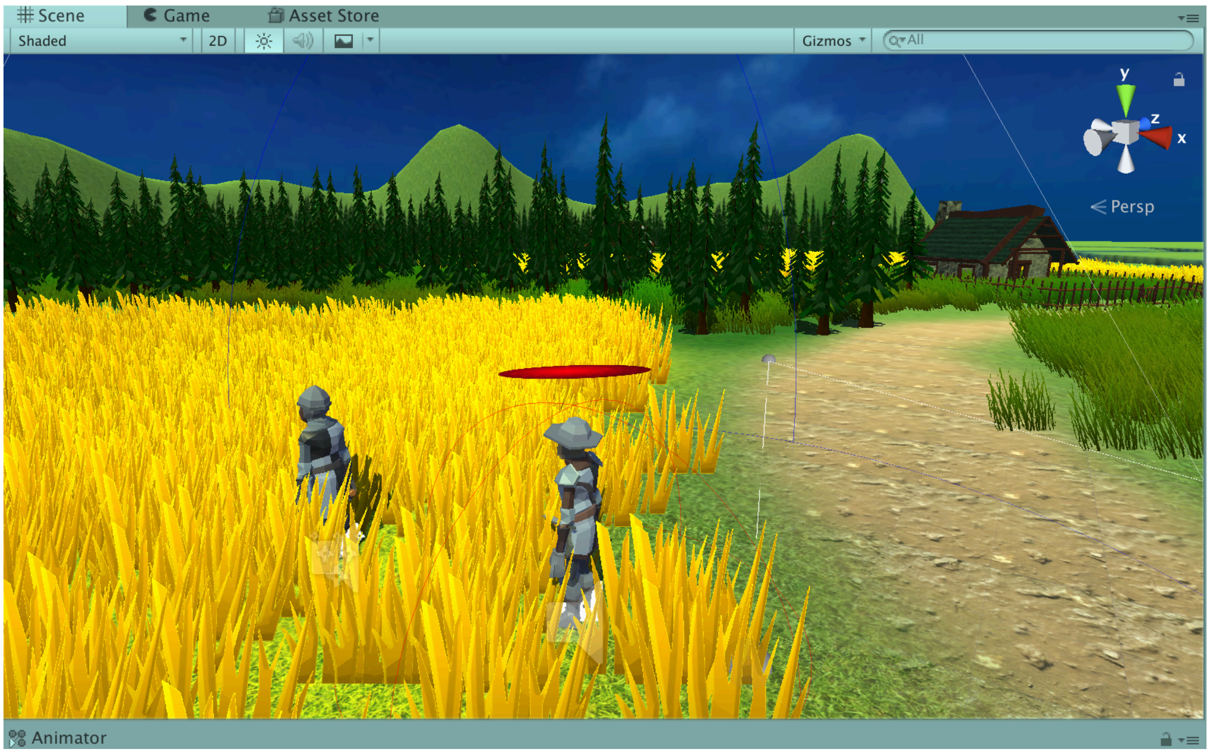
Assignment #2:
Question: How to design the functions of characters (include Player and Enemy)?
Key Words: Character, EnemyAI, PlayerControl



Player and Enemy could have some same functions, like SetDestination(), SetForwardAndTurn(), UpdateAnimator(), OnAnimatorMove(), ApplyExtraTurnRotation(). So, I create a general script named Character.cs to control them. Here is the code:

Character.cs

```
using UnityEngine;
using UnityEngine.AI;

namespace RPG.Character
{
    [SelectionBase]
    public class Character : MonoBehaviour
    {
        [Header("Audio")]
        [SerializeField] float audioSourceSpatialBlend = 0.5f;

        [Header("Animator")]
        [SerializeField] RuntimeAnimatorController animatorController;
        [SerializeField] AnimatorOverrideController animatorOverrideController;
        [SerializeField] Avatar charaterAvater;

        [Header("Capsule Collider Settings")]
        [SerializeField] Vector3 colliderCenter = new Vector3(0, 0.8f, 0);
        [SerializeField] float colliderRadius = 0.2f;
        [SerializeField] float colliderHeight = 2f;

        [Header("Movment Properties")]
        [SerializeField] float moveSpeedMultiplier = .7f;
        [SerializeField] float animationSpeedMultiplier = 1.2f;
        [SerializeField] float m_MovingTurnSpeed = 360;
        [SerializeField] float m_StationaryTurnSpeed = 180;
        [SerializeField] float moveThreshold = 1f;

        [Header("Nav Mesh Agent")]
        [SerializeField] float navMeshAgentSteeringSpeed = 5.0f;
        [SerializeField] float navMeshAgentStoppingDistance = 1.3f;
        [SerializeField] float navMeshAgentAcceleration = 120f;

        float turnAmount;
        float forwardAmount;
        NavMeshAgent navMeshAgent;
        Animator animator;
        Rigidbody myRididBody;
        bool isAlive = true;

        private void Awake()
        {
            AddRequiredComponents();
        }

        private void AddRequiredComponents()
        {
            var capsuleCollider = gameObject.AddComponent<CapsuleCollider>();
            capsuleCollider.center = colliderCenter;
            capsuleCollider.radius = colliderRadius;
            capsuleCollider.height = colliderHeight;

            myRididBody = gameObject.AddComponent<Rigidbody>();
            myRididBody.constraints = RigidbodyConstraints.FreezeRotation;

            var audioSource = gameObject.AddComponent<AudioSource>();
            audioSource.spatialBlend = audioSourceSpatialBlend;

            animator = gameObject.AddComponent<Animator>();
            animator.runtimeAnimatorController = animatorController;
            animator.avatar = charaterAvater;

            navMeshAgent = gameObject.AddComponent<NavMeshAgent>();
            navMeshAgent.speed = navMeshAgentSteeringSpeed;
            navMeshAgent.stoppingDistance = navMeshAgentStoppingDistance;
            navMeshAgent.acceleration = navMeshAgentAcceleration;
            navMeshAgent.autoBraking = false;
            navMeshAgent.updateRotation = false;
            navMeshAgent.updatePosition = true;
        }

        private void Update()
        {
            if (navMeshAgent.remainingDistance > navMeshAgent.stoppingDistance
&& isAlive)
            {
                Move(navMeshAgent.desiredVelocity);
            }
            else
            {
                Move(Vector3.zero);
            }
        }

        public float GetAnimSpeedMultiplier()
        {
            return animationSpeedMultiplier;
        }

        public void SetDestination(Vector3 worldPos)
        {
            navMeshAgent.destination = worldPos;
        }

        public AnimatorOverrideController GetOverrideController()
        {
            return animatorOverrideController;
        }

        void Move(Vector3 movement)
        {
            SetForwardAndTurn(movement);
            ApplyExtraTurnRotation();
            UpdateAnimator();
        }

        public void Kill()
        {
            isAlive = false;
        }

