AET 339 Technical Art I: Project 3
Khel Bagdasaryants, Jaclyn Ngo, Nada Abu-Rayyan
Spring 2021
Link To Project

Project Details:

The studio is building a Wizard MOBA game and needs you to build out magical spell effects for their initial character set. The effects needed are as follows (one wizard per team member).

Character 1: Winter Wizard
Effects: Snow Shield (shield ability) and Frozen Arrow (ranged melee)

Character 2: Poison Wizard
Effects: Poison Cloud (shield ability) and Sludge Shot (ranged AOE attack)

Character 3: Electric Wizard
Effects: Shocking Charge (shield ability) and Lightning Bolt (ranged melee)

Character 4: Charm Wizard
Effects: Captivating Smile (shield ability) and Love Is A Battlefield (ranged AOE attack)

Requirements:

- *FX must have some variety in their use/not be the same every time you cast a spell.*
- *FX must be complementary in style but distinct between characters.*
- *Must include sound*
- *Demo scene must make it possible to cycle between characters and test their spells. Characters and environments do not need to be interactive. Demo scene should be simple to showcase the effects.*

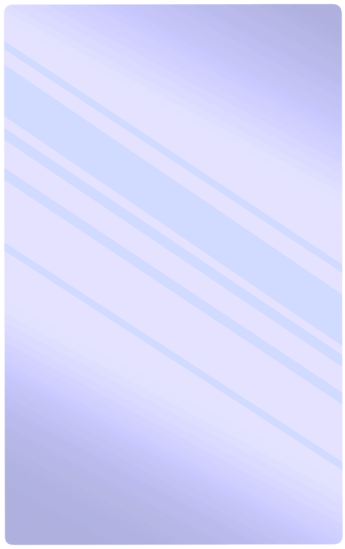
Assets Used:

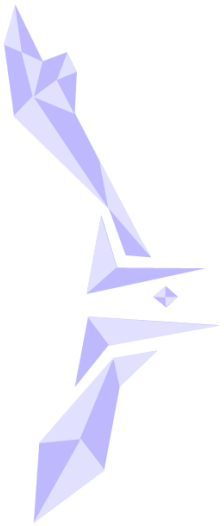
- Fairy Magic Wand, by Robinhood76
  https://freesound.org/people/Robinhood76/sounds/342432/
- Another Magic Wand Spell Tinkle, by Timbre
  https://freesound.org/people/Timbre/sounds/221683/
- Short Kiss, by Vospi
  https://freesound.org/people/Vospi/sounds/344209/
- Laser Gun Explosion, by Dpoggioli
  https://freesound.org/people/Dpoggioli/sounds/213610/
- Magic Spell, by Kostas17
  https://freesound.org/people/Kostas17/sounds/542825/
- Smoke Particle, by Kenney

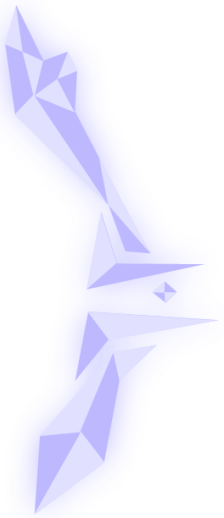
https://kenney.itch.io/kenney-game-assets-1

- Rain Particle, by GreenMask Games
  https://i.ibb.co/qgfgynJ/rainDrop.png
- Sci-Fi Forcefield, by StormwaveAudio
  https://freesound.org/people/StormwaveAudio/sounds/330629/
- Wind, by InspectorJ
  https://freesound.org/people/InspectorJ/sounds/376415/
- Slime, by jtap97
  https://freesound.org/people/jtap97/sounds/448889/

