**Introduction**

My scenario is a simple medieval-style battlefield with three buildings, auto-generating enemies, and player-controlled soldiers in a flat grassy field with woods. The player controls the character from a first-person perspective. The game officially begins when the player walks into the nearest building, picks up an object, walks through the stairs to the second floor, and throws the object down. The player's perspective turns to God's view in the sky, the enemy starts to generate wild monsters, and you can control the movement with the mouse by boxing the soldiers. You will need to control your soldiers to defend your castle from the monsters, and the game will fail when your castle is defeated. Defeating the monsters will give you gold, which can be used to buy soldiers. It would help if you bought many soldiers to attack the enemy castle to win.

**Project Plan**

First person control script

Control doors opening and closing and picking up objects

RTS control script

Bow and arrow soldier script

Enemy script

Life script

Complete animation control of enemies and soldiers

Create various UI

**Design**

A simple RTS game should have two camps with their main bases far apart, and both sides are fighting halfway to destroy the other's main base. Therefore, the scenery is mostly flat grasslands and trees used for decoration. The design also considered the ability of trees to be harvested as resources for the player or the enemy to build buildings and to hide enemies to surprise the careless player.

For the art style of the characters and buildings, I chose the medieval magic style. The medieval magic style is very mature; any current combat unit can find its counterpart in the medieval magic style; for example, fighters can correspond to flying dragons or some dwarven machines, and gunners can correspond to wizards; in short, it is a universal style that allows you to design all kinds of characters.

I first designed the most basic soldier for the player - the archer, which brings the player the advantage of long-range, but low blood and basically no chance to win one on one with the primary enemy units without moving. However, the player is flexible enough to control the crossbowman's movement when loading ammunition. In that case, he may be able to defeat the enemy alone.

Enemy basic units are orc infantry, melee units with thicker blood and higher attack power. They generate more frequently over time until they reach their maximum generation frequency. They always move towards the player's base and attack soldiers.

Both sides' bases are small medieval-style castles, and the player starts in medieval-style wooden houses. All units in the scene are uniform and coordinated. To ensure that the player can control the soldiers properly, the scene is set up with a white bar with a good view, using a mixture of linear light sources.

**Techinical Element**

I used diffuse reflective materials to add a medieval magical atmosphere to my scene. Diffuse materials are matte surfaces that absorb most of the light they receive - the darker they are, the more light they absorb. Thus medieval buildings with diffuse materials have an ancient look and feel.

**Script**

This is the first-person control script that enables character movement and view rotation.

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.SceneManagement;

public class first : MonoBehaviour

{

public Transform \_camera;

float x = 0f;

float y = 0f;

public float move\_speed = 7.5f;

public float rorate\_speed = 60f;

void move\_()

{

transform.Translate(Input.GetAxis("Horizontal") \* Time.deltaTime \* move\_speed, 0f, Input.GetAxis("Vertical") \* Time.deltaTime \* move\_speed);

}

void rotate\_()

{

x += Input.GetAxis("Mouse X") \* 50f \* Time.deltaTime;

y -= Input.GetAxis("Mouse Y") \* 50f \* Time.deltaTime;

if (y > 20f) y = 20f;

if (y < -20f) y = -20f;

if (x > 360f) x -= 360f;

if (x < -360f) x += 360f;

transform.rotation = Quaternion.Euler(y, x, 0.0f);

transform.rotation = Quaternion.Euler(0f, x, 0.0f);

\_camera.rotation = Quaternion.Euler(y, x, 0.0f);

}

private void LateUpdate()

{

move\_();

rotate\_();

}

}

This is the soldier's behavior script that controls the soldier's attack, movement, and animation switching.

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.AI;

public enum State

{

idle,

move,

attack

}

public class base\_ : MonoBehaviour

{

#region start

public NavMeshAgent agent;

public Animator animator;

AudioSource audio;

private void Start()

{

agent = GetComponent<NavMeshAgent>();

agent.isStopped = false;

outline\_ = GetComponent<Outline>();

animator = GetComponent<Animator>();

audio = GetComponent<AudioSource>();

//character test

hp = 100f;

}

#endregion

#region state

[SerializeField]

public State state;

public State State

{

get { return state; }

set

{

state = value;

switch(state)

{

case State.idle:

animator.CrossFade("crossbow\_01\_idle", 0.2f);

break;

case State.move:

Debug.Log("switch move");

animator.CrossFade("crossbow\_03\_run", 0.2f);

break;

case State.attack:

animator.CrossFade("crossbow\_04\_attack\_A", 0.2f);

break;

}

}

}

public void SetIdle()

