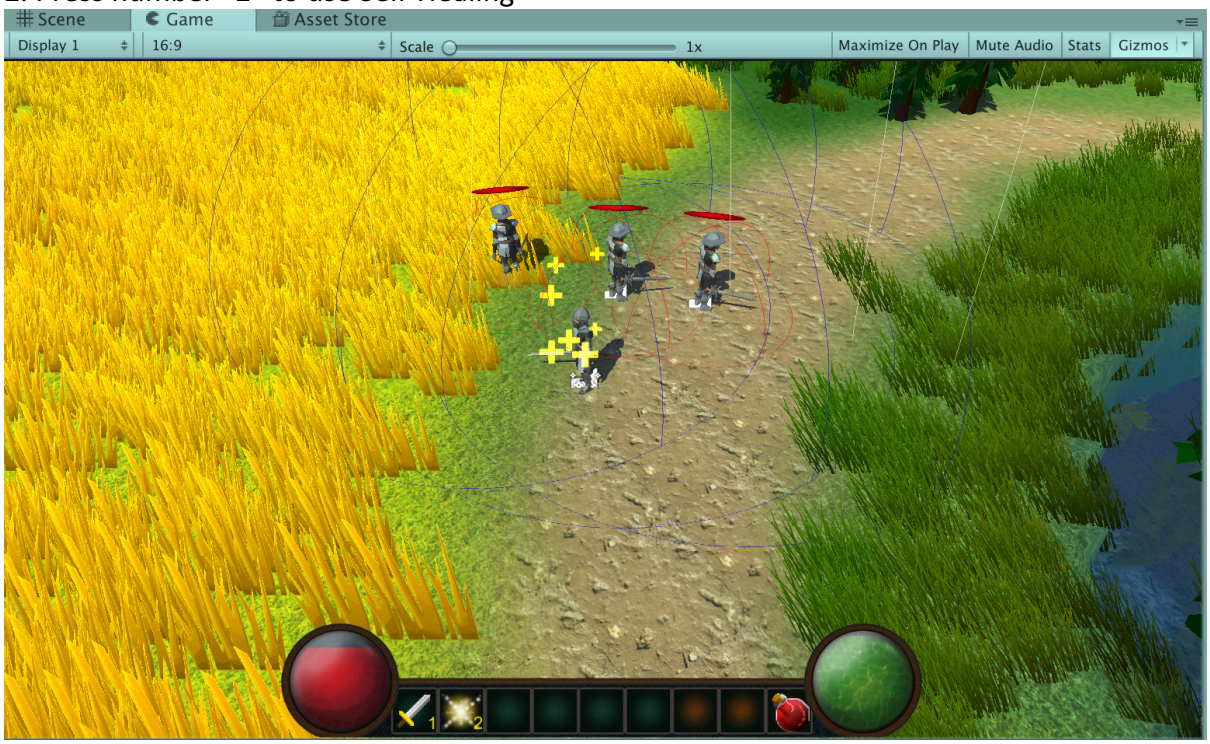
Assignment #3:
Question: How to design special abilities for player, like Area of Effect, Self-Healing?
Key Words: Config & Behaviours, Area of Effect, Self-Healing

In the playable Demo:
1. Press number “1” to use AOE
2. Press number “2” to use Self-Healing



For this assignment, I added two special abilities to the player: Area of Effect & Self-Healing.
Firstly, I created two commonly used classes for special abilities:
AbilityConfig: This class is used for creating settings for abilities.
AbilityBehaviour: This class creates specific functions for abilities.

AbilityConfig.cs

```
using UnityEngine;
using RPG.Utility;

namespace RPG.Character
{
    public abstract class AbilityConfig : ScriptableObject
    {
        [Header("Special Ability General")]
        [SerializeField] float energyCost = 10f;
        [SerializeField] GameObject particlePrefab = null;
        [SerializeField] AnimationClip abilityAnimation;
        [SerializeField] AudioClip[] audioClips = null;

        protected AbilityBehaviour behaviour;

        public abstract AbilityBehaviour GetBehaviourComponent(GameObject
objectToattachTo);
```

```
        public void AttachAbilityTo(GameObject objectToattachTo)
        {
            AbilityBehaviour behaviourComponet =
GetBehaviourComponent(objectToattachTo);
            behaviourComponet.SetConfig(this);
            behaviour = behaviourComponet;
        }

        public void Use(GameObject target)
        {
            behaviour.Use(target);
        }

        public float GetEnergyCost()
        {
            return energyCost;
        }

        public GameObject GetParticlePrefab()
        {
            return particlePrefab;
        }

        public AnimationClip GetAbilityAnimation()
        {
            return abilityAnimation;
        }

        public AudioClip GetRandomAudioClip()
        {
            return audioClips[Random.Range(0, audioClips.Length)];
        }

    }

}
```