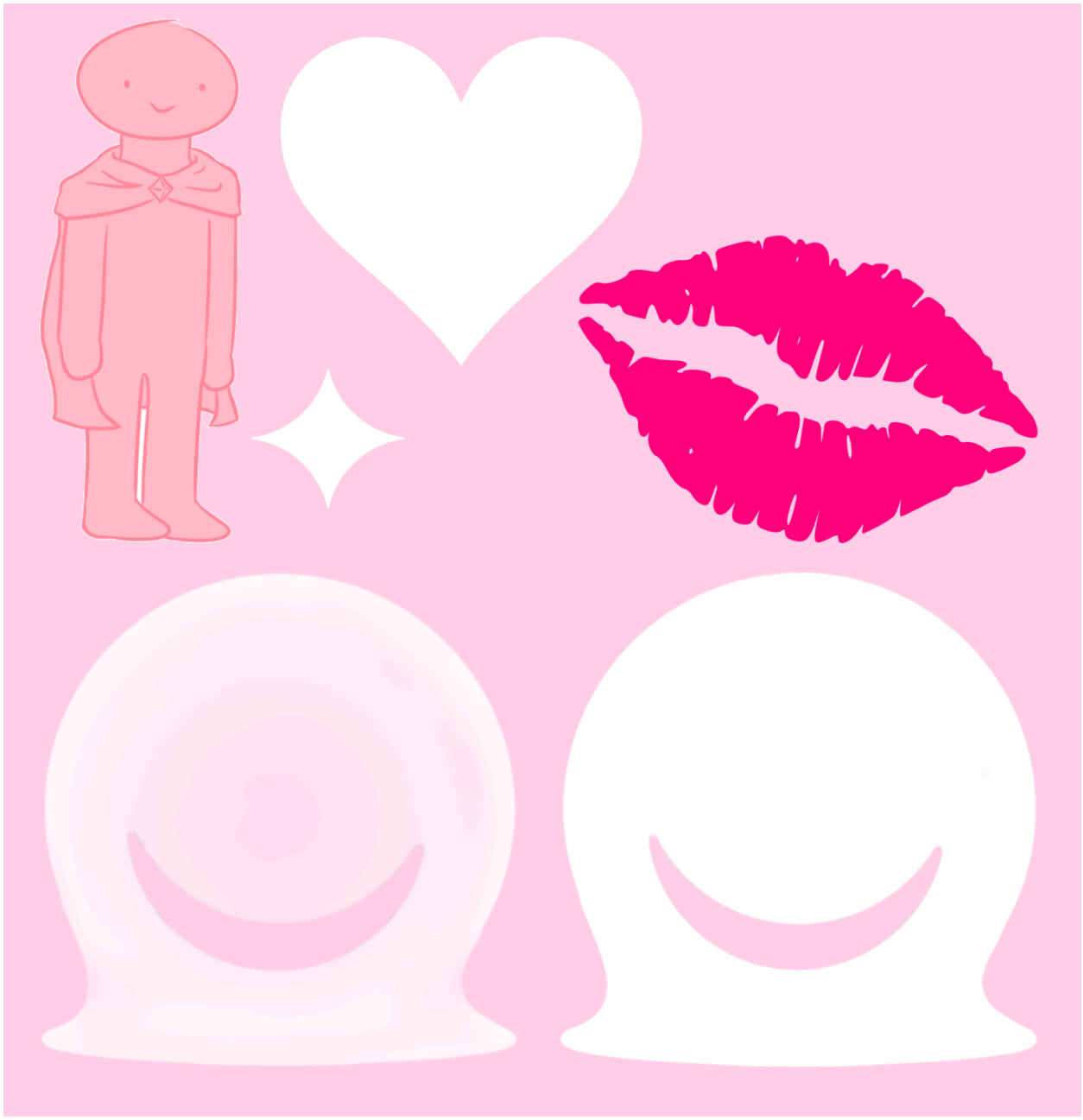
Winter Wizard Assets (created in Clip Studio Paint)



