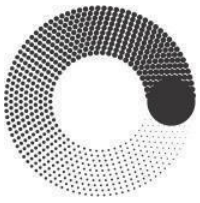
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ



МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
ВЫСШАЯ ШКОЛА ПЕЧАТИ И МЕДИАИНДУСТРИИ

Институт Принтмедиа и информационных технологий
Кафедра Информатики и информационных технологий

направление подготовки
09.03.02 «Информационные системы и технологии»

ЛАБОРАТОРНАЯ РАБОТА № 1

Дисциплина:

_____

Тема:

_____

Выполнил(а): студент(ка) группы _____
Ву Хоанг Нам Куан 211-728

_____
(Фамилия И.О.)

Дата, подпись _____*Quân*_ ____
(Дата) (Подпись)

Проверил _____
(Фамилия И.О., степень, звание) **(Оценка)**

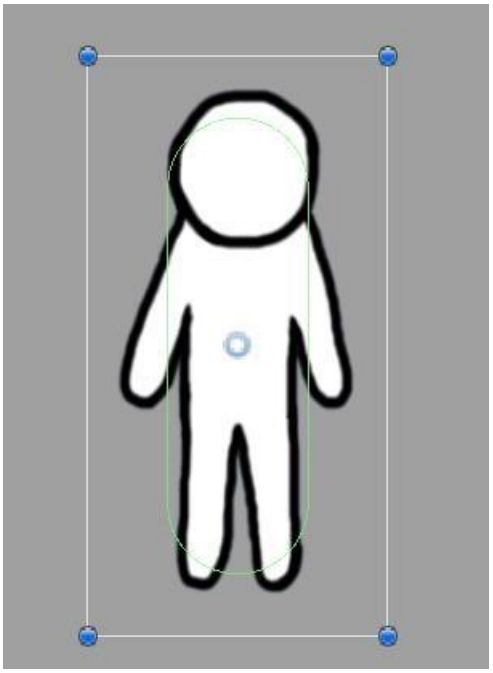
Дата, подпись _____
(Дата) (Подпись)