        void SetForwardAndTurn(Vector3 movement)
        {
            if (movement.magnitude > moveThreshold)
            {
                movement.Normalize();
            }
            var localMove = transform.InverseTransformDirection(movement);
            turnAmount = Mathf.Atan2(localMove.x, localMove.z);
            forwardAmount = localMove.z;
        }

        void UpdateAnimator()
        {
            // update the animator parameters
            animator.SetFloat("Forward", forwardAmount, 0.1f, Time.deltaTime);
            animator.SetFloat("Turn", turnAmount, 0.1f, Time.deltaTime);
            animator.speed = animationSpeedMultiplier;
        }

        void ApplyExtraTurnRotation()
        {
                float turnSpeed = Mathf.Lerp(m_StationaryTurnSpeed,
m_MovingTurnSpeed, forwardAmount);
            transform.Rotate(0, turnAmount * turnSpeed * Time.deltaTime, 0);
        }

        private void OnAnimatorMove()
        {
                if (Time.deltaTime > 0)
            {
                Vector3 velocity = (animator.deltaPosition *
moveSpeedMultiplier) / Time.deltaTime;

                // we preserve the existing y part of the current velocity.
                velocity.y = myRididBody.velocity.y;
                myRididBody.velocity = velocity;
            }
        }

    }
}
```

For Enemy, I should set four different states for enemy: Idel, Patrolling, attacking and chasing, and set different behaviors to them. I used IEnumerator to do things. Here is the code:
EnemyAI.cs

```
namespace RPG.Character
{
    [RequireComponent(typeof(HealthSystem))]
    [RequireComponent(typeof(Character))]
    [RequireComponent(typeof(WeaponSystem))]
    public class EnemyAI : MonoBehaviour
    {
        [SerializeField] float chaseRadius = 6f;
        [SerializeField] WaypointContainer patrolPath;
        [SerializeField] float waypointTolerance = 2f;
        [SerializeField] float waypointDwellTime = 0.5f;

        float currentWeaponRange;
        float distanceToPlayer;
        int nextWaypointIndex;
        PlayerControl player = null;
        Character character;

        enum State {
            idel,
            patrolling,
            attacking,
            chasing
        }
        State state = State.idel;

        private void Start()
        {
            player = FindObjectOfType<PlayerControl>();
            character = GetComponent<Character>();
        }

        private void Update()
        {
            distanceToPlayer = Vector3.Distance(player.transform.position,
transform.position);
            WeaponSystem weaponSystem = GetComponent<WeaponSystem>();
            currentWeaponRange =
weaponSystem.GetCurrentWeaponInUse().GetMaxAttackRange();
```

```
            bool inWeaponCircle = distanceToPlayer <= currentWeaponRange;
            bool inChaseCircle = distanceToPlayer > currentWeaponRange
                                 && distanceToPlayer <= chaseRadius;
            bool outsideChaseRing = distanceToPlayer > chaseRadius;

            if (outsideChaseRing && state != State.patrolling)
            {
                StopAllCoroutines();
                weaponSystem.StopAttacking();
                StartCoroutine(Patrol());
            }
            if (inChaseCircle && state != State.chasing)
            {
                StopAllCoroutines();
                weaponSystem.StopAttacking();
                StartCoroutine(ChasePlayer());
            }
            if(inWeaponCircle && state != State.attacking)
            {
                StopAllCoroutines();
                weaponSystem.AttackTarget(player.gameObject);
            }
        }


        IEnumerator Patrol()
        {
            state = State.patrolling;
            while(true)
            {
                Vector3 nextWaypointPos =
patrolPath.transform.GetChild(nextWaypointIndex).position;
                character.SetDestination(nextWaypointPos);
                CycleWaypointWhenClose(nextWaypointPos);
                yield return new WaitForSeconds(waypointDwellTime);
            }
        }