{

if(State != State.idle )State = State.idle;

}

#endregion

#region state\_on\_update

void state\_on\_update()

{

switch(state)

{

case State.idle:

break;

case State.move:

if(Vector3.Distance(transform.position,position)<1.5f)

{

State = State.idle;

}

break;

case State.attack:

break;

}

}

#endregion

#region action

Vector3 position;

public void move(Vector3 pos)

{

position = pos;

State = State.move;

agent.isStopped = false;

agent.SetDestination(pos);

}

public GameObject arrow;

GameObject arrow\_temp;

public Transform arrow\_pos;

Vector3 arrow\_target;

public void attack\_arrow(Vector3 pos)

{

State = State.attack;

transform.LookAt(pos);

arrow\_target = pos;

}

public void shoot\_arrow()

{

if (arrow\_temp == null) return;

arrow\_temp.SetActive(true);

arrow\_temp.transform.position = arrow\_pos.position;

arrow\_temp.GetComponent<ArrowLine>().shoot\_(arrow\_target);

audio.Play();

}

public void reload()

{

arrow\_temp = Instantiate(arrow, arrow\_pos.position, Quaternion.identity);

arrow\_temp.SetActive(false);

}

#endregion

#region select

public bool is\_select;

Outline outline\_;

public void switch\_select()

{

if (is\_select)

{

is\_select = false;

outline\_.enabled = false;

}

else

{

is\_select = true;

outline\_.enabled = true;

}

}

#endregion

#region attribute

[SerializeField] private float hp = 100f;

public void set\_hp(float t)

{

hp -= t;

if(hp < 0f || hp == 0f)

{

death();

}

}

void death()

{

animator.CrossFade("death", 0.2f);

Invoke("clear\_self", 3f);

}

void clear\_self()

{

Destroy(this.gameObject);

}

#endregion

private void Update()

{

state\_on\_update();

}

}

**Unique Element**

Figure 1, below, shows the soldier's animation controller using five animations, including standby, move, attack, wounded, and die. Switching between animations is controlled using scripts.

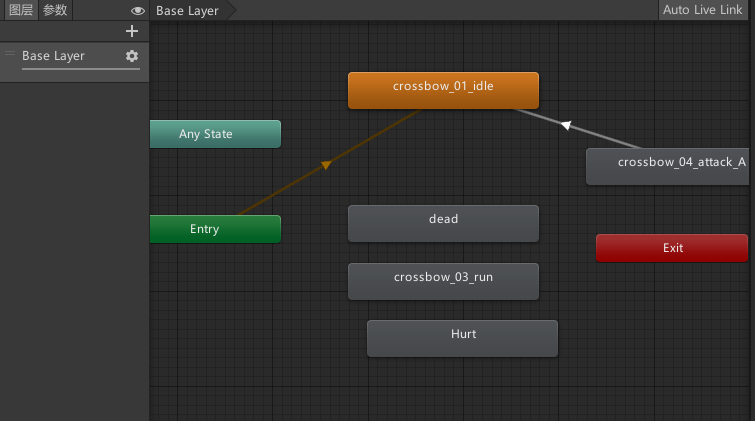


Figure 1: Animation Controller

Some animations will also call functions, shown below, Figure 2 and Figure 3, are frames of the attack animation that will call the reload function and the shoot\_arrow function. These two functions implement the reload and shoot\_arrow functions and coordinate with the action of the animation.

