Создание “груши для битья”

Для создания манекена, необходимо сделать следующие действия:

1. Создать пустой объект и назвать его “mannequin”
2. Объекту “mannequin” дать следующие компоненты:
    1. “SpriteRenderer” и настроить его как указано на рисунке 82
    2. “Capsule Collider 2D” и настроить его, как указано на рисунке 82
3. Добавить объекту “mannequin” слой “enemy”
4. Добавить объекту “mannequin” анимацию. Анимация должна содержать в себе вид, указанный на рисунке 83, а именно:
    1. Параметр “Hit” имеющий тип триггер
    2. ячейку “idle” содержащую в себе анимацию бездействия
    3. ячейку “Hit” содержащую в себе анимацию получения урона
5. Создать объект “3D Object” - “Text - TextMeshPro”, как указано на рисунке 84, после чего изменить название на “DMG”.
6. Объект “DMG” должен иметь следующие компоненты:
    1. “Rect Transform” и оставить компонент без изменений
    2. “Mesh Renderer” и оставить компонент без изменений
    3. “TextMeshPro - Text” и подвергнуть компонент изменениям, как на рисунке 85
    4. “Rigidbody2D” с изменениями, указанными на рисунке 86

5. “Liberation Sans SDF Material” с изменениями, указанными на рисунке 87

7. Добавить объект “DMG” в префаб и удалить из окна иерархии. Для того, чтобы сделать из объекта префаб, его необходимо перенести из окна иерархии в окно проекта. Префаб - это оригинал объекта, для дублирования.

8. Создать скрипт с названием “DMG”, и добавить его к префабу “DMG”. Код скрипта описан ниже.

9. Создать скрипт с названием “mannequin”, и добавить его к объекту “mannequin”. Код скрипта описан ниже.

10. Добавить код в скрипт “Player”, который будет описан ниже.