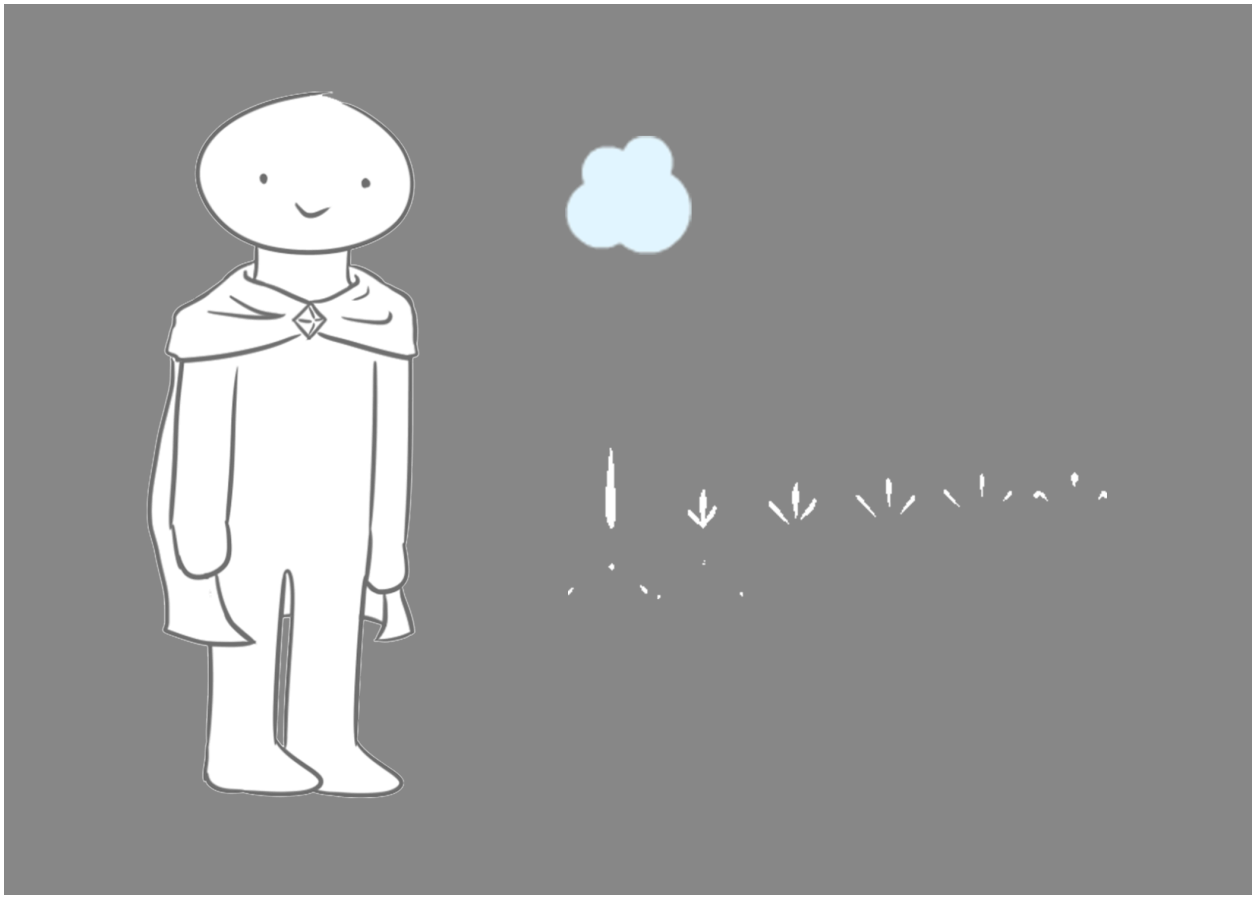


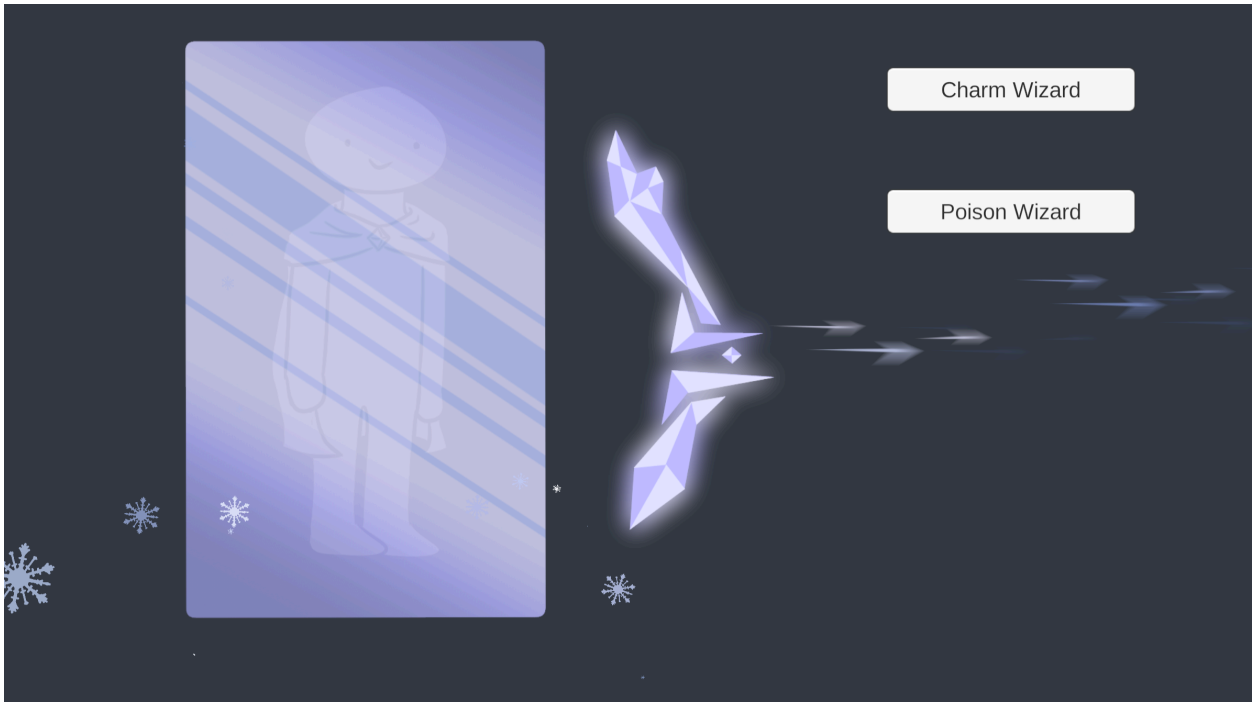


Charm Wizard Assets (created in photoshop)



Poison Wizard Assets (Gathered from Kenney Assets and GreenMask Games):





Video showing functionality: https://www.youtube.com/watch?v=ydsebPIj82M

Analysis:
Based on the required features, a few questions were asked:

- *How should the spells be activated?*
- *How should we swap out characters?*