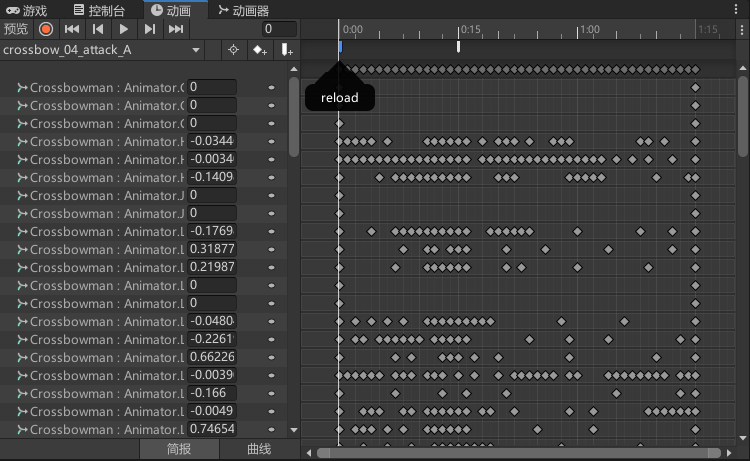


Figure 2: Attack animation function 1

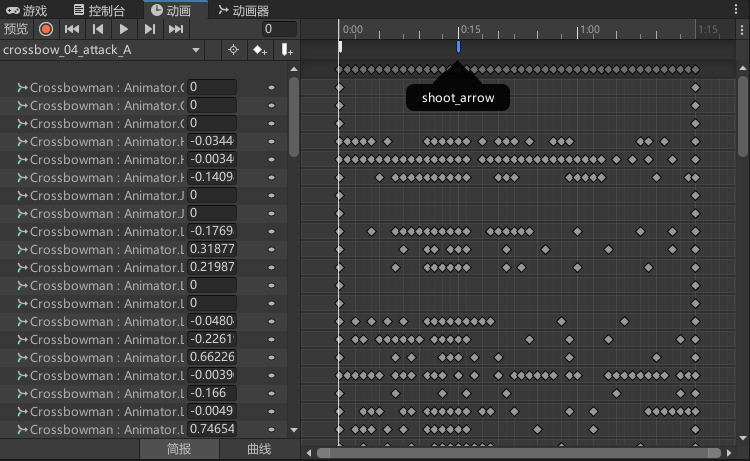


Figure 3: Attack animation function 2

**Summary**

My scenario shows all the required essential elements. I am using a linear light as the primary light source for the player to control the soldiers through a good view. As the video shows, an amazing medieval magical style scene and each character, the whole scene looks harmonious in style, full of fantasy elements and a sense of historical atmosphere. The game also has a basic sound system, the sound made by the soldiers when shooting arrows and the game's background music. Sound is a complex system that needs further improvement. To make the player pay attention to the enemy castle, which is the game's ultimate goal, I added a particle system to make the enemy castle emit a prominent particle effect over it. Scripting is the most complicated part of the game, mainly implementing the three aspects of player control, soldier behavior and monster behavior, building life values, etc. In short, scripting is the most time-consuming part of the game. On the basis of these, I made this simple RTS game, which implements the soldier's box selection and control, as well as a simple life value system for each game unit. Both character units appearing in the game have a complete set of animation systems and behavior logic implemented. This game has some fun, although it is fundamental.

**Future work**

As an RTS game, this game has too few characters and buildings. More buildings and characters should be added in the future, for example, characters and buildings for collecting resources, characters that can fly in the air, and defensive buildings that can attack. Of course, these characters and buildings should be in a uniform style, that is, medieval magic style; there should not be modern-style planes, tanks, and others.

The current gold system needs to be simplified. In the future, new wilderness resources should be added, which require players to fight with the enemy. This will increase the complexity of the game, as well as the intensity of the battles.

**Appendix**

Enemy.cs :

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.AI;

public class Enemy : MonoBehaviour

{

NavMeshAgent nav;

Animator ani;

private void Start()

{

nav = GetComponent<NavMeshAgent>();

attackTarget = Manager.ins.castle;

ani = GetComponent<Animator>();

}

public float attackGap = 2f;

float tempAttackGap = 0f;

bool canAttack = true;

public Life attackTarget;

public Life tempTarget;

private void Update()

{

if(tempAttackGap > attackGap && !canAttack)

{

canAttack = true;

tempAttackGap = 0f;

}

else

{

tempAttackGap += Time.deltaTime;

}

if(isAttackCastle)

{

if(canAttack)

{

attack(attackTarget);

return;

}

}

if(tempTarget==null)

{

nav.SetDestination(attackTarget.transform.position);

if (Vector3.Distance(transform.position, attackTarget.transform.position)<7f)

{

if(canAttack)

{

attack(attackTarget);

canAttack = false;

}

}

}

else

{

nav.SetDestination(tempTarget.transform.position);

print(Vector3.Distance(transform.position, tempTarget.transform.position));

if (Vector3.Distance(transform.position, tempTarget.transform.position) < 1.5f)

{

if (canAttack)

{

attack(tempTarget);

canAttack = false;

}

}

}

}

Life t;

void attack(Life life)

{

transform.LookAt(life.transform);

ani.CrossFade("attack", 0.2f);

canAttack = false;

t = life;

}

public void EnemyHurt()

{

t.Hurt(50f);

}

bool isAttackCastle = false;

private void OnCollisionEnter(Collision collision)

{

if(collision.gameObject.tag=="Castle")

{

//isAttackCastle = true;

//attack(attackTarget);

}

}

}

**References**

Below are all the references used in my application and report.

**GitHub**

GitHub homepage：

Application directory：

**Asset References**

* Toony Tiny RTS Set
* Stylized Nature Bundle

**Script References**

* Quick Outline