ГБОУ "Президентский ФМЛ № 239"

Компьютерная 2D игра

Годовой проект по информатике

Лобанов Александр, 10-3 класс

Постановка задачи

Целью моего проекта было создание небольшой 2D игры жанра «платформер». Главный герой, управление которым производится с клавиатуры, должен уметь передвигаться вправо и влево, карабкаться по стенам, прыгать, наносить урон врагам.

Различные враги должны наносить урон персонажу, мешая ему продвигаться по игровой карте. В игре должны быть препятствия или объекты, наносящие урон персонажу и не подверженные разрушению.

В игре должен присутствовать несложный интерфейс: главное меню, пауза, выбор из нескольких уровней.

Графика игры должна быть приятной, все двигающиеся объекты должны быть анимированы.

Уточнение исходных и выходных данных

Исходными данными в процессе игры являются нажатия на различные клавиши клавиатуры. Результатом является совершение управляемым персонажем определённых действий, например, бег, прыжок, атака. Управление интерфейсом происходит с использованием компьютерной мыши посредством нажатия кнопок на экране. Влиять на действие игры будут только определённые клавиши, нажатие же остальных никак не отразится на происходящем на экране.

Математическая модель

Все объекты – физические тела с конкретными границами. На всё поле игры действует гравитация. Некоторые объекты подвержены её влиянию (главный персонаж, движущийся монстр), некоторые – нет (блоки земли, стены, сюрикены, выпускаемые персонажем). Все объекты имеют свои координаты, движение происходит вдоль двух координатных осей (x, y). Координату z, отличную от нуля, имеет лишь объект «камера». Всё происходящее на сцене регулируется скриптом.

Выбор метода решения

Среда разработки – Unity, редактор кода - VisualStudio, язык скрипта – C#.

Весь скрипт написан на основе ООП. Каждому действующему объекту принадлежит свой конкретный скрипт, так или иначе взаимодействующий со скриптами других объектов. То есть весь код представляет из себя набор классов, названия которых соответствуют объектам, к которым эти классы привязаны. (Пример – класс Character отвечает за действия главного персонажа, класс MoveableMonster – за действие подвижного монстра и тд).

Непосредственно в Unity была реализована грамотная сортировка всех объектов, что положительно отразилось на удобстве работы с игрой на стадии разработки. Имеются экземпляры всех готовых объектов, что донельзя облегчает создание различных уровней.

Комментированный листинг

Разберём основные функции кода на примере самого разветвлённого и объёмного скрипта – скрипта главного персонажа (“Character”).

Все действия персонажа реализованы как отдельные методы, например – метод Run() отвечает за бег, метод Jump() – за прыжок, метод Shoot() – за метание сюрикенов.

```
ссылка: 1
private void Run() //метод, в котором реализован бег
{
    Vector3 direction = transform.right * Input.GetAxis("Horizontal");
    transform.position = Vector3.MoveTowards(transform.position, transform.position + direction, speed * Time.deltaTime);//непосредственно движение
    sprite.flipX = direction.x < 0.0F; //поворот персонажа (смотрит по направлению движения)
   if (isGrounded&&!isDying) State = CharState.Run;//CharState - переменная, работающая с анимацией персонажа. Здесь - анимация "бег".

}
ссылка: 1
private void Jump()//метод, в котором реализован прыжок
{
    rigidbody.AddForce(transform.up*jumpForce,ForceMode2D.Impulse);

}
ссылка: 1
private void Shoot()//метод, в котором реализовано метание сюрикенов
{
    if (Bullets > 0)
    {
        Vector3 position = transform.position; position.y += 0.2F;
        Bullet newBullet = Instantiate(bullet, position, bullet.transform.rotation) as Bullet; //появление сюрикена посредине персонажа
        newBullet.Direction = newBullet.transform.right * (sprite.flipX ? -1.0F : 1.0F);//движение сюрикена по направлению движения персонажа
        newBullet.Parent = gameObject;
        Bullets--;
    }

}
```

Отправной точкой в специализированные методы является метод Update, вызывающийся на каждом кадре. В нём в зависимости от ввода с клавиатуры вызываются описанные выше специализированные методы.

```
Сообщение Unity | Ссылок: 0
private void Update()//отправная точка в специализированные методы, вызывается каждый кадр.
{
    if (isGrounded&&!isDying)  State = CharState.Idle;
    if (Input.GetButton("Horizontal")) Run();
    if (isGrounded && Input.GetButtonDown("Jump")) Jump();
    if (Input.GetButtonDown("Fire1")) Shoot();


}
```