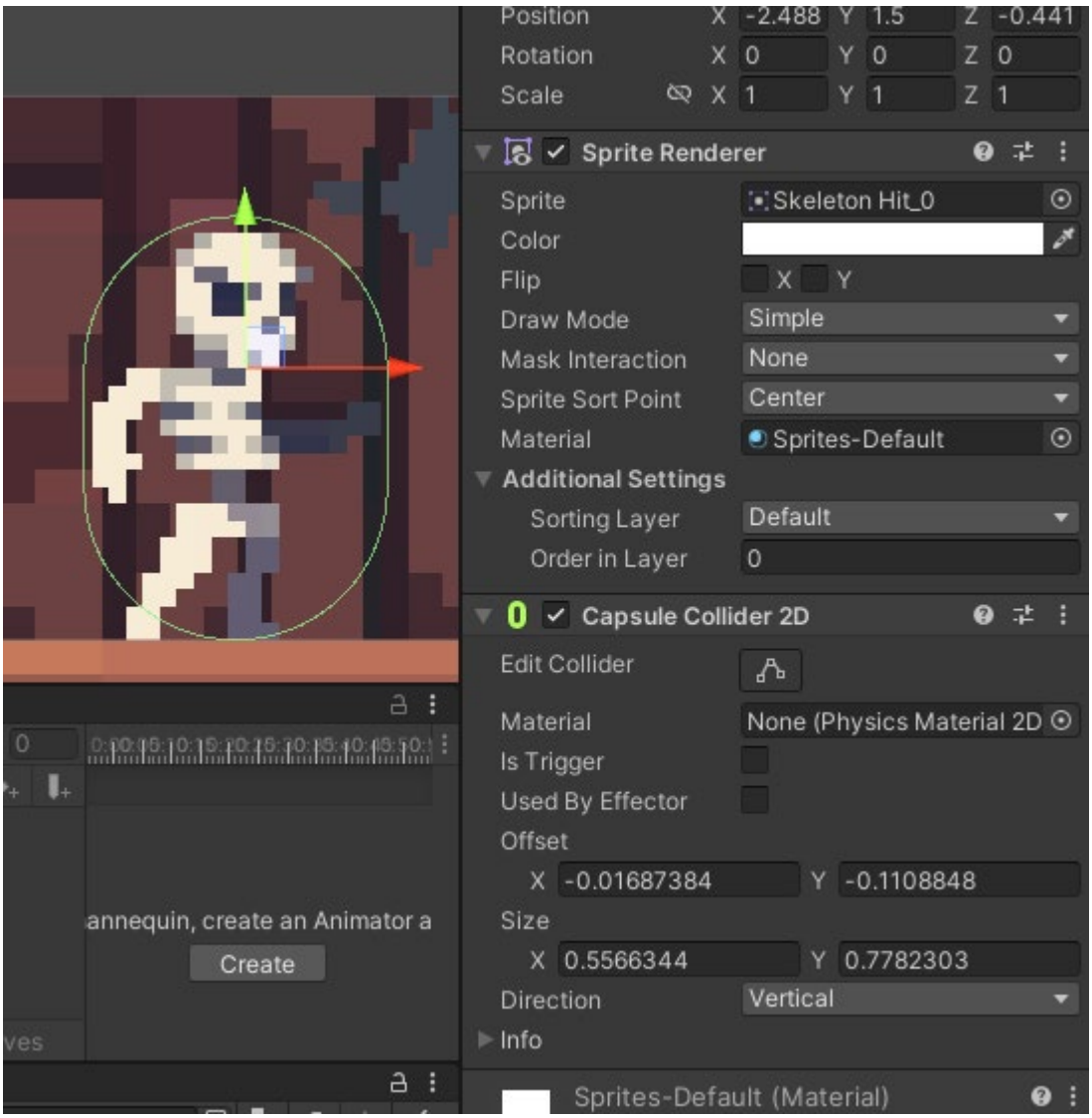

Рисунок 82. Настройки манекена

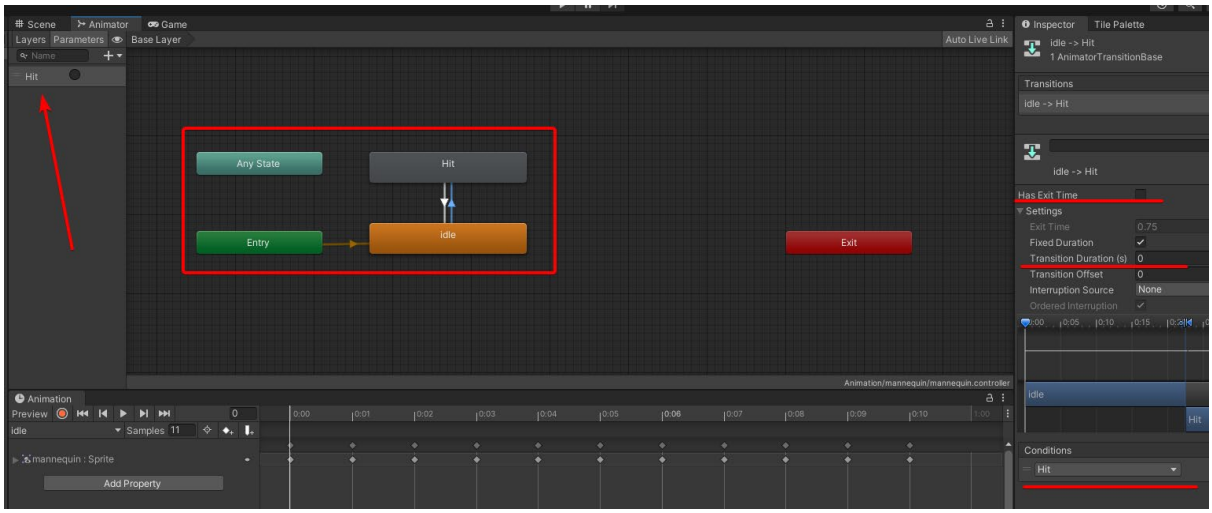

Рисунок 83. Изменения аниматора