- *How should the animation state for the spells function?*
- *How do we handle the special effects for each spell?*
- *How should the sound effects play once the spell activates?*
- *How do we make our spells juicy?*

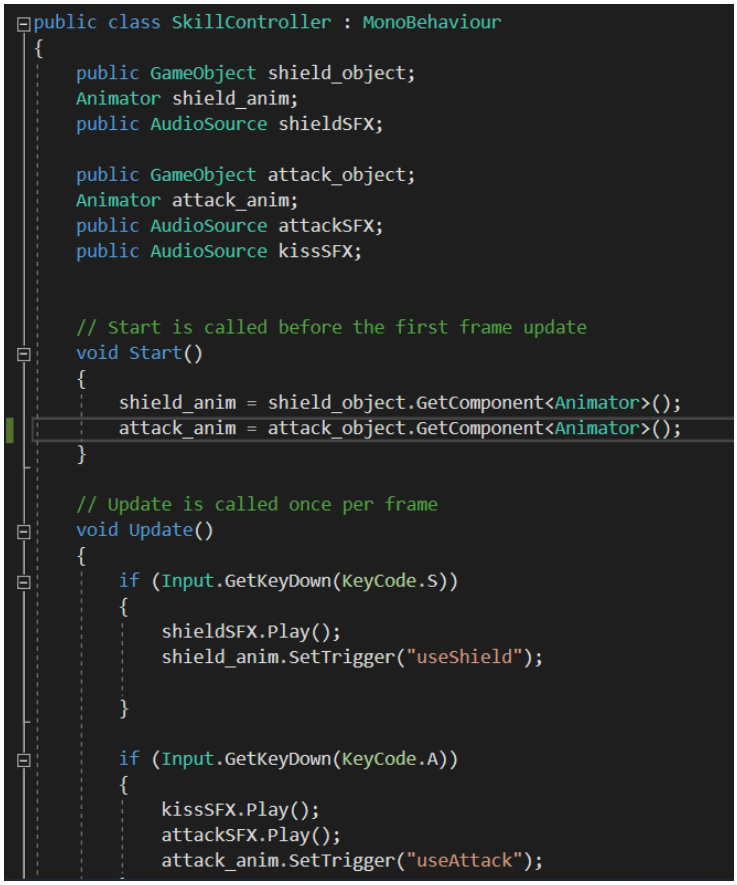
Outside of Unity, we can use Photoshop and a variety of other drawing tools to create transparent art assets to use to make our spells in Unity.

