection] = selectedItem;`
    - **Data syncing using AWS service is not completed due to limited time. Therefore, LoadCharacter.cs may only have limited functionality.**
        - **Because the original Cognito Sync Service used for data syncing inside AWS Cognito is removed from the newest package, one solution I can think of but have not implemented is to create a pool with custom attributes that can save user preferences.**

Link to the unity package:
https://drive.google.com/file/d/1XPqtLFpJLS6r5fz85Wa0SrqfTLsAWv_Y/view?usp=sharing

Login / Sign-up Interface

- Current functionality
    - Existed user can use the email address and the password to login
    - New users will have to create a new account using a preferred email address and username. A password (requires at least 8 characters that also include both upper and lower cases) need to be created in order to complete the sign-up process. The user will then receive a confirmation email to verify the email address. After verifying the email address, the user will be able to login successfully.
    - An error prompt will show up helping the user if there is an unsuccessful login / sign-up attempt. It includes a coroutine and lock that creates the fade out effect.
    - A local refresh token is also created and saved so that the user will not need to go through the login process again after a certain amount of time.
    - The user should be directed to the character selection interface after logging in.
- UIInputManager.cs
    - Controls the UI elements on the screen when the application starts.
- UserSessionCache.cs
    - A small class that saves the login data as JSON
- LambdaManager.cs
    - **MAY HAVE BUGS**
    - **This project does not require the use of lambda functions**
    - Directly calls the lambda function for a specific user pool
- AuthenticationManager
    - Uses AWS Cognito and communities with the Amazon service and allows the actual login / signup process.
    - The following need to be changed in case a new user / identity pool is wanted
        - `Region`: the AWS region of where your services live, can be found inside the identity pool and user pool id
        - `IdentityPool`: Cognito identity pool id, can be found under Cogito->Mange User Pools->YOUR POOL->General Setting
        - `AppClientID`: Cognito app client id, can be found under Cogito->Mange User Pools->YOUR POOL->General Setting->App Clients
        - `UserPoolID`: Cognito user pool id, can be found under Cogito->Mange User Pools->YOUR POOL->General Setting



```
public class AuthenticationManager : MonoBehaviour
{
    public static Amazon.RegionEndpoint Region = Amazon.RegionEndpoint.GetBySystemName("us-east-2");

    const string IdentityPool = "us-east-2:166aed30-aef6-4107-b6f0-52045e5637d3";
    const string AppClientID = "3ufqcro28rhtvj2mq8ufg0814";
    const string userPoolId = "us-east-2_WgB7saxn6";
```