Государственное бюджетное общеобразовательное учреждение
"Президентский физико-математический лицей № 239"

2D игра “PiG”

Годовой проект по информатике

Поляков Михаил 10-3 класс

Постановка задачи:

С помощью игрового движка “UNITY” нужно было разработать 2D игру, в которой главной целью являлось победа над всеми врагами. Основная механика игры - прыжки между платформ, поэтому главный персонаж должен уметь:

-Бегать
-Прыгать
-Бить или стрелять для уничтожения врагов

В целом игра должна выглядеть довольно красиво, соблюдая некоторые человеческие цветовые ассоциации(например, главный враг должен быть мрачным, а главный герой, олицетворяя добро, должен быть светлым).

В игре должно несколько главное меню и несколько уровней, каждый из которых должен иметь особенность(первый уровень - ознакомительный, далее - сложнее)

Враги в игре не должны быть однотипными, они должны различаться по типу движения, по типу атаки итд. Помимо врагов в игре должны быть препятствия, которые также будут наносить урон игроку.

Необходимо сделать игру хоть немного интересной, чтобы пройти ее можно было не сразу, а только немного потренировавшись и изучив уровень.

Уточнение исходных и выходных данных:

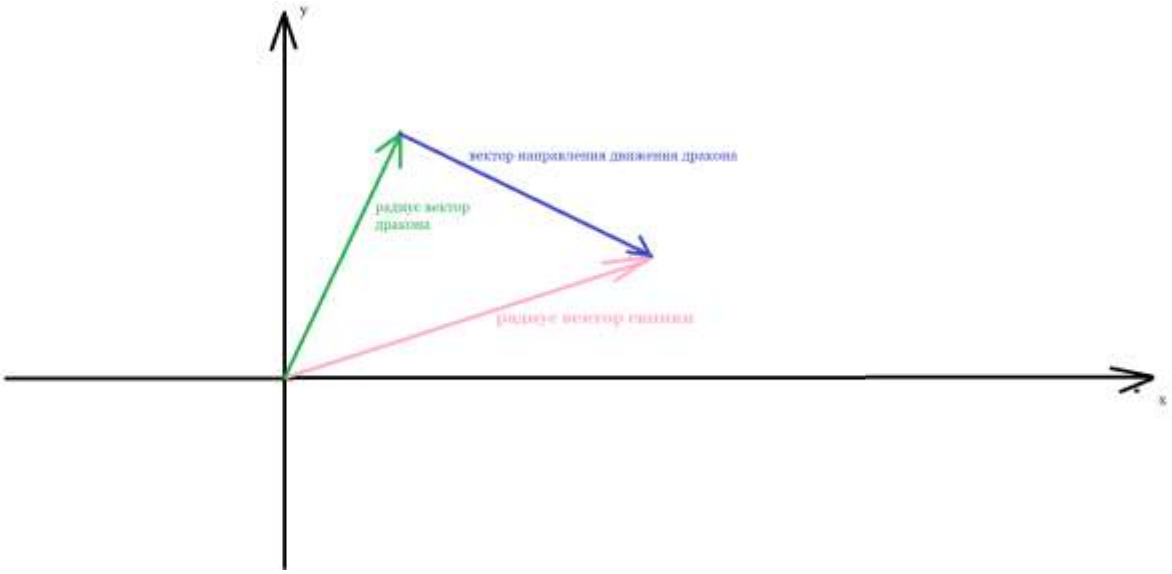
Исходными данными для игры по сути является расстановка разных видов платформ и врагов. Во время ее запуска, редактировать сцену как вам хочется - нельзя, но зато в редакторе, то есть в самом Unity, создание нового уровня, или редактирование уже имеющегося не составляет труда, если имеются некоторые заготовки (враги, платформы, камера, сам игрок). Собственно создание этих заготовок и является разработкой игры.

Когда игра запущена, и вы играете входными данными являются лишь ваши нажатия на клавиатуру(например, для выстрела нужно нажать на левую кнопку мыши)

Выходными же данными является процесс игры.

Математическая модель:

Каждый объект на сцене(за исключением платформ и особых объектов) является физическим телом, у которого есть масса и форма. Местоположение любого объекта характеризуется радиус вектором. На все физические объекты сцены действует сила тяжести, обусловленная ускорением свободного падения, которое для всех объектов одинаково. Скорость движущихся объектов задается вручную.



Дракончик - летающий враг, который преследует игрока. Направление его движения определяется разность радиус векторов игрока и самого дракона.