Замечания: _____

_____

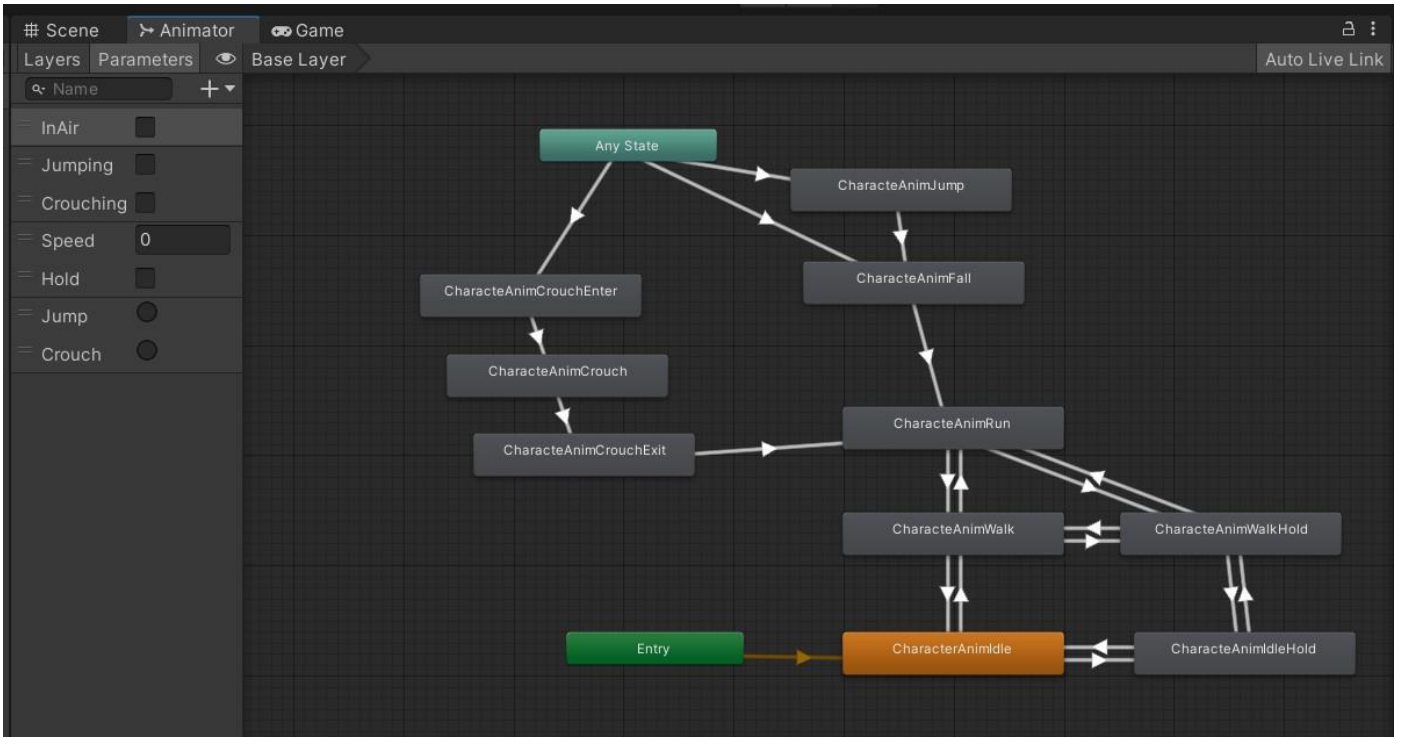
_____

I.Главный герой:



Основные персонажи включают: Sprite Renderer, Animator, Rigidbody2D, Capsule Collider 2D, Player Control Script.

Animator:



«Аниматор» для управления состояниями персонажа, используйте параметры для переключения между состояниями, различные анимации. Бегать, прыгать, сидеть, стоять на месте и т.д....

Script:
Скрипт управления персонажем содержит более 500 строк, поэтому я не могу скопировать и вставить его сюда.