Approach:
After research and testing particle systems, animator controller, and sprites/images, the following approach was chosen:

- *A script that activates the shield and attack spells on certain button press*
- *A script that changes the active character/scene*
- *An animator controller that animates a spell (shield, special symbols, etc)*
- *A particle system to create special effects for both the shield and attack spells*
- *Sound FX from freesound.org that triggers once a spell is activated.*
- *A canvas to add buttons to other wizards*

Breakdown of Approach:

Scripts functionality:



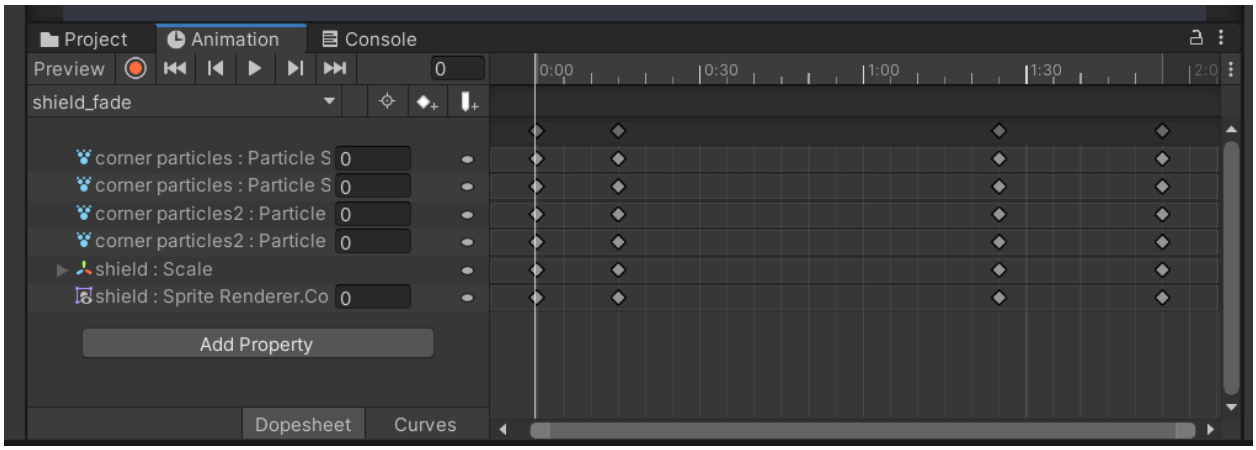
```
public class SkillController : MonoBehaviour
{
    public GameObject shield_object;
    Animator shield_anim;
    public AudioSource shieldSFX;

    public GameObject attack_object;
    Animator attack_anim;
    public AudioSource attackSFX;
    public AudioSource kissSFX;


    // Start is called before the first frame update
    void Start()
    {
        shield_anim = shield_object.GetComponent<Animator>();
        attack_anim = attack_object.GetComponent<Animator>();
    }

    // Update is called once per frame
    void Update()
    {
        if (Input.GetKeyDown(KeyCode.S))
        {
            shieldSFX.Play();
            shield_anim.SetTrigger("useShield");

        }

        if (Input.GetKeyDown(KeyCode.A))
        {
            kissSFX.Play();
            attackSFX.Play();
            attack_anim.SetTrigger("useAttack");
```

SkillController is a script that activates the sound effects and animation on keypress. Each spell (shield and attack), needs at least three variables: a game object that has the animator controller attached to it, a variable to contain the animator, and an audio source. In the start function, get the animator component from the game object that has the attached animator controller. Save it to a variable. We'll use this in update. In update, use if statements to determine when a certain key is pressed. If it's pressed, play the audio source and play the animation.

