

Лабораторная работа. Создание 2D-игры Snake.

Данный урок создан на основе следующего видео-урока: [How to make Snake in Unity \(Complete Tutorial\)](#)

Цель работы: Создание 2D-игры Snake с использованием игрового движка Unity и инструментов разработки, что позволяет получить практические навыки в программировании, проектировании игры и работе с графикой.

Задачи:

1. Разработка концепции игры:

- Определение основных механик и правил игры.
- Разработка схемы управления и взаимодействия пользователя.

2. Создание пользовательского интерфейса (UI):

- Разработка основного экрана игры.

3. Программирование игрового процесса:

- Реализация движения змейки.
- Обработка столкновений (с едой, границами игрового поля и самой змейкой).
- Реализация системы увеличения змейки при поглощении еды.
- Реализация случайного появления еды на игровом поле.
- Реализация системы очков

4. Тестирование и отладка:

- Проведение тестирования для выявления багов и ошибок.
- Оптимизация игрового процесса для улучшения производительности.

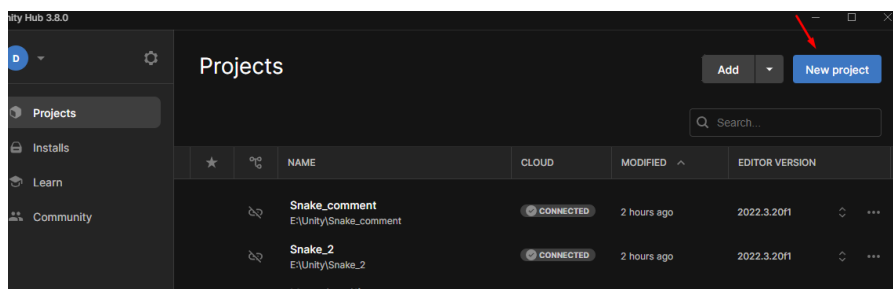
5. Финальные настройки и сборка проекта:

- Провести рефакторинг кода для улучшения читаемости и производительности.
- Скомпилировать и сохранить финальную версию игры.

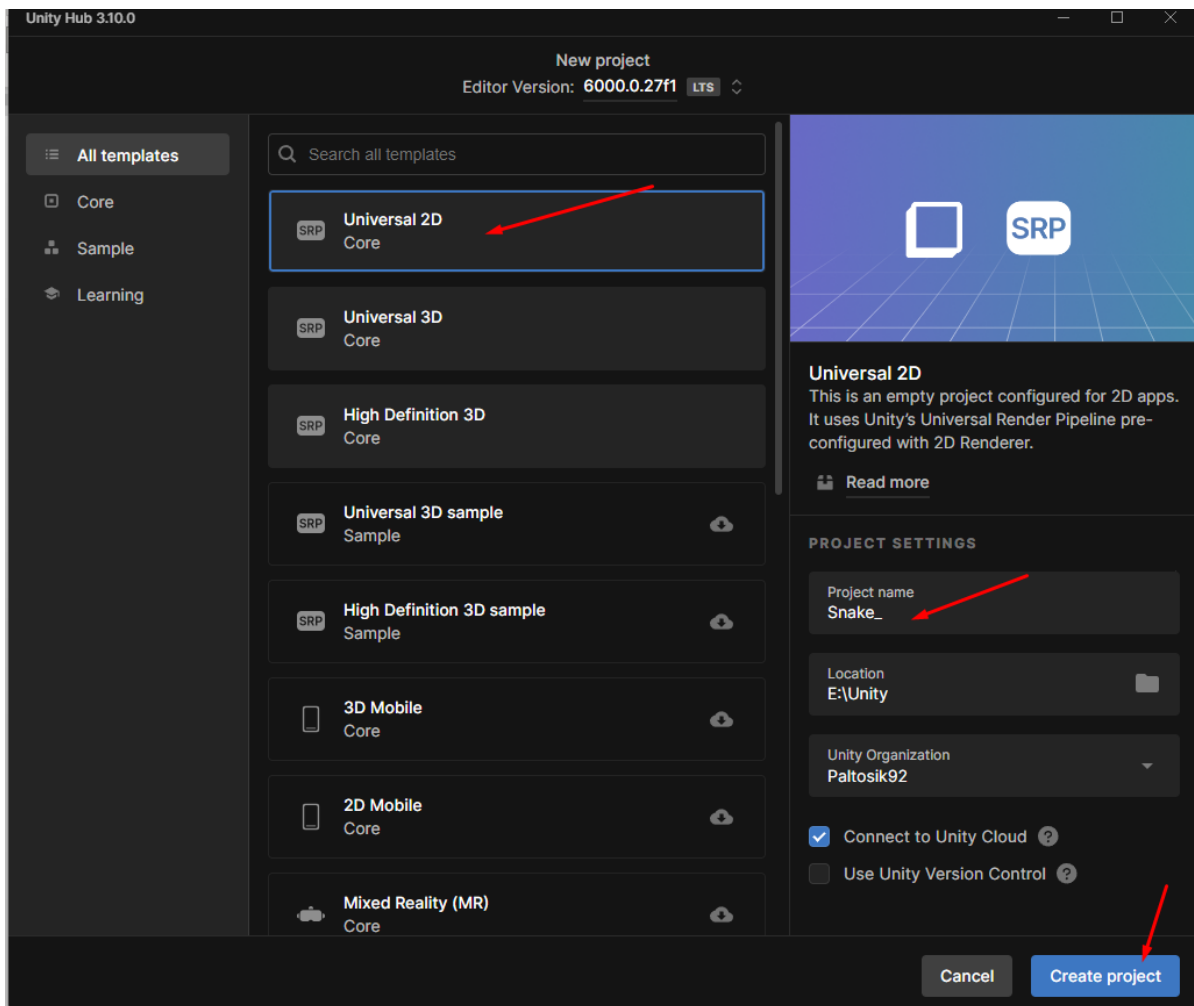
Ход работы:

1. Запускаем Unity Hub.