AbilityBehaviour.cs

```
using System.Collections;
using UnityEngine;

namespace RPG.Character
{
    public abstract class AbilityBehaviour : MonoBehaviour
    {
        protected AbilityConfig config;

        const string ATTACT_TRIGGER = "Attack";
        const string DEFAULT_ATTACK_STATE = "DEFAULT ATTACK";
        const float PARTICLE_CLEAN_UP_DELAY = 7f;

        public abstract void Use(GameObject target = null);

        public void SetConfig(AbilityConfig configToSet)
        {
            config = configToSet;
        }
```

```
        protected void PlayParticleEffect()
        {
            var particlePrefab = config.GetParticlePrefab();
            var particleObject = Instantiate(
                particlePrefab,
                transform.position,
                particlePrefab.transform.rotation
                );

            // TODO decide if particle system attaches to player
            particleObject.transform.parent = transform; // set world space is
prefab if required

            particleObject.GetComponent<ParticleSystem>().Play();
            StartCoroutine(DestoryParticleWhenFinished(particleObject));
        }

        IEnumerator DestoryParticleWhenFinished(GameObject particlePrefab)
        {
            while (particlePrefab.GetComponent<ParticleSystem>().isPlaying)
            {
                yield return new WaitForSeconds(PARTICLE_CLEAN_UP_DELAY);
            }
            Destroy(particlePrefab);
            yield return new WaitForEndOfFrame();
        }

        protected void PlayAbilityAnimation()
        {
            var animatorOverrideController =
GetComponent<Character>().GetOverrideController();
            var animator = GetComponent<Animator>();
            animator.runtimeAnimatorController = animatorOverrideController;
            animatorOverrideController[DEFAULT_ATTACK_STATE] =
config.GetAbilityAnimation();
            animator.SetTrigger(ATTACT_TRIGGER);
        }

        protected void PlayAbilitySound()
        {
            var abilitySound = config.GetRandomAudioClip();
            var audioSource = GetComponent<AudioSource>();
            audioSource.PlayOneShot(abilitySound);
        }
    }
}
```

Next work is designing functions and configs for abilities:
Part 1: Designing for Area Effect config and Behaviour:
AreaEffectConfig.cs

```
using UnityEngine;

namespace RPG.Character
{
    [CreateAssetMenu(menuName = ("RPG/Special Ability/Area Effect"))]
```

```
    public class AreaEffectConfig : AbilityConfig
    {
        [Header("Area Effect Specific")]
        [SerializeField] float radius = 5f;
        [SerializeField] float damageToEachTarget = 15f;

        public override AbilityBehaviour GetBehaviourComponent(GameObject
objectToattachTo)
        {
            return objectToattachTo.AddComponent<AreaEffectBehaviour>();
        }

        public float GetDamageToEachTarget()
        {
            return damageToEachTarget;
        }

        public float GetRadius()
        {
            return radius;
        }

    }
}
```

AreaEffectBehaviour.cs

```
using UnityEngine;
using RPG.Utility;
using System;

namespace RPG.Character
{
    public class AreaEffectBehaviour : AbilityBehaviour
    {

        public override void Use(GameObject target)
        {
            DealAOEDamage();
            PlayParticleEffect();
            PlayAbilitySound();
            PlayAbilityAnimation();
        }

        private void DealAOEDamage()
        {
            //static sphere cast for target
            RaycastHit[] hits = Physics.SphereCastAll(
                transform.position,
                (config as AreaEffectConfig).GetRadius(),
                Vector3.up,
                (config as AreaEffectConfig).GetRadius());

            // for each hit
            // if damageable
            // deal damage to target + player base damage
            foreach (RaycastHit hit in hits)
            {
```

```
                var damageable =
hit.collider.gameObject.GetComponent<HealthSystem>();
                bool hitPlayer =
hit.collider.gameObject.GetComponent<PlayerControl>();
                if (damageable != null && !hitPlayer)
                {
                    var damageToDeal = (config as
AreaEffectConfig).GetDamageToEachTarget();
                    damageable.TakeDamage(damageToDeal);
                }
            }
        }

    }
}
```

Part 2: Designing for Self-Healing config and Behaviour:
SelfHealconfig.cs

```
using UnityEngine;

namespace RPG.Character
{
    [CreateAssetMenu(menuName = ("RPG/Special Ability/Self Heal"))]
    public class SelfHealConfig : AbilityConfig
    {
        [Header("Self Heal Specific")]
        [SerializeField] float extraHealth = 50f;

        public override AbilityBehaviour GetBehaviourComponent(GameObject
objectToattachTo)
        {
            return objectToattachTo.AddComponent<SelfHealBehaviour>();
        }

        public float GetExtraHealth()
        {
            return extraHealth;
        }
    }
}
```

SelfHealBehaviour

```
using UnityEngine;

namespace RPG.Character
{
    public class SelfHealBehaviour : AbilityBehaviour
    {
        PlayerControl player = null;

        void Start()
        {
            player = GetComponent<PlayerControl>();
        }
```

```
        public override void Use(GameObject target)
        {
            var playerHealth = player.GetComponent<HealthSystem>();
            playerHealth.Heal((config as SelfHealConfig).GetExtraHealth());
            PlayParticleEffect();
            PlayAbilitySound();
            PlayAbilityAnimation();
        }

    }
}
```