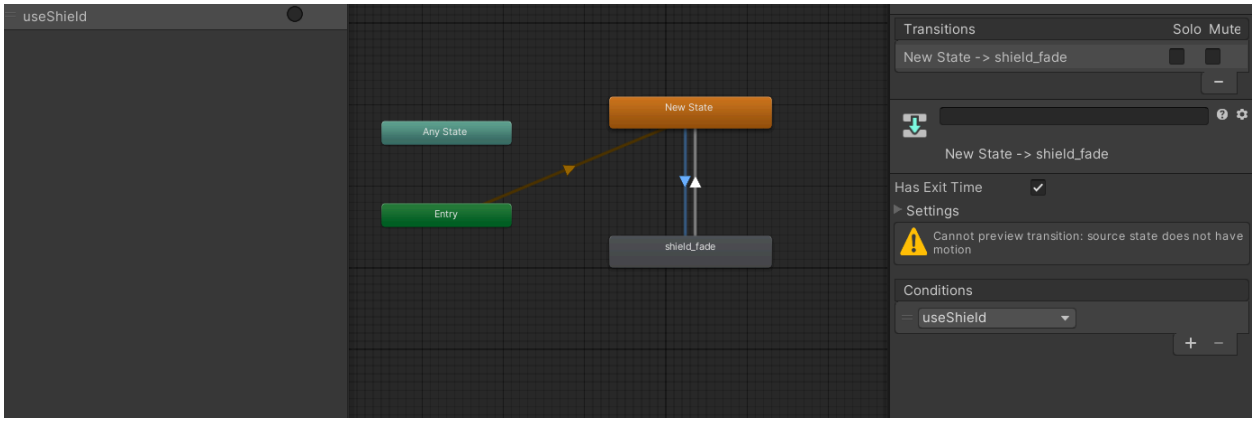
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class ButtonLoad : MonoBehaviour
{
    public string sceneName;
    public void LoadScene()
    {
        SceneManager.LoadScene(sceneName);
    }
}
```

ButtonLoad is a script that just takes a string and load the respective scene according to the string. The scenes have to be in the build settings in order for the scene to load.

Animator Controller functionality:

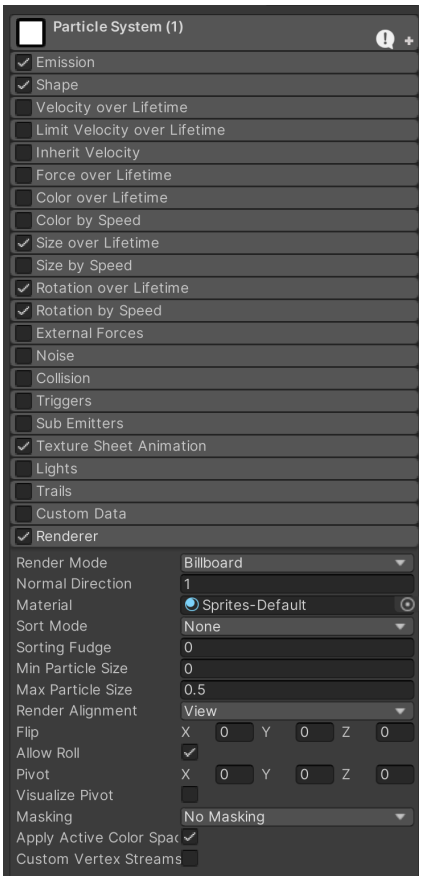


All the animators in Unity look something similar to this. We decided that in order to make our spells consistent, we will animate the transparency to fade in/out the spells. Scale, position, and rotation were also a couple of other factors we changed per spell. We also animated when the particle systems should appear and disappear. Additional flairs were also added in the animator such as multiple images for our spell and glow effects.



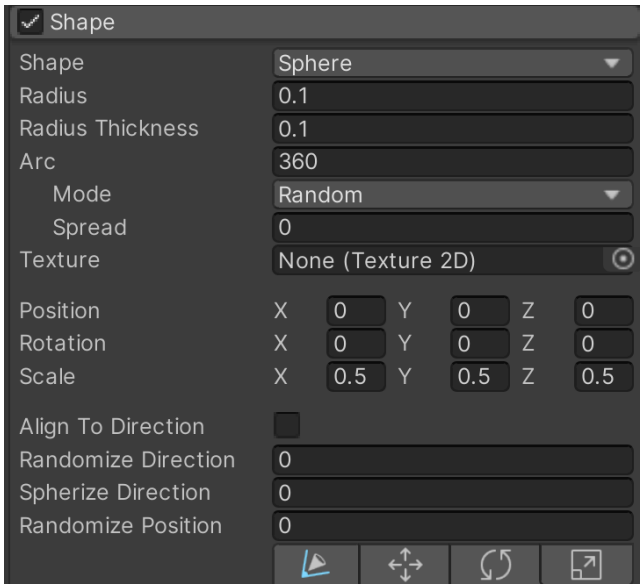
In the Animator tab, we have the animation states. Both our shield and attack states are formatted the same way. In order to prevent the animation from playing when awake, an empty state is set as default. A transition is created and connects to the animation state. In order to move from empty state to animation state, a trigger (useShield or useAttack) is made. When the conditions are met (in other words the trigger is activated), then empty state will transition to animation state. Once the animation finishes playing, it will automatically go back to empty state and awake for the key press again. Looping is also turned off.

Particle system breakdown:

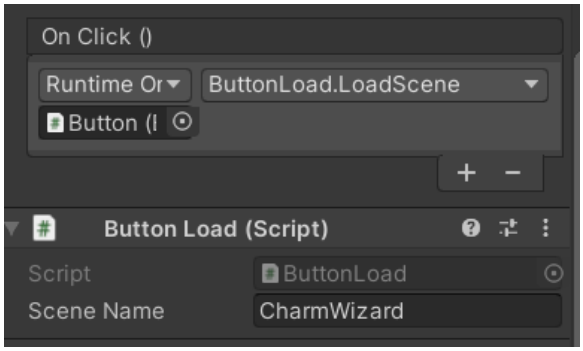