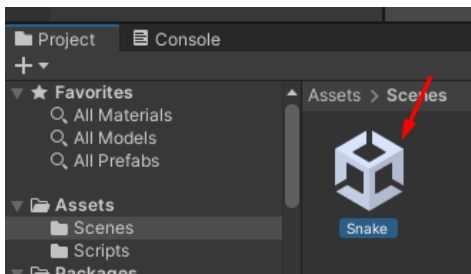
2. Создаём новый проект – New project:



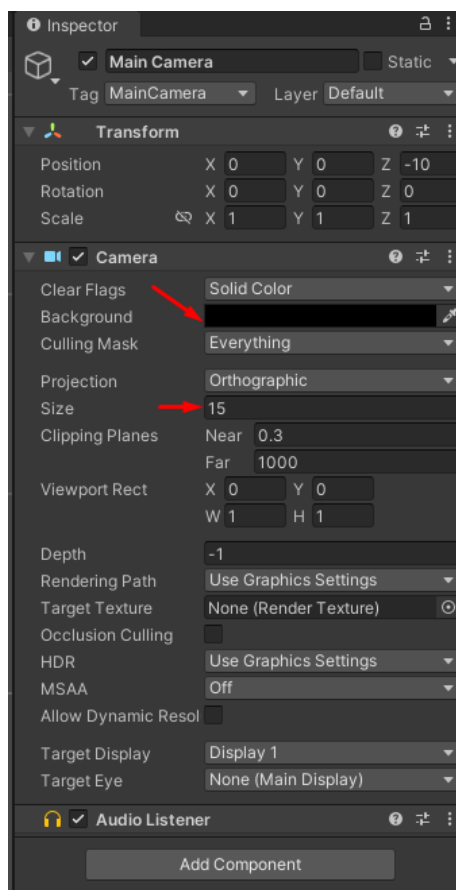
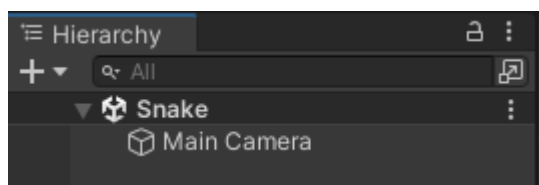
3. Выбираем **2D**. Вводим название проекта, выбираем место расположения, и нажимаем **Create project**.



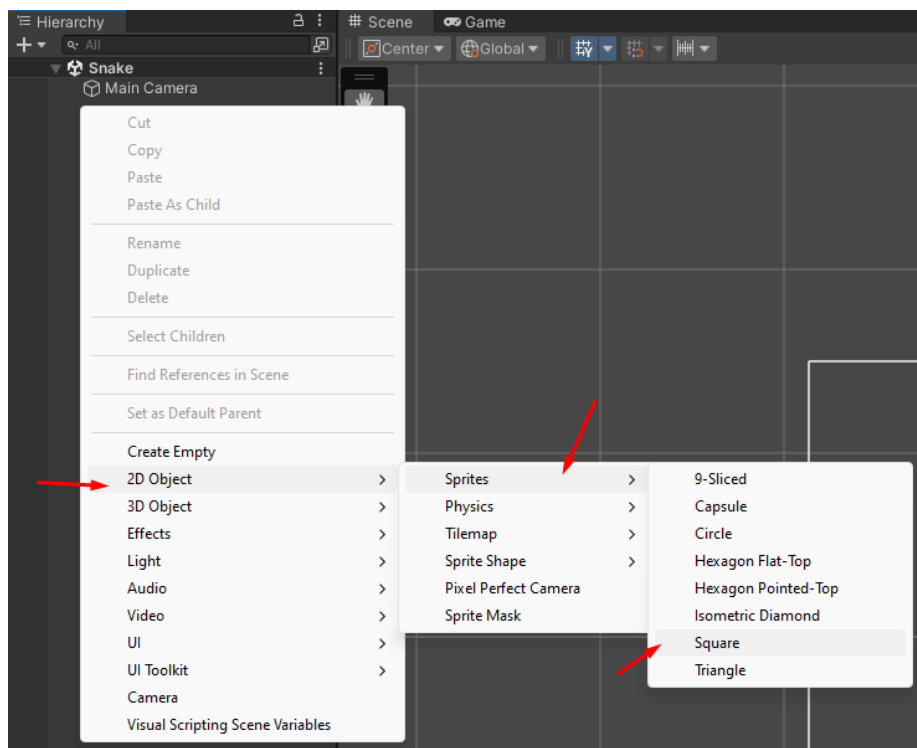
4. В папке **Scenes** меняем название сцены, на имя игры:



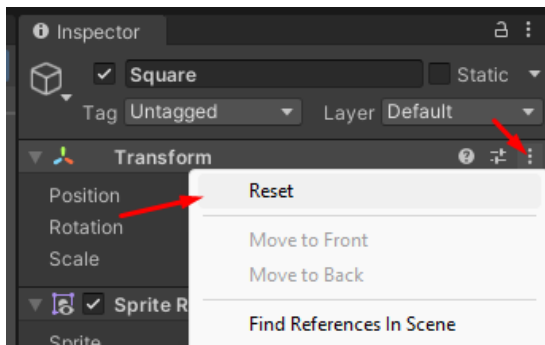
5. Меняем цвета объекта **Main Camera** (нажимаем на него в **Hierarchy**, и в **Inspector** появляются свойства объекта) на тот который вам нравится, и также меняем размер, в нашем случае возьмём 15 (указывает на область, которую видит наша камера, и мы отдадим наш zoom на кратное пяти число, для более удобных расчётов в будущем):



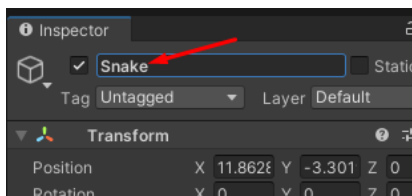
6. Щёлкаем правой кнопкой мыши в **Hierarchy** → **2D Object** → **Sprites** → **Square**:



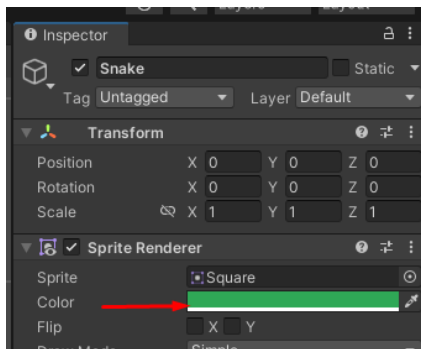
7. В **Inspector** сбрасываем для нашего объекта трансформацию (не забудьте, что объект должен быть выделен):