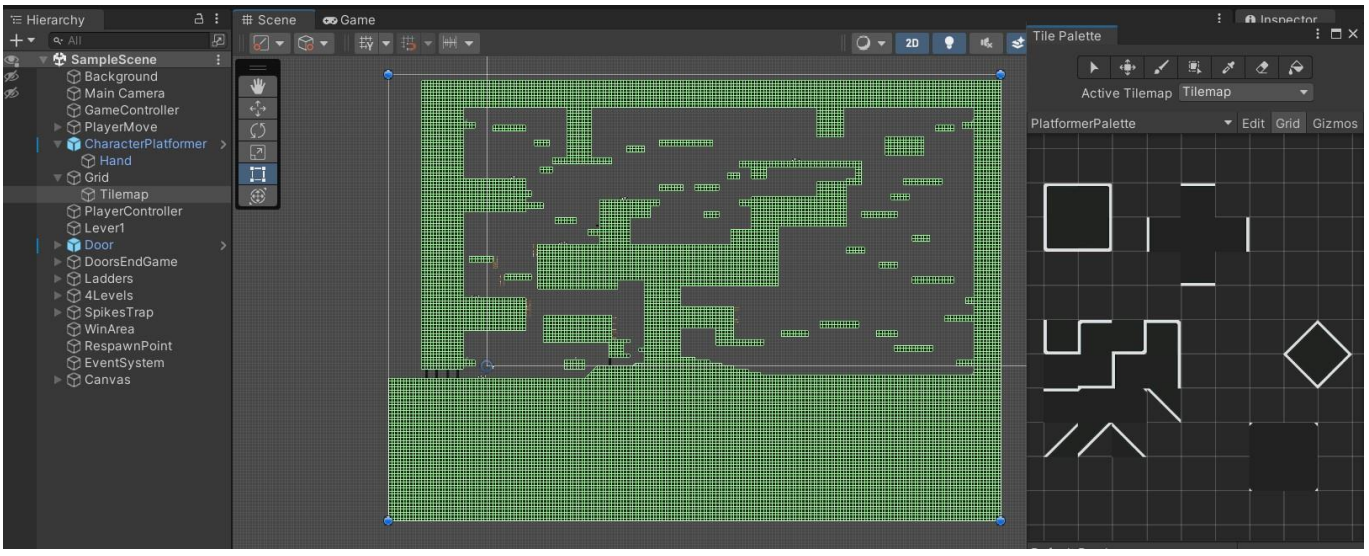
Сценарий имеет очень быстрый итеративный цикл, именно благодаря которому игра работает, этот цикл будет проверять состояние персонажа, получать вводы с клавиатуры, мыши и... и выполнять функции доступного кода. Я добавил несколько вещей, таких как: HP, скорость бега, ускорение бега, ускорение остановки, сила прыжка, двойной прыжок, гравитация, направления движения с помощью клавиш на клавиатуре, ...

```
//Handle physics
void FixedUpdate()
{
    if (si_dead)
        return;

    if (isClimbing)
    {
        rigid.gravityScale = 0f;
        rigid.velocity = new Vector2(rigid.velocity.x, vertical * speed);
        return;
    }
    else
    {
        rigid.gravityScale = normalGravity;
    }

    //Movement velocity
    float desiredSpeed = Mathf.Abs(move_input.x) > 0.1f o move_input.x * move_max : 0f;
    float acceleration = Mathf.Abs(move_input.x) > 0.1f o move_accel : move_deccel;
    acceleration = !is_grounded o jump_move_percent * acceleration : acceleration;
    move.x = Mathf.MoveTowards(move.x, desiredSpeed, acceleration * Time.fixedDeltaTime);

    UpdateFacing();
    UpdateJump();
    UpdateCrouch();

    //Move
    rigid.velocity = move;
}

//Handle render and controls
void Update()
{
    if (si_dead)
```

```
            return;

        hit_timer += Time.deltaTime;
        grounded_timer += Time.deltaTime;

        vertical = Input.GetAxisRaw("Vertical");

        if (isLadder && Mathf.Abs(vertical) > 0f)
        {
            isClimbing = true;
        }

        if (isInsideSpikes)
        {
            TakeDamage(10);
        }

        //Controls
        fllayerControls controls = fllayerControls.Get(player_id);
        move_input = !disable_controls o controls.GetMove() : Vector2.zero;
        jump_press = !disable_controls o controls.GetJumpDown() : false;
        jump_hold = !disable_controls o controls.GetJumpHold() : false;

        if (jump_press || move_input.y > 0.5f)
            Jump();

        //Reset when fall
        if (transform.position.y < fall_pos_y - GetSize().y)
        {
            TakeDamage(max_hp * fall_damage_percent);
            if (reset_when_fall)
                Teleport(last_ground_pos);
        }
    }
```