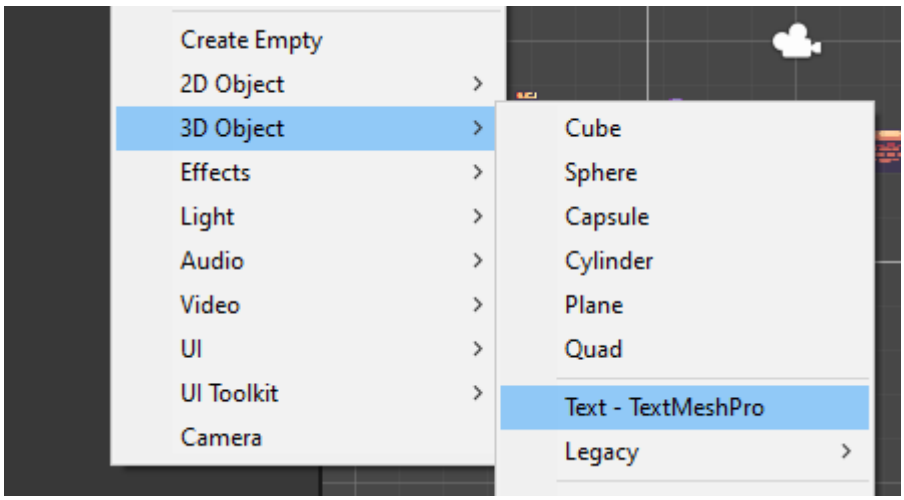

Рисунок 84. Создание “Text - TextMeshPro”

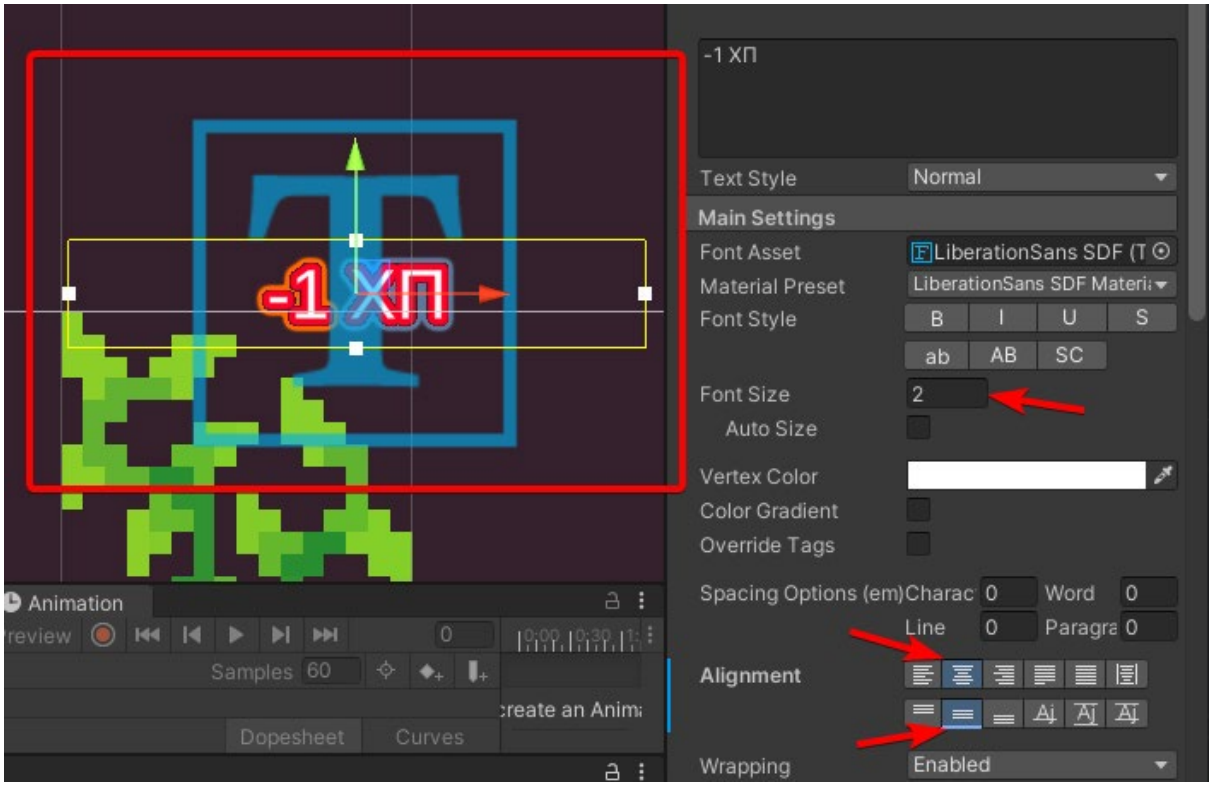

Рисунок 85. Настройки объекта “DMG”