We use Unity's particle system along with custom 2D assets to accomplish many of the special effects used in our spells. For creating a 2D particle system, scroll down in the inspector down to the renderer section and change the material to "Sprites-Default" and then enable texture sheet animation. Enabling texture sheet animation will allow 2D assets to be chosen as the texture for the particle. Once our particles were created and adjusted, they were linked up with the animator to fade in and out upon spell activation. Each component that makes up a particle can be adjusted from speed, color, rotation, to emission rate. One of the most important

components to be changed is the shape section which is crucial for differentiating certain effects. A sphere shape is good for particles that float around an object within a certain radius such sparkles as around a shield whereas a cone with a small angle can allow particles to shoot out like projectiles such as in the case of the Ice Wizard's attack.

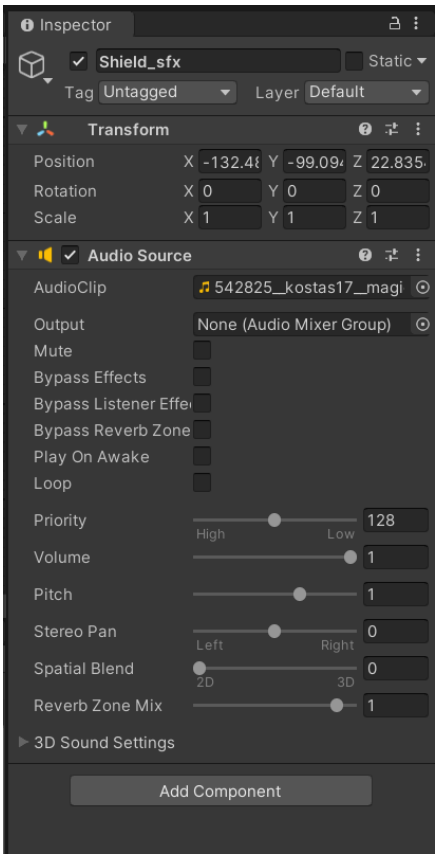


Canvas Functionality:



We used buttons to move from one wizard to another. Add a button and add the ButtonLoad script to it. In the public variable Scene Name, type in the name of the scene character for character. In OnClick(), add a functionality. Drag and drop the button object into the slot under Runtime. To the right of Runtime, select the script ButtonLoad and function LoadScene. The button should now load the scene specified in the string variable.

Sound FX breakdown:



All sound effects follow the same method. An empty game object is created and the audio source component is added. Then the audio is dropped into the AudioClip section. Loop and Play on Awake are deselected to ensure that the audio doesn't play right away or continuously. Volume is also adjusted as needed. These game objects are then dragged and dropped into the public audio source variables for the SkillController script to use.