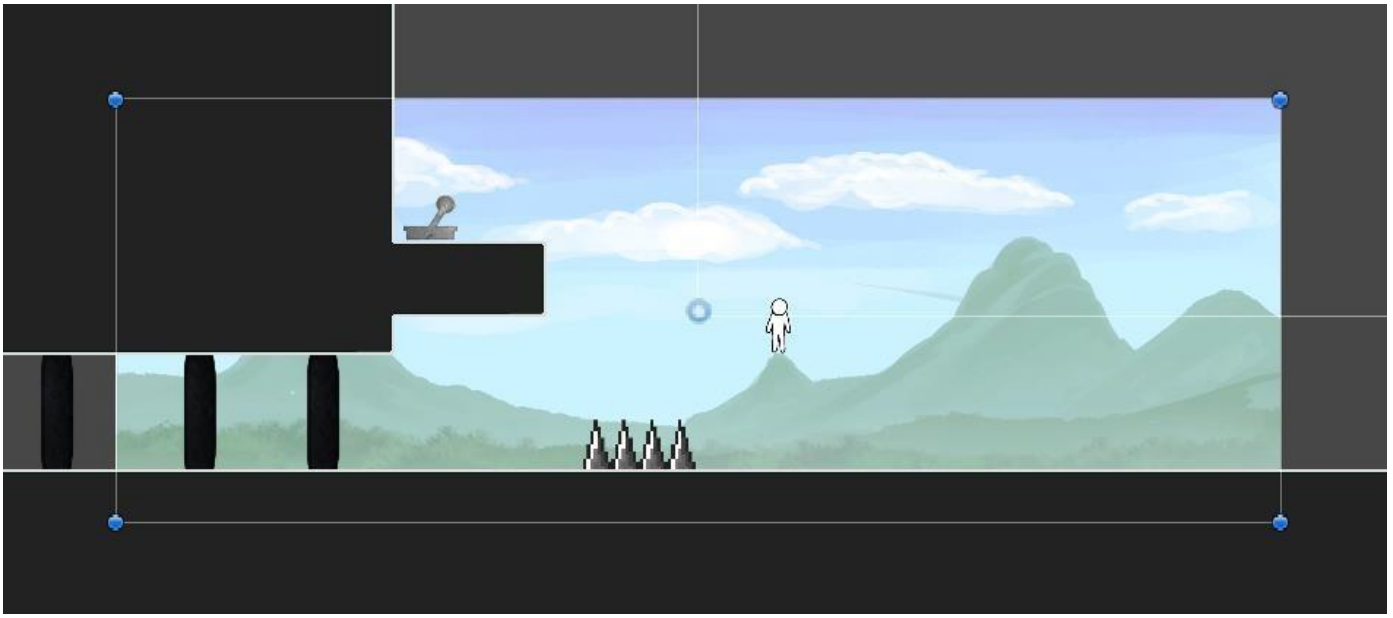
Выше приведены 2 основных цикла для работы персонажа игрока.

II.Главная карта:



Основная карта сделана из тайловой карты, сгенерированной из "GRID" и "TILEPALETTE".
В Tilemap встроен коллайдер, поэтому он может сталкиваться с «Rigidbody2D» игрока, чтобы игрок мог стоять на местности.

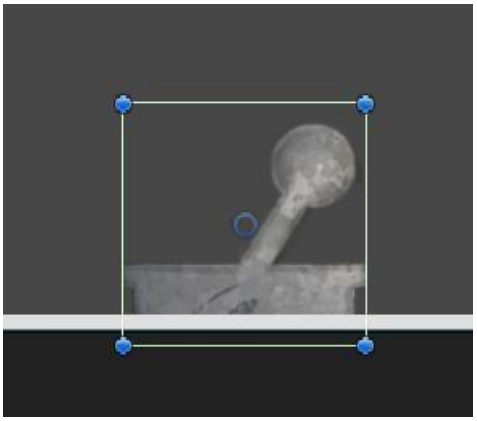
III. Following Background:



```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class BackgroundFollow : MonoBehaviour
{
    private Vector3 offset = new Vector3(0f, 0f, 10f);
    [SerializeField] private Transform camera;

    // Update is called once per frame
    void Update()
    {
        transform.position = camera.position + offset;
    }
}
```