Ещё один специализированный метод – процедура получения урона.

```
Ссылок: 5
public override void ReceiveDamage()// процедура получения урона
{

    Lives--;
    rigidbody.velocity = Vector3.zero;
    rigidbody.AddForce(transform.up * 5.0F,ForceMode2D.Impulse); //небольшое откидывание персонажа при получении урона
    Debug.Log(lives);
    if (Lives == 0)
    {

        Die(); //вызов метода, описанного в классе прародителе "Unit"
    }

}
```

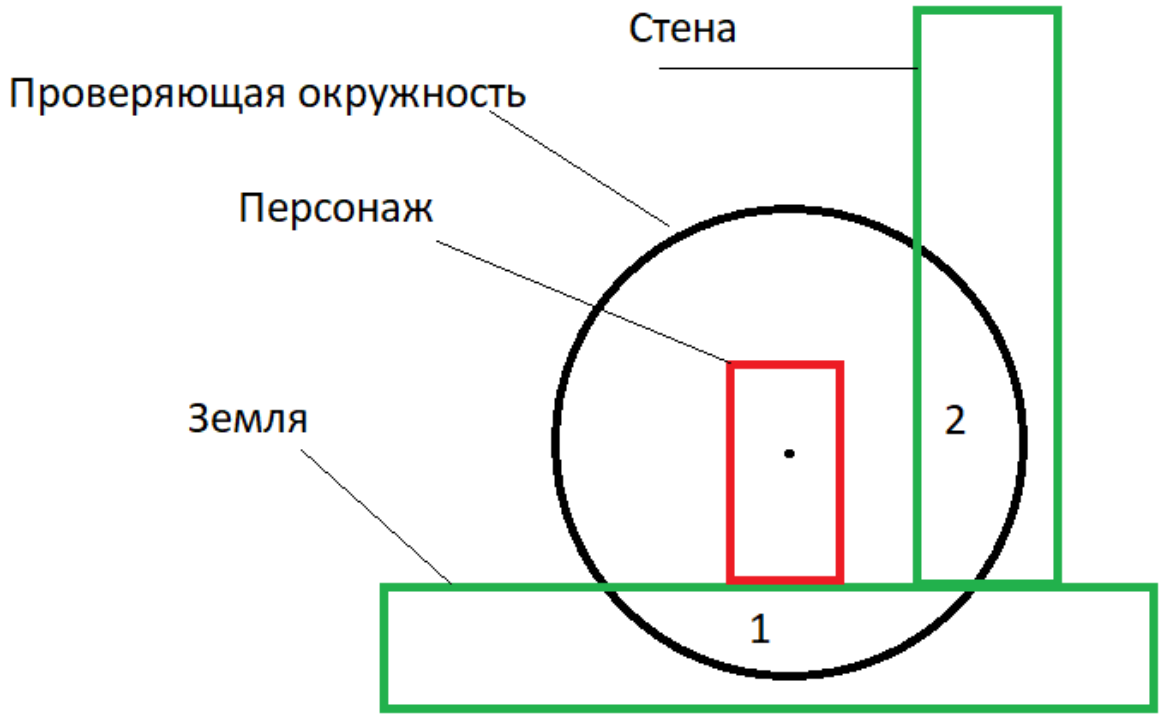
Отправной точкой в этот метод является метод OnTriggerEnter2D(Collider2D collider).

```
private void OnTriggerEnter2D(Collider2D collider)//метод, отвечающий за вызов метода ReceiveDamage(),
                                                  //т.е.за начало процедуры получения урона
{

  Unit unit = collider.gameObject.GetComponent <Unit>();//если объектом, с которым соприкоснулся персонаж, является
    if (unit) { ReceiveDamage(); }                      //другой живой объект (т.е. монстр), нужно получить урон
}
```

Немаловажным методом является метод, в котором проверяется, находится ли персонаж в открытом воздухе или стоит на земле, у стены. Принцип его работы считаю нужным разобрать подробнее.

```
private void CheckGround() //метод, проверяющий на земле (у стены) ли персонаж
{
    Collider2D[] colliders = Physics2D.OverlapCircleAll(transform.position, 0.1F); //проверка коллайдеров в окружности опр. радиуса
    isGrounded = colliders.Length > 1; //если таких коллайдеров больше одного, значит персонаж стоит на земле (у стены).
    if(!isGrounded) State = CharState.Jump;//если персонаж в водухе, анимация - прыжок
}
```



Окружность вокруг персонажа проверяет, сколько коллайдеров (физических границ объектов) внутри её. Очевидно, если персонаж в воздухе, внутри окружности будет только один коллайдер – коллайдер самого персонажа. В любом другом случае будет ещё как минимум один коллайдер – земли или стены. Таким образом можно избежать двойного прыжка, ведь метод Jump() попросту не будет доступен, если персонаж уже в воздухе.