        private void CycleWaypointWhenClose(Vector3 nextWaypointPos)
        {
            if (Vector3.Distance(transform.position, nextWaypointPos) <=
waypointTolerance)
            {
                nextWaypointIndex = (nextWaypointIndex + 1) %
patrolPath.transform.childCount;
            }
        }

        IEnumerator ChasePlayer()
        {
            state = State.chasing;
            while (distanceToPlayer >= currentWeaponRange)
            {
                character.SetDestination(player.transform.position);
                yield return new WaitForEndOfFrame();
            }
        }
```

And for Player, the character should act based on the mouse click and what objects are, like RegisterForMouseEvents(), OnMouseOverWalkableLayer(), OnMouseOverEnemy(), ScanForAbilityKeyDown()….Here is the code:
PlayerControl.cs

```
using UnityEngine;
using System.Collections;

using RPG.CameraUI;

namespace RPG.Character
{
    public class PlayerControl : MonoBehaviour
    {
        Character character;
        SpecialAbilities abilities;
        WeaponSystem weaponSystem;

        void Start()
        {
            character = GetComponent<Character>();
            abilities = GetComponent<SpecialAbilities>();
            weaponSystem = GetComponent<WeaponSystem>();

            RegisterForMouseEvents();
        }

        void Update()
        {
            ScanForAbilityKeyDown();
        }

        private void RegisterForMouseEvents()
        {
            var cameraRaycaster = FindObjectOfType<CameraRaycaster>();
            cameraRaycaster.onMouseOverEnemyLayer += OnMouseOverEnemy;
            cameraRaycaster.onMouseOverWalkableLayer +=
OnMouseOverWalkableLayer;
        }

        private void ScanForAbilityKeyDown()
        {
            for (int keyIndex = 1; keyIndex < abilities.GetNumberOfAbilities();
keyIndex++)
            {
                if (Input.GetKeyDown(keyIndex.ToString()))
                {
                    abilities.AttemptSpecialAbility(keyIndex);
                }
            }
        }

        void OnMouseOverWalkableLayer(Vector3 destination)
        {
            if (Input.GetMouseButton(0))
            {
                weaponSystem.StopAttacking();
                character.SetDestination(destination);
            }
        }

        bool IsTargetInRange(GameObject target)
        {
            float distanceToTarget = (target.transform.position -
transform.position).magnitude;
            return distanceToTarget <=
weaponSystem.GetCurrentWeaponInUse().GetMaxAttackRange();
        }

        void OnMouseOverEnemy(EnemyAI enemy)
        {
            if (Input.GetMouseButton(0) && IsTargetInRange(enemy.gameObject))
            {
                weaponSystem.AttackTarget(enemy.gameObject);
            }
            else if (Input.GetMouseButton(0)
&& !IsTargetInRange(enemy.gameObject))
            {
                StartCoroutine(MoveAndAttack(enemy));
            }
            else if (Input.GetMouseButtonDown(1) &&
IsTargetInRange(enemy.gameObject))
            {
                abilities.AttemptSpecialAbility(0, enemy.gameObject);
            }
            else if (Input.GetMouseButtonDown(1)
&& !IsTargetInRange(enemy.gameObject))
            {
                StartCoroutine(MoveAndPowerAttack(enemy));
            }
        }

        IEnumerator MoveToTarget(GameObject target)
        {
            character.SetDestination(target.transform.position);
            while (!IsTargetInRange(target))
            {
                yield return new WaitForEndOfFrame();
            }
            yield return new WaitForEndOfFrame();
        }

        IEnumerator MoveAndAttack(EnemyAI enemy)
        {
            yield return StartCoroutine(MoveToTarget(enemy.gameObject));
            weaponSystem.AttackTarget(enemy.gameObject);
        }

        IEnumerator MoveAndPowerAttack(EnemyAI enemy)
        {
            yield return StartCoroutine(MoveToTarget(enemy.gameObject));
            abilities.AttemptSpecialAbility(0, enemy.gameObject);
        }
    }
}
```