Фон будет автоматически следовать за камерой игрока

IV. Levels and Doors:



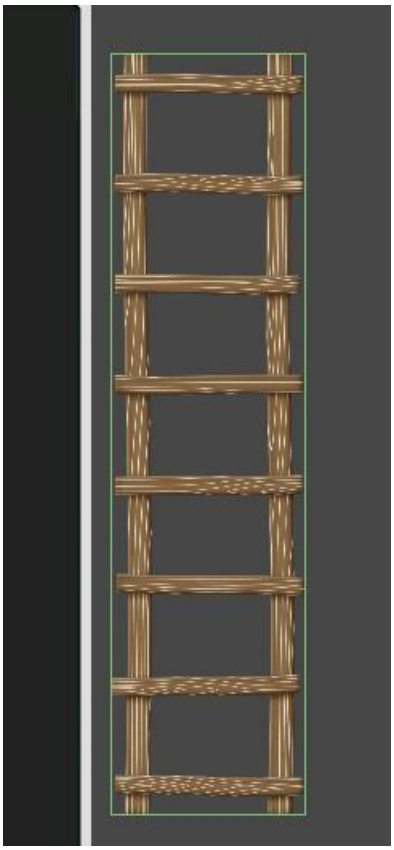
```
void Update()
{

    timer += Time.deltaTime;

    if (state != prev_state)
    {
        ChangeSprite();
        prev_state = state;
    }
}

private void OnDestroy()
{
    levers.Remove(this);
}

void OnTriggerEnter2D(Collider2D coll)
{
    if (coll.gameObject.GetComponent<fllayerCharacter>())
    {
        if (state == LeverState.disabled)
            return;

        Activate();
    }
}
```

Включает в себя: Sprite, Collider2D, ... используется для открытия и закрытия дверей

```
void FixedUpdate()
{
    //Get nb switch on
    int nb_switch = GetNbSwitches();

    //keys
    nb_switch += nb_keys_inside;

    //Open door
    bool activated = (nb_switch >= nb_switches_required);
    Vector3 move_dir = GetMoveDir();
    should_open = opened_at_start ? !activated : activated;
    target_pos = transform.position;

    if (should_open)
    {
        Vector3 diff = transform.position - initialPos;
        if (open_speed >= 0.01f && diff.magnitude < max_move)
        {
            target_pos = initialPos + move_dir.normalized * max_move;
            target_pos.z = 1f;
        }
    }
    else
    {
        Vector3 diff = transform.position - initialPos;
        float dot_prod = Vector3.Dot(diff, move_dir);

        if (close_speed >= 0.01f && dot_prod > 0.001f && diff.magnitude > 0.01f)
        {
            target_pos = initialPos;
            target_pos.z = 1f;
        }
    }
}

private void Update()
{
    Vector3 move_dir = target_pos - transform.position;
    if (move_dir.magnitude > 0.01f)
    {
        float speed = should_open ? open_speed : close_speed;
        float move_dist = Mathf.Min(speed * Time.deltaTime, move_dir.magnitude);
        transform.position += move_dir.normalized * move_dist;
    }
```

Дверь имеет индекс и может быть разблокирована на соответствующих уровнях.

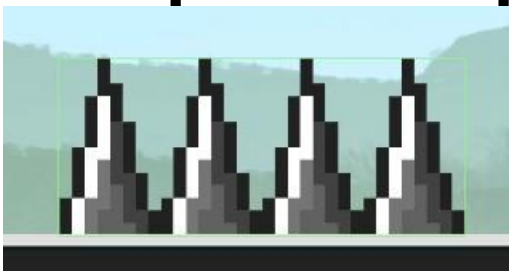
V. Ladders:



```
if (isClimbing)
{
    rigid.gravityScale = 0f;
    rigid.velocity = new Vector2(rigid.velocity.x, vertical * speed);
    return;
}
else
{
    rigid.gravityScale = normalGravity;
}
```

Когда игрок сталкивается с коллайдером Лестницы, игрок сможет подняться, уменьшив вес игрока = 0