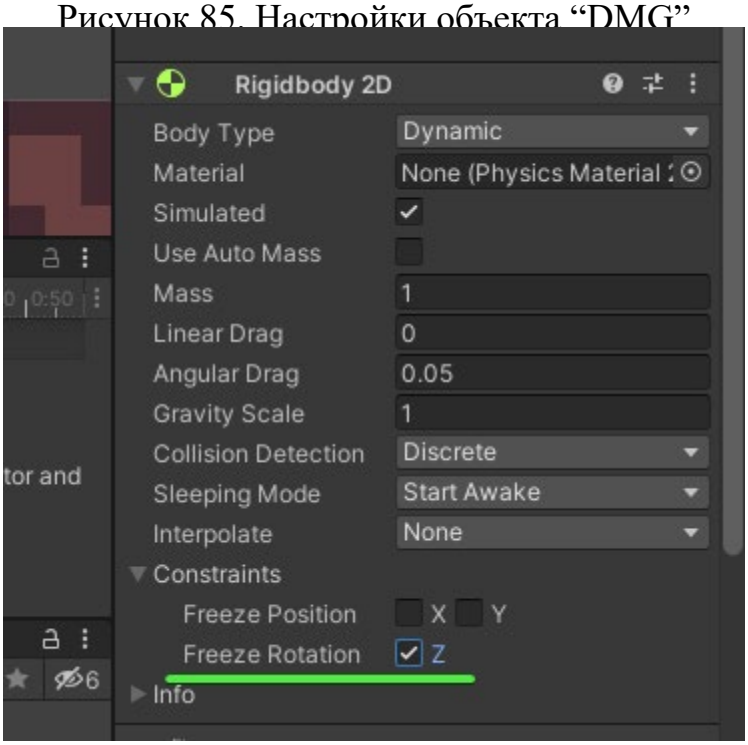

Рисунок 86. Изменения компонента

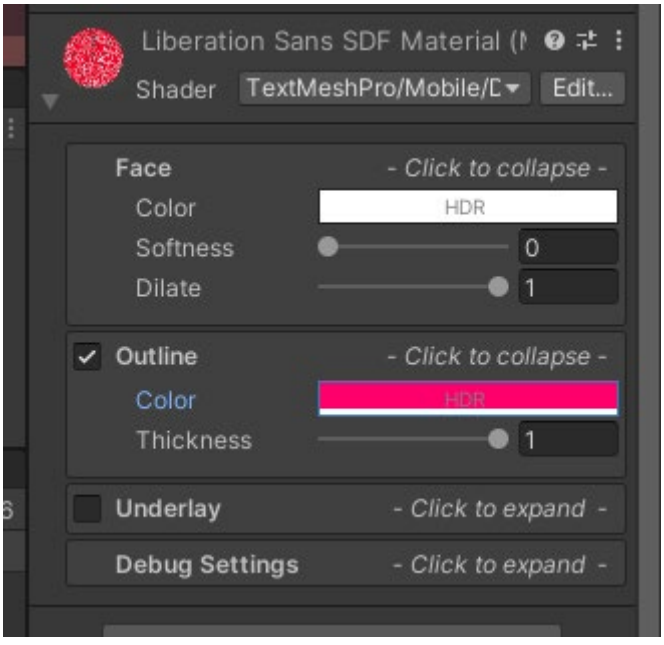

Рисунок 87. Изменения компонента

Скрипт “DMG” должен содержать в себе следующий код:

Подключаемые библиотеки для работы с кодом

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class DMG : MonoBehaviour

{

Создаем переменную, в которую будет положен компонент отвечающий за физику объекта

[SerializeField] Rigidbody2D rb2d;

Метод, активирующийся единожды при запуске сцены

private void Start()

{

Присваивает переменной новую скорость, по оси x и y

rb2d.velocity = new Vector2(0, 5);

Вызывает срабатывание метода с названием “Damage()”

StartCoroutine(Damage());

}

Метод сработает через 0.5 секунды, который удаляет объект, на котором находится данный скрипт и останавливает работу этого метода.