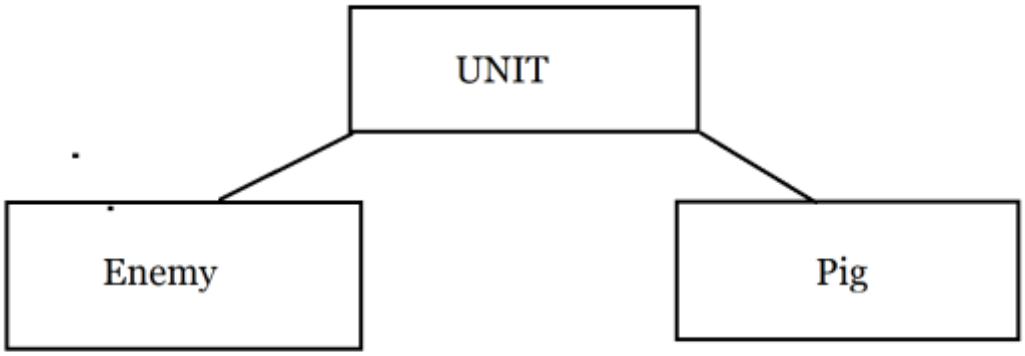
Анализ используемой структуры данных:

Так как четких входных и выходных данных нет, на данный вопрос ответить проблематично.

Выбор метода решения:

1) В основе проекта лежит ООП. Скрипт для любой “живого” объекта наследуется от общего класса Unit. Unit подразделяется на класс врагов

и класс самого игрока.



2) Далее класс Enemy подразделяется на классы разных врагов: летающий, патрулирующий, стоящий на месте и стреляющий.
3) В каждом классе нужно написать алгоритм, необходимый для выполнения задумки: алгоритмы прыжка, бега, стрельбы итд

Комментированный листинг:

Игрок:

```
  private void Update()
      //вызывается каждый кадр
      //проверяет нажата ли кнопка прыжка, взаивисимости от чего и проиходит прыжок
{

      if (Input.GetKeyDown(KeyCode.W) && (onGround || (jumps > 0)))
      {
          Jump();
          jumps--;
      }

      //часть кода отвечает за отображение сердечек
      if (hp > numOfHearts) hp = numOfHearts;

      for (int i = 0; i < hearts.Length; i++)
      {
          if (i < numOfHearts) { hearts[i].enabled = true; }
          else { hearts[i].enabled = false; }

          if (i < hp) { hearts[i].sprite = fullHeart; }
          else { hearts[i].sprite = emptyHeart; }
      }
  }
new void FixedUpdate()
    //вызывается через фиксированный очень маленький промежуток времени
    //проверяется нажатие кнопок, сколько жизней у игрока и наличие земли под ногами игрока
    //результатом является движение, смерть или возможность прыжка
{
    base.FixedUpdate();

    if (Input.GetButton("Horizontal")) Run();
    else animator.SetFloat("Speed", 0);

    if ((hp < 1)&&(Alive))
    {
        animator.SetTrigger("Dead");
        Alive = false;
    }

    Check(); //отвечает за проверку земли под ногами игрока
    if (onGround) jumps = 1;
}
```

```
private void Run() //отвечает за бег
{
    animator.SetFloat("Speed", 1);

    Vector3 direction = Right * Input.GetAxis("Horizontal");

    transform.position = Vector3.MoveTowards(transform.position, transform.position + direction, speedX * Time.deltaTime);
    if ((direction.x * transform.right.x < 0f)) transform.Rotate(0f, 180f, 0f);
    //если направление движения не совпадает с направлением игрока, то нужно повернуть игрока

}
```

```
ссылок: 1
private void Jump()//прыжок
{
    rb.AddForce(transform.up * verticalImpulse, ForceMode2D.Impulse);
}
```

```
ссылок: 1
private void Check()
{
    Collider2D[] colliders = Physics2D.OverlapCircleAll(transform.position, rad);
    onGround = colliders.Length > 1;
    //если количество коллайдеров около ног игрока больше 1, то можно прыгать
    animator.SetBool("IsJumping", !onGround);
}
```

```
public override void Damage()
    //когда игрока бьют, он немного отлетает, а не просто стоит на месте
{
    base.Damage();

    rb.velocity = Vector3.zero;
    rb.AddForce(transform.up * 50.0F, ForceMode2D.Impulse);
    rb.AddForce(transform.right * -Input.GetAxis("Horizontal") * 10.0F, ForceMode2D.Impulse);
}
```

Босс:

```
void Patrol()//патрулирование босса
{
    transform.position = Vector3.MoveTowards(transform.position, transform.position + transform.right, speed * Time.deltaTime);
    turn = !Physics2D.OverlapCircle(groundcheck.position, 0.3f, Ground)||bossCollider.IsTouchingLayers(Box);
    if (turn) transform.Rotate(0f, 180f, 0f);

    animator.SetBool("IsRunning", true);
}
```

```
ссылок: 1
void Shoot()//стрельба босса
{
    animator.SetBool("IsRunning", false);

    Instantiate(Rasengun, firepoint1.position, firepoint1.rotation);
    Instantiate(Rasengun, firepoint2.position, firepoint2.rotation);
}
```

Класс “живых” существ Unit:

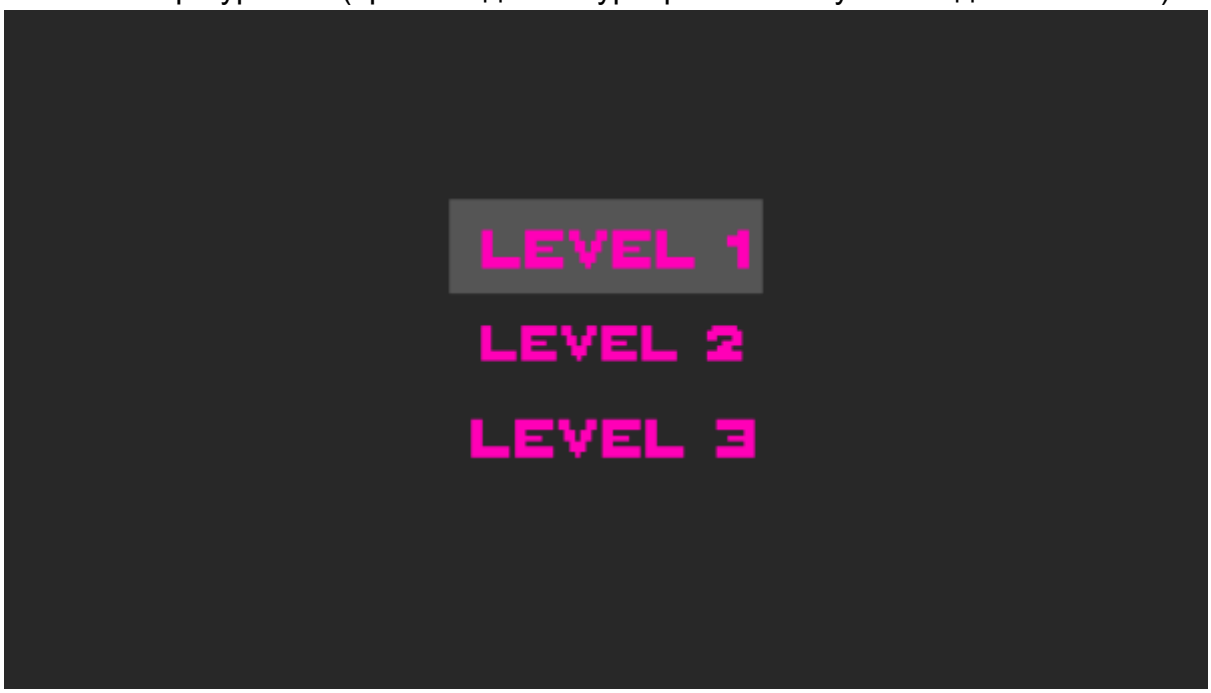
```
protected void FixedUpdate()
    //любой враг умирает если у него нет жизней или он находится слишком низко(высоко)
{
    if (hp < 1)
    {
        Die();
    }

    time--;
    if (time <= 0) gameObject.GetComponentInChildren<Renderer>().material.color = Color.white;

    if (Mathf.Abs(transform.position.y) > 100) hp = 0;
}

public virtual void Die()
{
    Destroy(gameObject, 0.4F);
}
Ссылок: 5
public virtual void Damage()
    //при нанесении урона объекту у него уменьшется количество жизней и он краснеет
{
    hp--;

    time = 10f;
    if ((time >= 0) && (hp > 0)) gameObject.GetComponentInChildren<Renderer>().material.color = Color.red;
}
Ссылок: 2
public void Heal()
//функция восстанавливет количество жизней и объект зеленеет
{
    hp = maxHp;
    time = 20f;
    if ((time >= 0) && (hp > 0)) gameObject.GetComponentInChildren<Renderer>().material.color = Color.green;
}
```

Остальной код с комментариями можно посмотреть на github.

Пример работы программы:

Начальное меню:

Меню выбора уровня: (при наведении курсора на кнопку она подсвечивается)



Первый уровень:

Меню, появляющееся при проигрыше:

Меню победы:

Второй уровень:

Третий уровень(босс):

Анализ правильности решения:

На мой взгляд игра получилась довольно неплохой, хотя и не имеет революционных механик. Одной из целей являлось сделать игру сносно выглядящей, поэтому много времени ушло на поиск картинок, их самостоятельное рисование.

Дав поиграть своим друзьям в получившуюся игру, оказалось что пройти ее с первого раза очень тяжело, а победить босса у многих и вовсе не получилось, ведь для победы над ним недостаточно просто много его бить. Поэтому на мой взгляд механика босса получилась удачно.

Тем не менее в игре удалось реализовать несколько видов врагов:

1) Статичный
2) Патрулирующий
3) Летающий, преследующий игрока
4) Стреляющий
5) Босс(является солянкой из стреляющего и патрулирующего)

Помимо врагов игроку может навредить препятствие в виде кактуса, так же имеются так называемые “хилки”, подобрав которые игрок может восстановить жизни.

В игре имеется почти полноценное меню, а в конце каждого уровня, в зависимости от успеха прохождения появляется переходное меню.

Удалось реализовать все анимации(бега, прыжка, полета, смерти, уменьшения/восстановления жизней). На каждом уровне в реальном времени отображается количество жизней игрока.

Реализована самая основная задача - управление персонажем работает корректно.

Должен добавить, что хоть проект и выглядит, как что-то завершенное, таковым он не является. С самого начала планы были грандиозные, и по ходу работы над проектом возникало очень много идей, которые реализовать не получилось за неимением некоторых знаний.