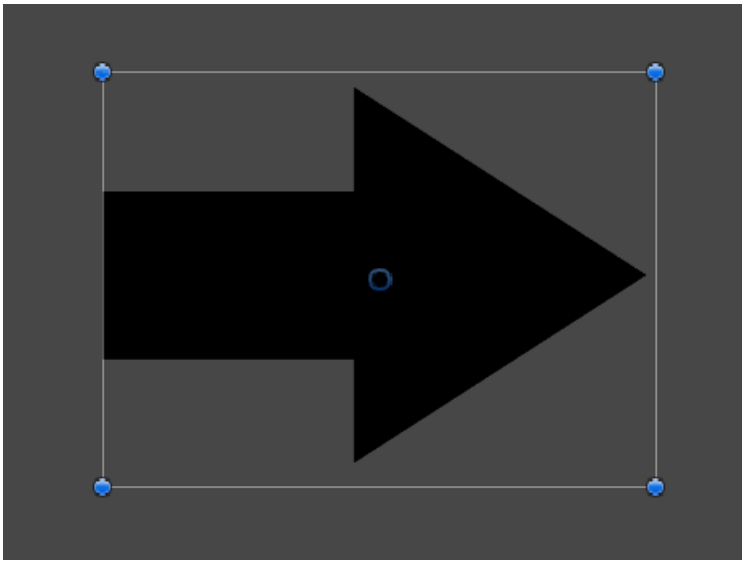
VI. Spikes Trap:



```
if (isInsideSpikes)
{
    TakeDamage(10);
}
```

Когда игрок входит в коллайдер 2D шипов, игрок получает урон, уменьшенный до полосы HP.

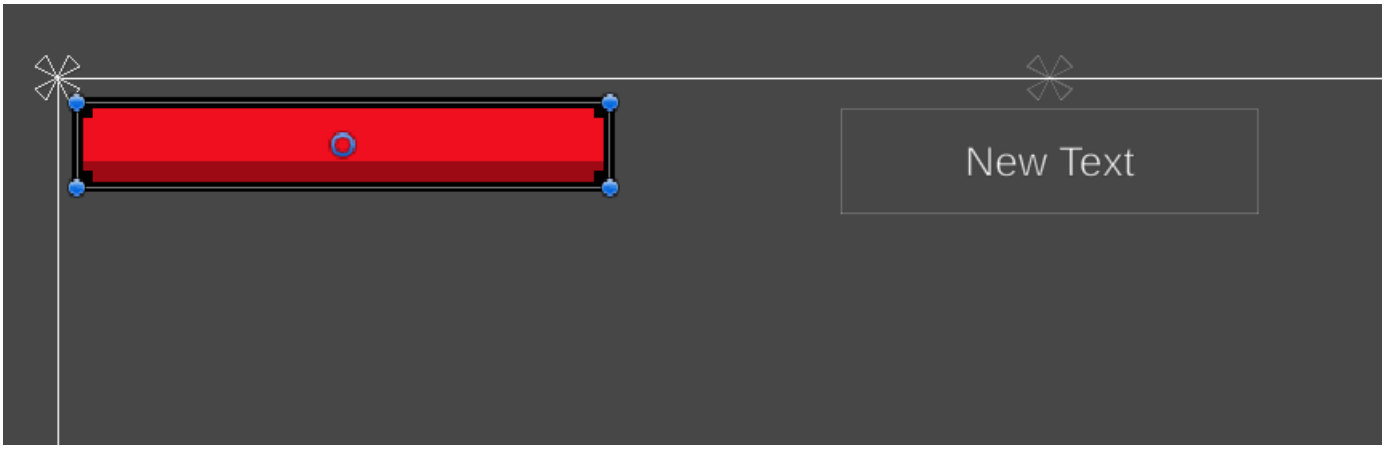
VII. Mission Arrows:



```
public Transform target;
public bool isreachedtarget = false;

// Update is called once per frame
void Update()
{
    if(isreachedtarget)
    {
        foreach (Transform child in transform)
        {
            GameObject.Destroy(child.gameObject);
        }
        Destroy(gameObject);
        return;
    }
    var dir = target.position - transform.position;

    var angle = Mathf.Atan2(dir.y, dir.x) * Mathf.Rad2Deg;
    transform.rotation = Quaternion.AngleAxis(angle, Vector3.forward);
}
```

Стрелка миссии будет вращаться в направлении 4 основных уровней, чтобы помочь игрокам найти их и выиграть игру.