Скрипты остальных подвижных объектов не содержат принципиально иных методов. Основная механика игры сполна представлена в скрипте главного персонажа.

Пример работы программы.

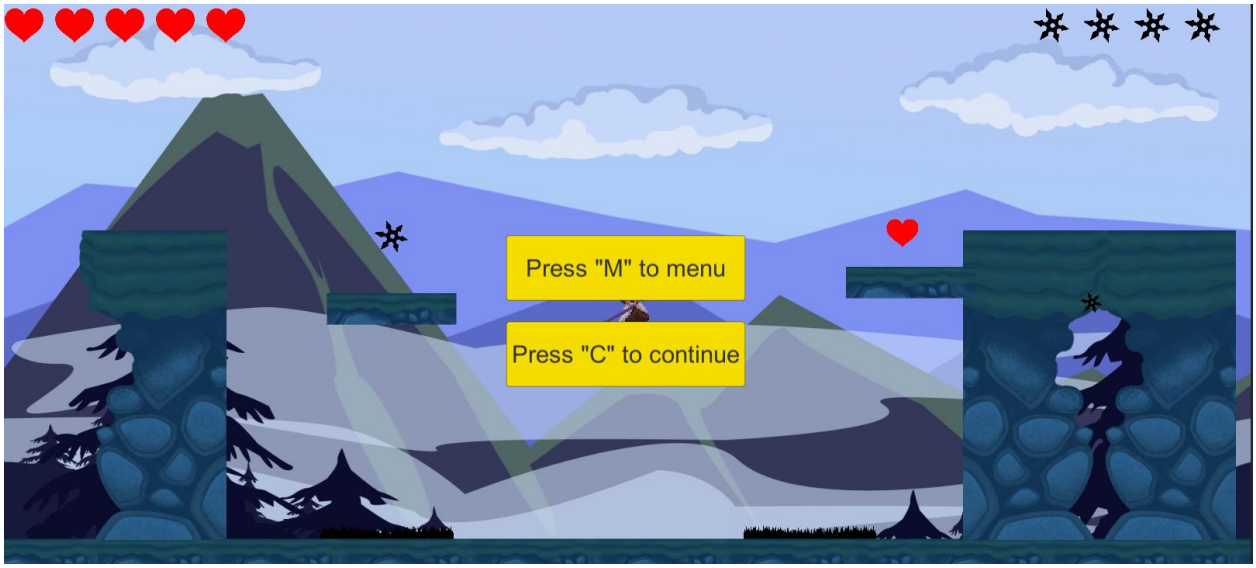
Основная игровая сцена. В верхнем левом углу – оставшиеся жизни. В верхнем правом – сюрикены. Два чёрных объекта под персонажем – шипы, наносящие урон.

Пример подвижного монстра. Скелет ходит между двумя объектами, находящимися на произвольном расстоянии друг от друга. Наносит урон персонажу.

Пример неподвижных монстров. Вырастают из-под земли. Наносят урон персонажу. Так же, как и подвижный монстр, умирают от попадания сюрикеном.

Здоровье и количество сюрикенов можно восполнить с помощью специальных предметов, разбросанных по карте.

Поставленная на паузу игру. Причём работают как кнопки на клавиатуре, так и копки непосредственно на экране (нужно нажать мышкой).



Анализ правильности решения.

Игра работает исправно. Баги и ошибки происходят крайне редко. На мой субъективный взгляд выглядит игра приятно, цветовая гамма и задний фон весьма спокойные. Все действующие объекты анимированы. Персонаж без промедления откликается на подаваемые с клавиатуры команды, интуитивно понятное управление происходит быстро и чётко. Обновляющиеся в реальном времени панели жизни и запаса сюрикенов работают без нареканий. Точки пополнения запаса работают исправно, при этом они не увеличивают число жизней или сюрикенов, если запас полный.

Таким образом поставленная задача выполнена.