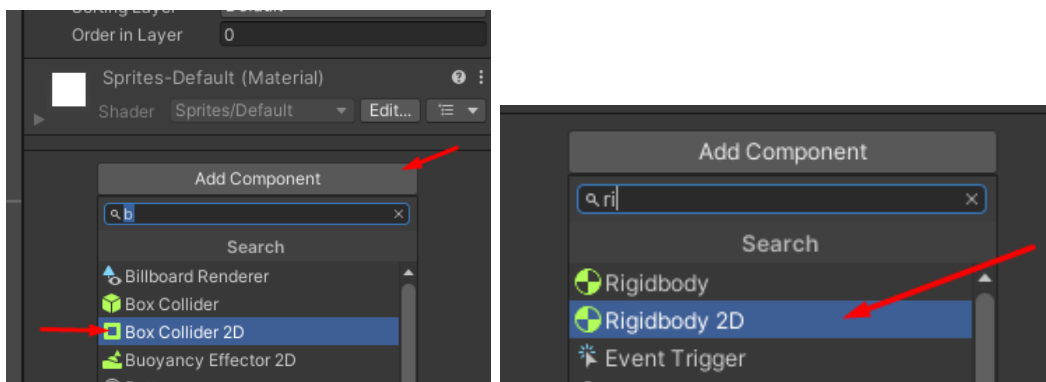
Меняем название на **Snake**:



Меняем цвет змейки (например, зеленый):



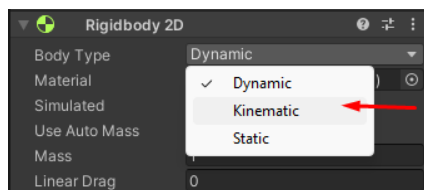
8. Добавляем два компонента, нажав на **Add Component - Box Collider 2D** и **Rigidbody 2D**:



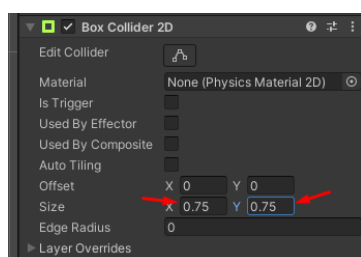
Rigidbody 2D – Добавление компонента (класса) Rigidbody2D к спрайту передает его под контроль физического движка. Само по себе это означает, что на спрайт будет воздействовать сила тяжести, и им можно управлять из скриптов с помощью сил. При добавлении соответствующего компонента collider спрайт также будет реагировать на столкновения с другими спрайтами.

Box Collider 2D – Коллайдер для 2D-физики, представляющий собой прямоугольник, выровненный по оси, влияет на столкновение объектов.

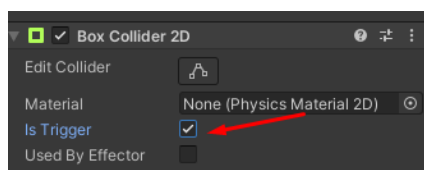
Так как игра у нас простая, на не нужны такие параметры как масса, гравитация и др., поэтому меняем **Body Type** на **Kinematic**:



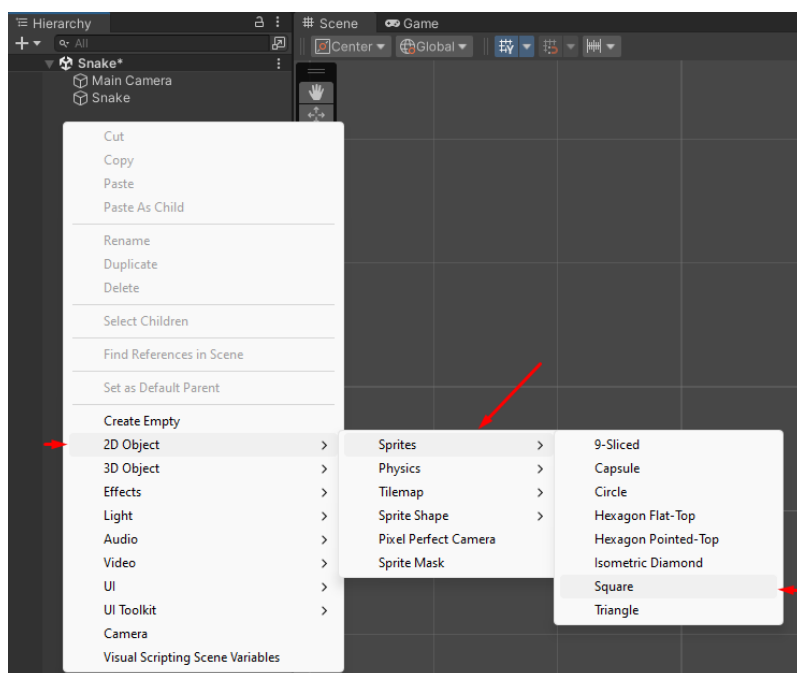
У **Box Collider 2D** изменим размер на **0.75** (чтобы при соприкосновении с собой игра не завершалась, или еда, которую мы позже добавим, не исчезала при касании боком):



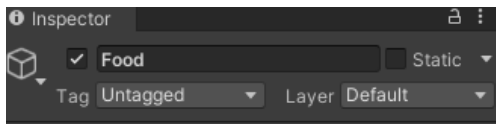
9. Ставим галочку **Is Trigger** (триггеры не вызывают физических столкновений, но используются для обнаружения определённых событий или реализации игровой механики.):



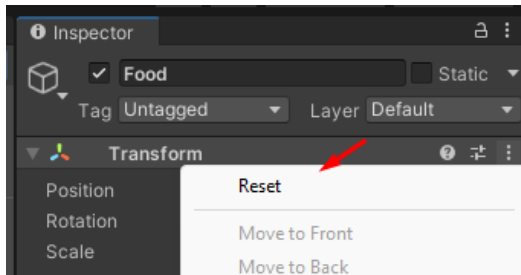
10. Создаём новый 2D объект:



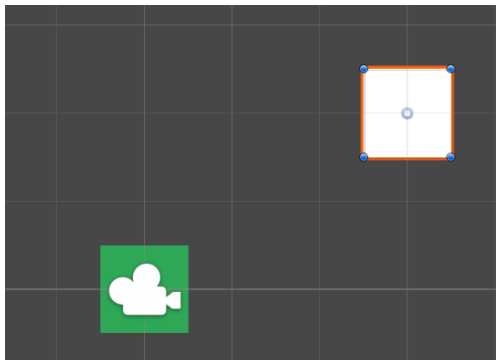
Называем его **Food**:



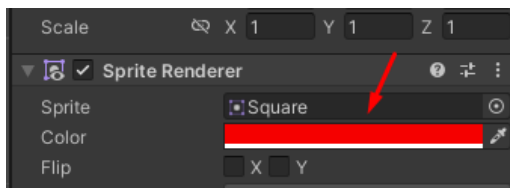
Сбрасываем трансформацию:



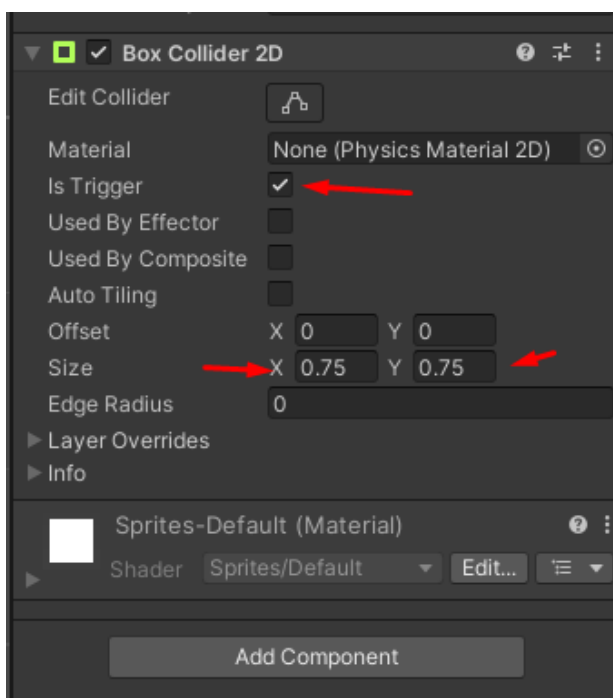
И перенесём немного в сторону от еды:



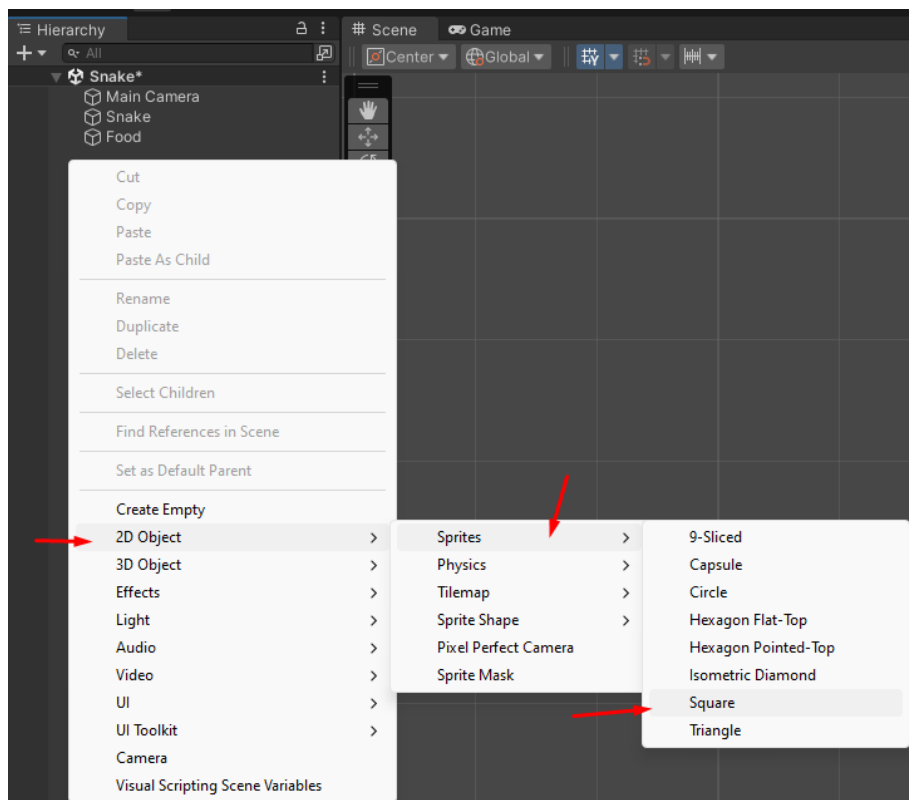
Меняем цвет (например, красный):



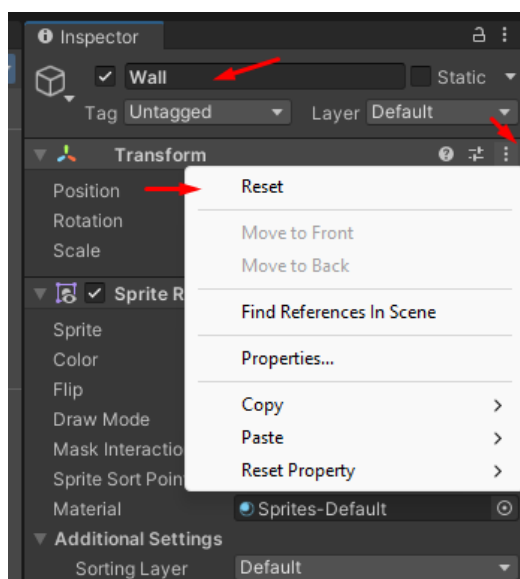
Добавляем **Box Collider 2D** (ставим галочку **Is Trigger** и меняем размер на **0.75**):



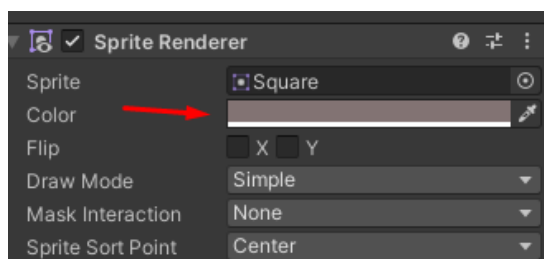
11. Создадим стены:



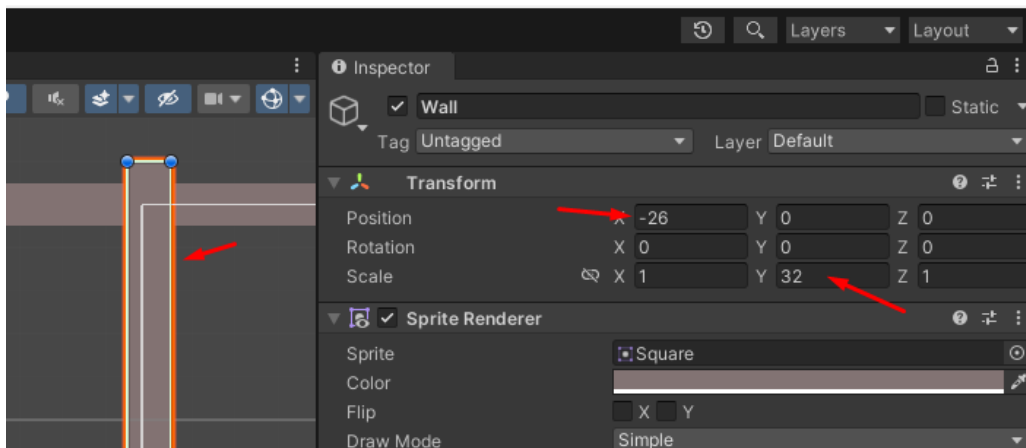
Меняем название и сбрасываем трансформацию:



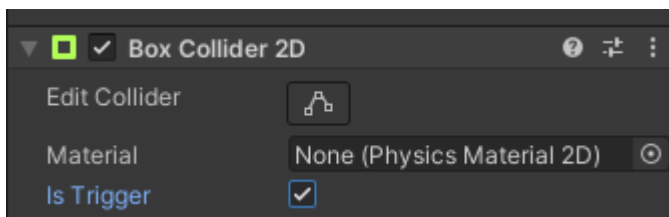
Меняем цвет:



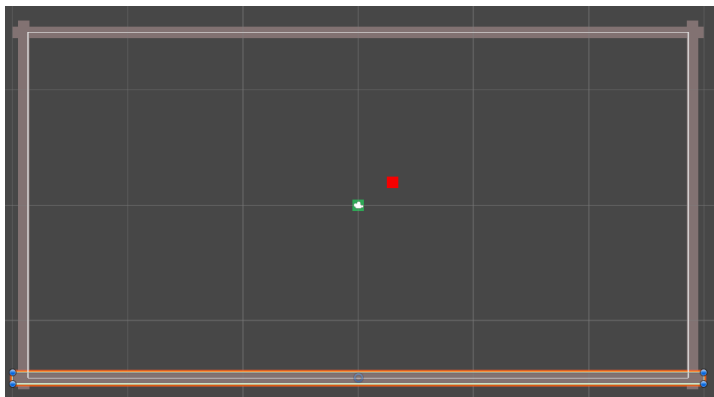
12. Передвигаем влево **position** и увеличиваем размер **scale** (убедитесь, что у вас целые числа):



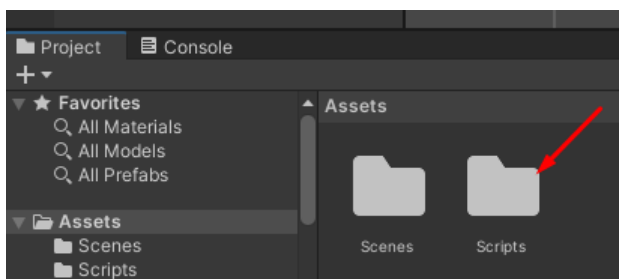
Добавляем **Box Collider 2D** (ставим галочку **Is Trigger**):



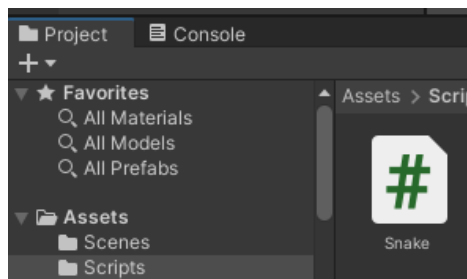
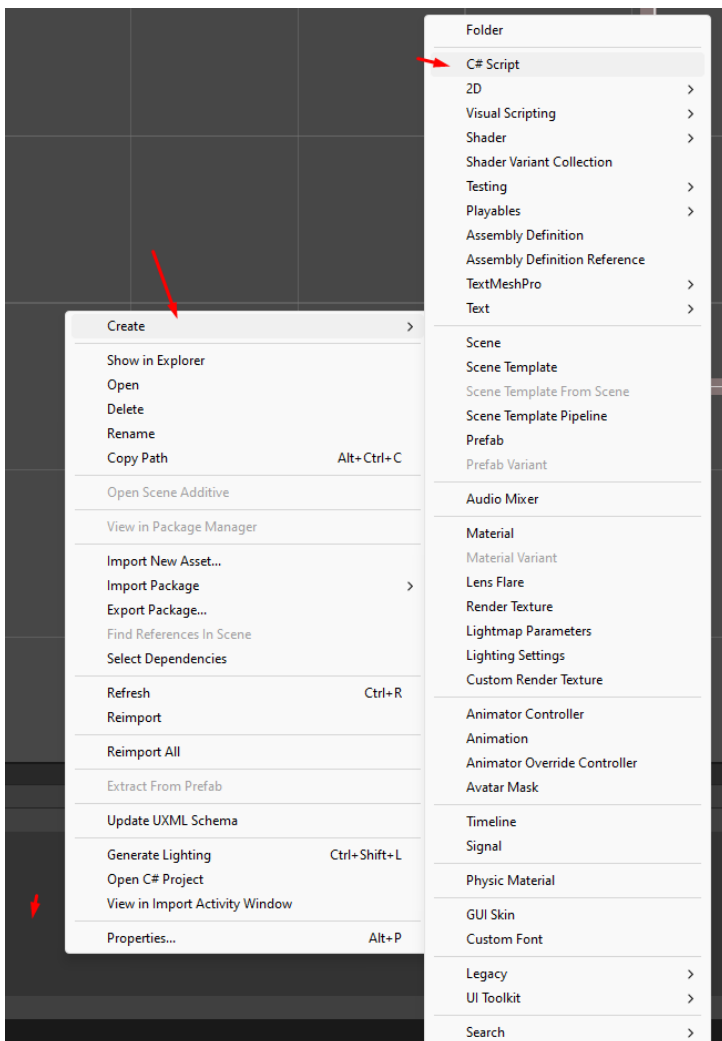
13. Далее создаём дубликаты наших стен, и меняем для них позицию и размер, чтобы получилось следующим образом:



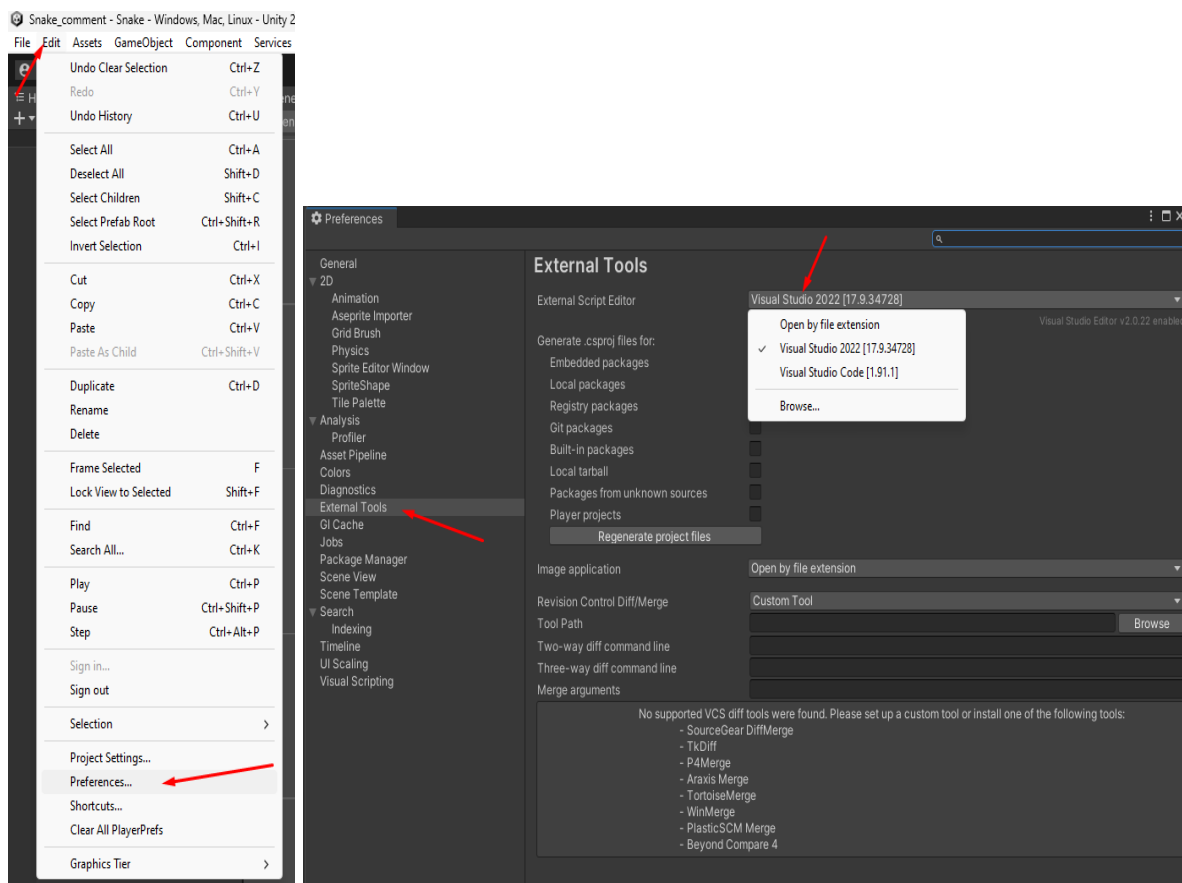
14. В папке **Assets** создаём папку **Scripts**:



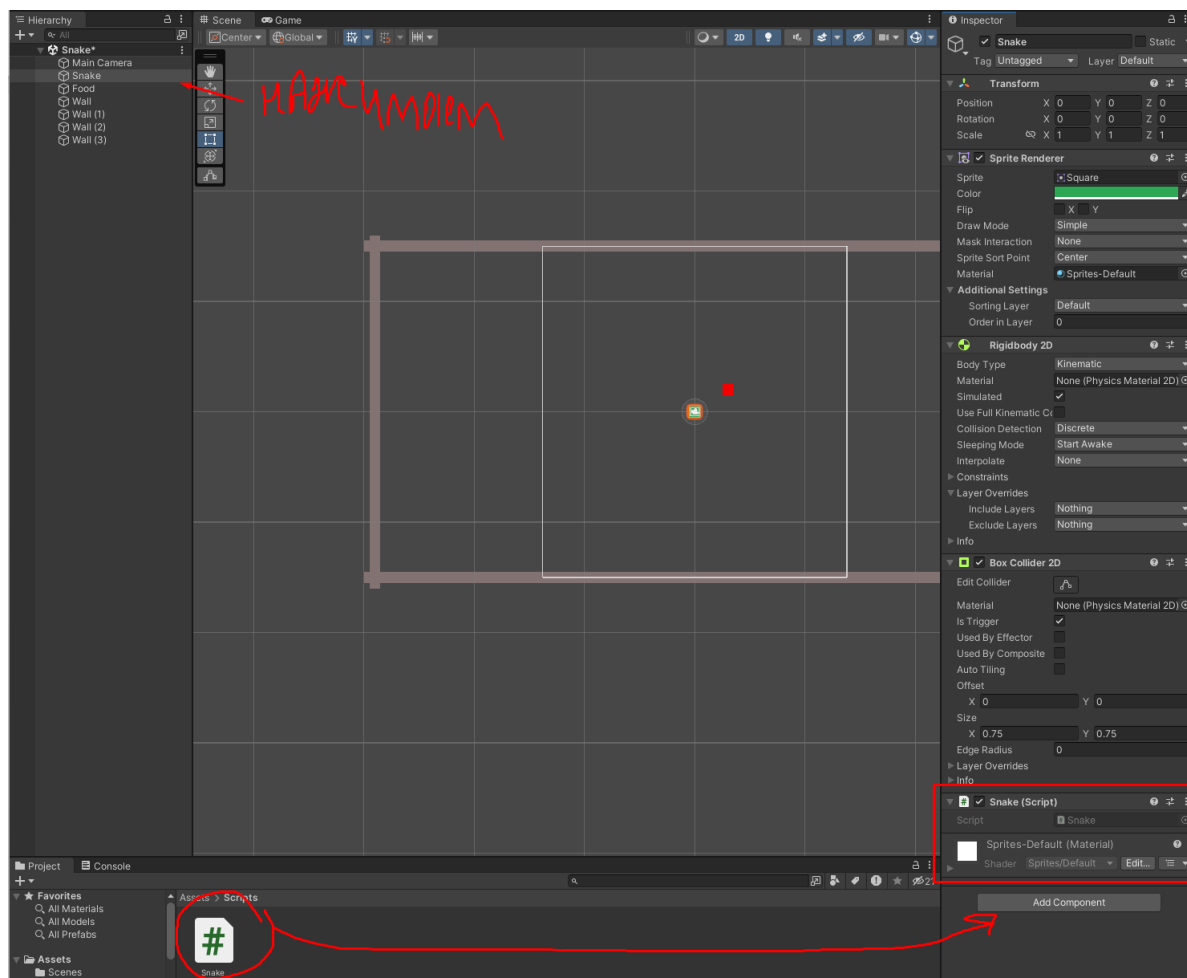
В ней создаём скрипт (щёлкаем правой кнопкой мыши) и называем **Snake**:



Чтобы проверить какая программа открывает скрипты нужно перейти в **Edit → Preferences → External Tools → External Script Editor:**



После нам надо перетащить скрипт на объект **Snake**:



15. Щёлкаем дважды на скрипт **Snake**, и он откроется в вашей IDE:

```
Snake.cs X
Assembly-CSharp Snake Start()
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class Snake : MonoBehaviour
6 {
7     // Start is called before the first frame update
8     void Start()
9     {
10     }
11
12     // Update is called once per frame
13     void Update()
14     {
15     }
16 }
17
18
19
```

Можно удалить неиспользуемые строки кода (подсвечиваются серыми), а также всё что находится внутри класса, т.к. обычно пишут с 0. В итоге скрипт будет выглядеть так:

```
Snake.cs X
Assembly-CSharp Snake
1 using UnityEngine;
2
3 public class Snake : MonoBehaviour
4 {
5 }
6
7
```

```
using UnityEngine; // Подключает пространство имен
UnityEngine, которое содержит основные классы и функции
Unity.
public class Snake: MonoBehaviour // Объявляет публичный
класс Snake, который наследуется от MonoBehaviour. Это
позволяет использовать методы и свойства Unity, такие как
Start, Update, и т.д.
```

16. Для передвижения нашей змейки нам нужно объявить переменную:

```
private Vector2 _direction; // Объявляет приватное поле
_direction типа Vector2
```

Затем нам нужно добавить метод Update:

```
private void Update() // Метод Update вызывается каждый
кадр.
{
}
```

Затем нам нужно задать направление движения, которые вводит пользователь:

```
private void Update() // Метод Update вызывается каждый
кадр.
{
    // Проверяет, нажата ли одна из клавиш W, S, A или
    D, и изменяет направление движения змейки соответственно.
```

```

        if (Input.GetKeyDown(KeyCode.W))
        {
            _direction = Vector2.up;
        }
        else if (Input.GetKeyDown(KeyCode.S))
        {
            _direction = Vector2.down;
        }
        else if (Input.GetKeyDown(KeyCode.A))
        {
            _direction = Vector2.left;
        }
        else if (Input.GetKeyDown(KeyCode.D))
        {
            _direction = Vector2.right;
        }
    }
}

```

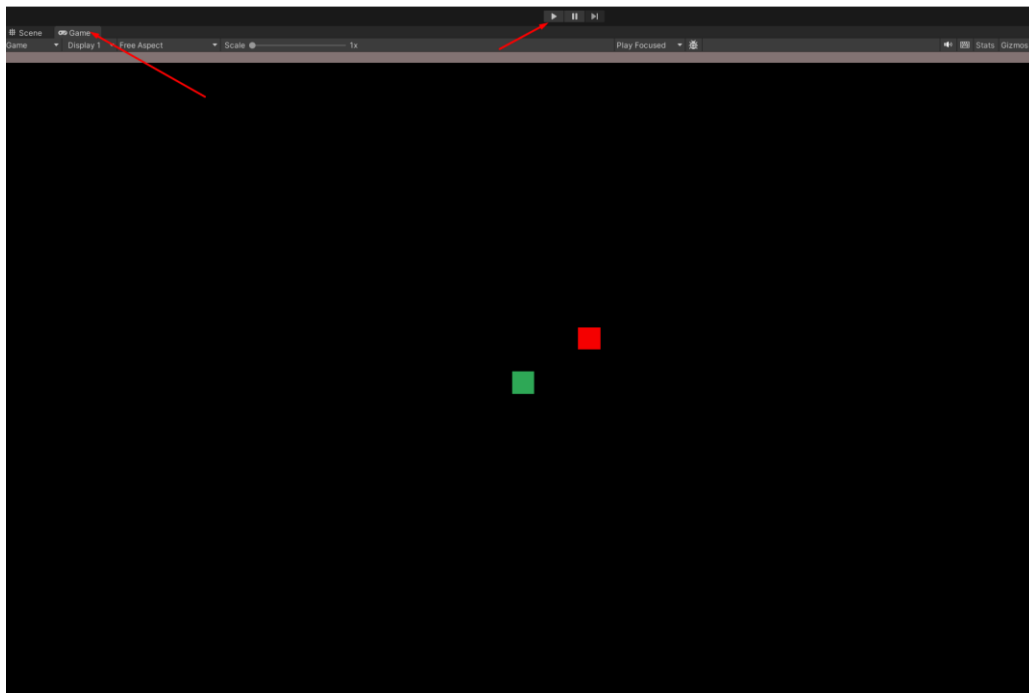
Теперь нам для изменения движения змейки нужно исходя из нашего направления изменить положение змейки. Это делается с помощью метода **FixedUpdate()**:

```

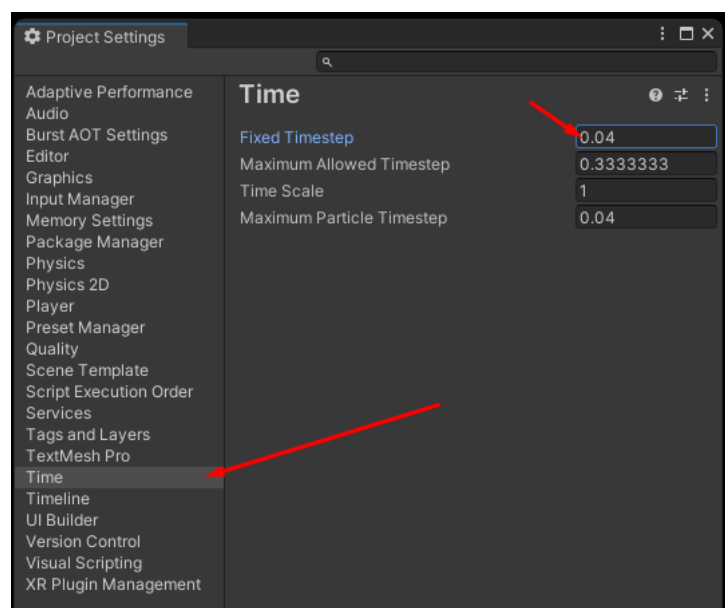
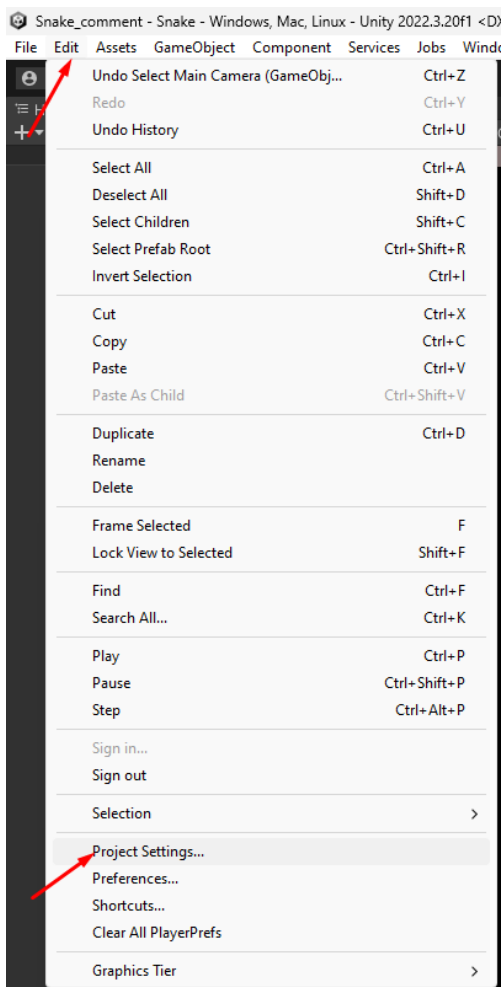
private void FixedUpdate() // Метод FixedUpdate
вызывается с фиксированной частотой.
{
    //Обновляет позицию головы змейки в соответствии с
    текущим направлением движения.
    // this.transform.position: получает текущую позицию
    объекта, к которому прикреплен этот скрипт.
    // new Vector3(...): создает новый вектор с тремя
    компонентами (x, y, z), который будет новой позицией
    объекта.
    // + _direction.x: добавляет значение x из вектора
    направления _direction к округленной координате x.
    // 0.0f: устанавливает координату z в 0.0, так как змейка
    движется только в 2D-плоскости.
    this.transform.position = new Vector3(
        Mathf.Round(this.transform.position.x) +
        _direction.x,
        Mathf.Round(this.transform.position.y) +
        _direction.y,
        0.0f);
}