VIII. Canvas:

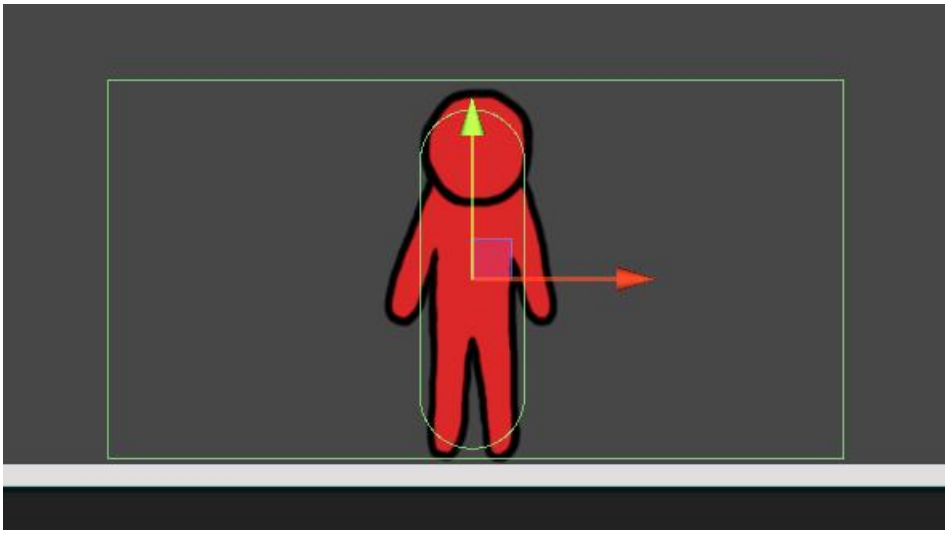


Canvas включает полосу здоровья и квесты textMeshPro для игроков.
HP Bar – это Slider, и мы можем настроить для него значение.

```
public Slider slider_hp;
public void TakeDamage(float damage)
{
    if (!is_dead && !invulnerable && hit_timer > 0f)
    {
        hp -= damage;
        hit_timer = -1f;

        if (hp <= 0f)
        {
            slider_hp.value = 0;
            Kill();
        }
        else
        {
            slider_hp.value = hp;
            if (onHit != null)
                onHit.Invoke();
        }
    }
}
```

Новые улучшения для 2-го семестра

IX. Enemy:



Если ни один игрок не обнаружен, противник будет свободно перемещаться между двумя предустановленными точками.

```
    private void Update()
    {
        if (isPlayerDetected)
        {
            StopAndAttackPlayer();
        }
        else
        {
            MoveToWaypoint();
        }
     }
```