IEnumerator Damage()

{

yield return new WaitForSeconds(0.5f);

Destroy(gameObject);

StopCoroutine(Damage());

}

}

После добавления данного кода объекту “DMG”, его параметры должны выглядеть, как указано на рисунке 88.

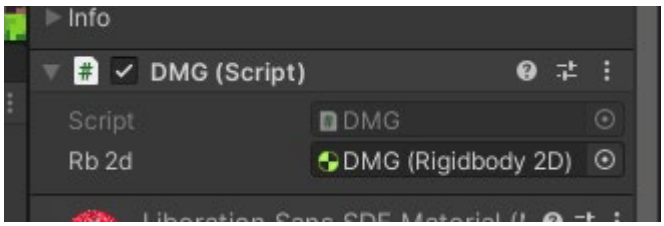

Рисунок 88. Параметры скрипта

Скрипт “mannequin” должен содержать в себе следующий код:

Подключаемые библиотеки для работы с кодом

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class mannequin : MonoBehaviour

{

Создаем переменную, в которую будет положен компонент отвечающий за анимацию объекта

Animator _animator;

Создаем переменную, которая отвечает за максимально возможное значение здоровья персонажа

int _maxHealth = 71;

Создаем переменную, которая отвечает за значение здоровья у персонажа на данный момент

int _currentHealth;

Метод, активирующийся единожды при запуске сцены

void Start()

{

Присваиваем переменной компонент аниматор

_animator = GetComponent<Animator>();

Присваиваем значение максимального здоровья текущему

_currentHealth = _maxHealth;

}

Метод, который сработает через 1 секунду и обновит данные переменной, после чего остановит сам себя

IEnumerator Heal()

{

yield return new WaitForSeconds(1f);

_currentHealth = 71;

StopCoroutine(Heal());

}

Метод, принимающий целочисленное значение

public void TakeDamage(int damage)

{

Активирует стрелочку в окне “Animator”, которая имеет название “Hit”

_animator.SetTrigger("Hit");

Значение переменной “*_currentHealth*” равно переменной у которой отняли значение переменной “damage”

_currentHealth -= damage;

Оператор, проверяющий, если значение переменной меньше или равно нулю, активирует метод “Heal()”

if (_currentHealth <= 0)

{

StartCoroutine(Heal());

}

}

}

После, необходимо добавить в окно аниматора параметр “Attack”, имеющим тип триггер. После чего добавить анимацию атаки с названием “attack” и в окне аниматора сделать необходимые действия. А именно, из ячейки “Any State” дать стрелочку, которая имеет параметр “Attack” и

измененные параметр “Transition Duration” равный 0, в ячейку “attack” , а из нее в ячейку “idle”.

Также скрипт “Player” должен иметь следующие строки:

Создается переменная для принятия координат, где будет находиться область атаки

public Transform _attackPoint;

Создается переменная для радиуса атаки

float _attackRange = 0.64f;

Создается переменная для выбора слоя, в котором будет происходить поиск врагов

public LayerMask EnemyLayers;

Создается переменная, которая отвечает за наносимый урон героем

int _attackDamage = 5;

Создается переменная для задержки между ударами

float _attackRate = 2f;

Создается переменная для разрешения использовать удар

float _nextAttackTime = 0f;

Создается переменная для создания объекта

[SerializeField] GameObject _showDamage;

Создается переменная для редактирования отображаемого текста

[SerializeField] TMP_Text DMG;

void Update()

{

Оператор, который следит за разрешение следующего использования атаки

if (Time.time >= _nextAttackTime)

{

Оператор, который при нажатии на левую кнопку мыши и нахождении персонажа на земле, обновляет данные в переменной "_nextAttackTime" и вызывает метод "Attack()"

if (Input.GetButtonDown("Fire1") && GroundCheck.isGround())

{

Переменная, которая высчитывает следующую возможность для атаки

_nextAttackTime = Time.time + 2f / _attackRate;

Активирует метод

Attack();

}

}

}

Приватный метод "Attack()", который активирует анимацию атаки,

void Attack()

{

Запустить анимацию с именем "Attack"

animator.SetTrigger("Attack");

Массив коллайдеров, которые берутся из круга, находящимся в точке, которая указана в переменной “*_attackPoint*”, с радиусом, взятым из переменной “*_attackRange*”, и слоем, указанным в переменной “*EnemyLayers*”.