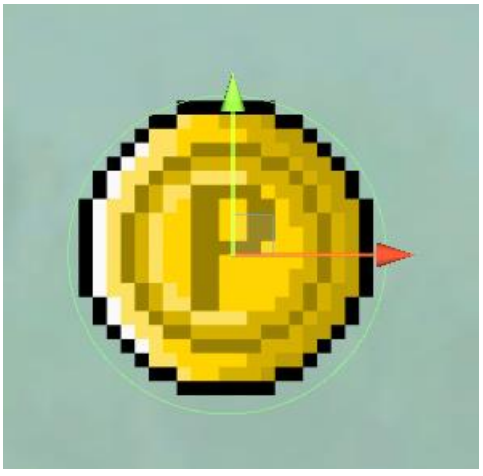
Если игрок обнаружен, противник остановится и атакует игрока, в противном случае противник продолжит движение по 2 обозначенным точкам.

```
private void StopAndAttackPlayer()
    {
        if (player != null)
        {
            Vector3 moveDirection = player.position - transform.position;

            if (moveDirection.x < 0)
            {
                spriteRenderer.flipX = true;
            }
            else if (moveDirection.x > 0)
            {
                spriteRenderer.flipX = false;
            }

            if (moveDirection.magnitude > attackRange)
            {
                transform.Translate(moveDirection.normalized * moveSpeed * Time.deltaTime);
            }
```

```
            else
            {
                attackTimer += Time.deltaTime;
                if (attackTimer >= attackDelay)
                {
                    AttackPlayer();
                    attackTimer = 0f;
                }
            }
        }
    }

    private void AttackPlayer()
    {
        Debug.Log("Enemy is attacking player.");
        playerCharacter.TakeDamage(5);
     }
```

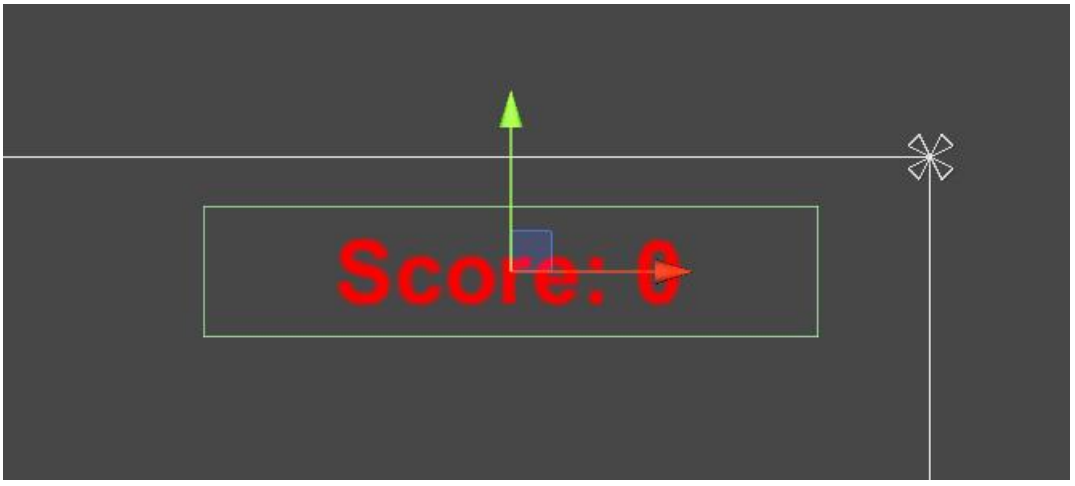
```
private void MoveToWaypoint()
    {
        Vector3 targetPosition = waypoints[currentWaypointIndex].position;
        moveDirection = targetPosition - transform.position;
        transform.Translate(moveDirection.normalized * moveSpeed * Time.deltaTime);

        if (moveDirection.x < 0)
        {
            spriteRenderer.flipX = true;
        }
        else if (moveDirection.x > 0)
        {
            spriteRenderer.flipX = false;
        }

        if (Vector3.Distance(transform.position, targetPosition) < 0.1f)
        {
            currentWaypointIndex = (currentWaypointIndex + 1) % waypoints.Length;
        }
     }
```

X. Coins:



Монета — это пункт, когда игрок подходит к монете и касается ее, ему добавляется 1 очко, отображаемое на экране.

```
private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.CompareTag("Coin"))
        {
            CollectCoin(collision.gameObject);
        }
    }

    private void CollectCoin(GameObject coin)
    {
        coin.SetActive(false);
        score++;
        UpdateScoreUI();
    }

    private void UpdateScoreUI()
    {
        scoreText.text = "Score: " + score.ToString();
    }
```



Создавайте больше цепочек монет, чтобы сделать игру более интересной.