Collider2D[] hitEnemies = Physics2D.OverlapCircleAll(_attackPoint.position, _attackRange, EnemyLayers);

Цикл, который проверяет коллайдеры имеющие слой с названием “*enemy*” из массива “*hitEnemies*”

foreach (Collider2D enemy in hitEnemies)

{

Определяет текст, который будет содержать переменная

DMG.text = "-" + _attackDamage + " ХП";

Активирует метод, который передает переменную “*_attackDamage*” в скрипт “*mannequin*”, находящийся в объекте, взятом из слоя “enemy”

enemy.GetComponent<mannequin>().TakeDamage(_attackDamage);

Создает объект, указанный в переменной “_showDamage”, на месте, указанном с помощью “enemy.transform.localPosition”, и градусом, взятым из оригинала

Instantiate(_showDamage, enemy.transform.localPosition, Quaternion.identity);

}

}

Метод, который рисует на сцене редактора, для отображения и редактирования невидимых объектов, таких как переменная “_attackRange”.

```
 private void OnDrawGizmos()//ля

{

    if (_attackPoint == null)

    return;

    Gizmos.DrawWireSphere(_attackPoint.position, _attackRange);

}
```

Параметры скрипта “Player” будут выглядеть, как указано на рисунке 89.

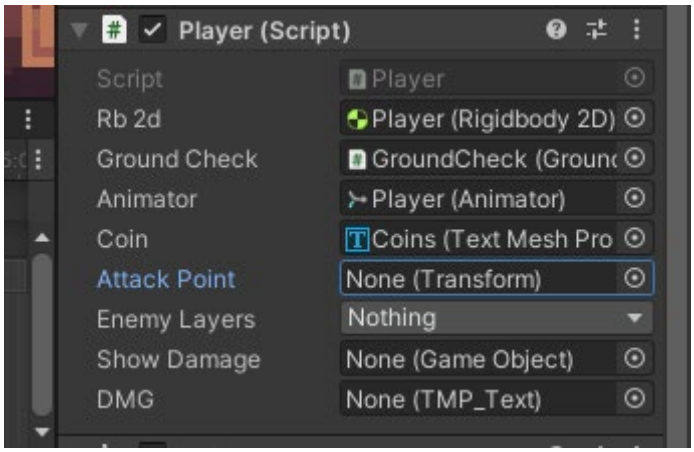

Рисунок 89. Не заполненные параметры скрипта “Player”

Для заполнения параметра “AttackPoint” необходимо в объекте “Player” создать пустой объект с названием “AttackPoint” и поместить его перед персонажем.

Для заполнения параметра “Enemy Layers” необходимо создать слой “enemy” как объяснялось выше.

Для заполнения параметров “Show Damage” и “DMG” необходимо параметрам присвоить префаб “DMG”.

После изменений в скрипте “Player” он будет выглядеть, как указано на рисунке 90

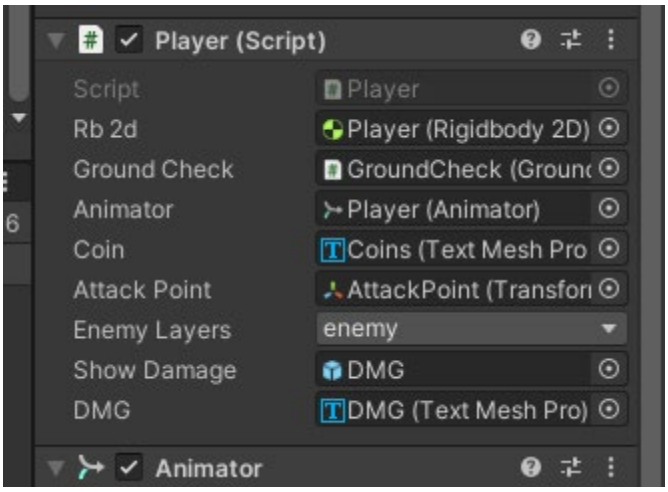

Рисунок 90. Заполненные параметры скрипта “Player”