```

16. Проверяем что всё работает в игре:

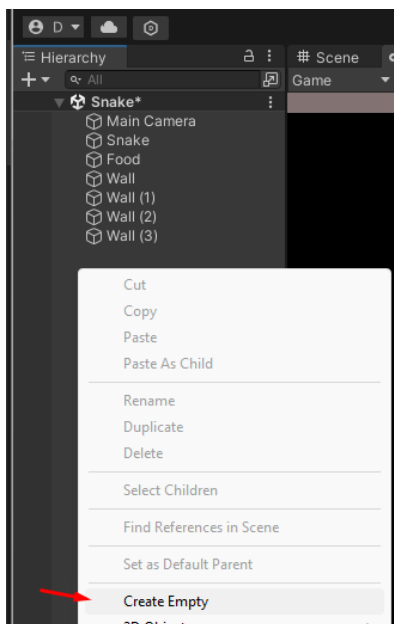


Давайте немногоотрегулируем скорость движения змейки. Переходим в **Edit** → **Project Settings** → **Time** → **Fixed Timestep**. Поменяем на 0.04:

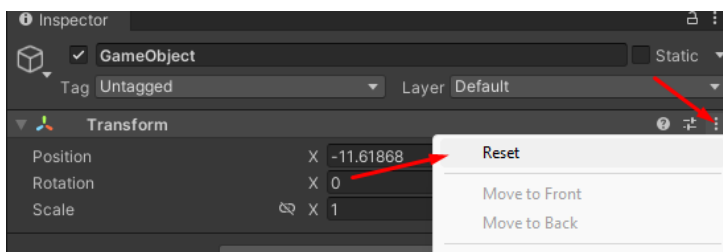


17. Теперь займёмся нашей едой. Чтобы она рандомно появлялась на нашем игровом поле, нам нужно создать с вами новый объект, который не будет выходить за пределы стен.

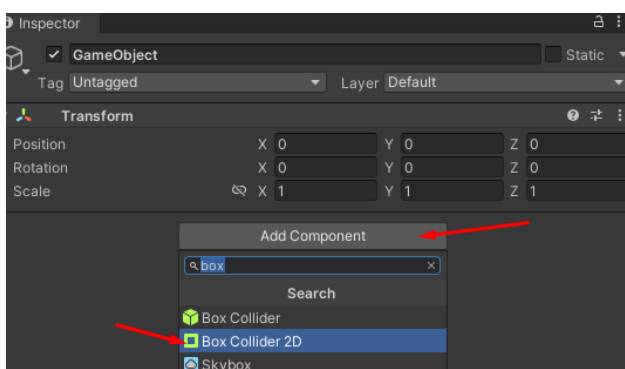
Нажимаем правой кнопкой мыши в нашей **Hierarchy** → **Create Empty**:



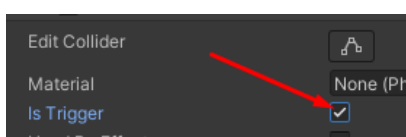
Reset:



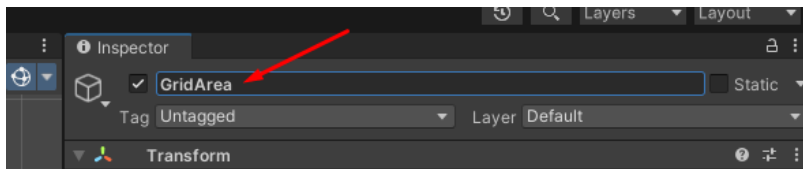
Добавляем **Box Collider 2D**:



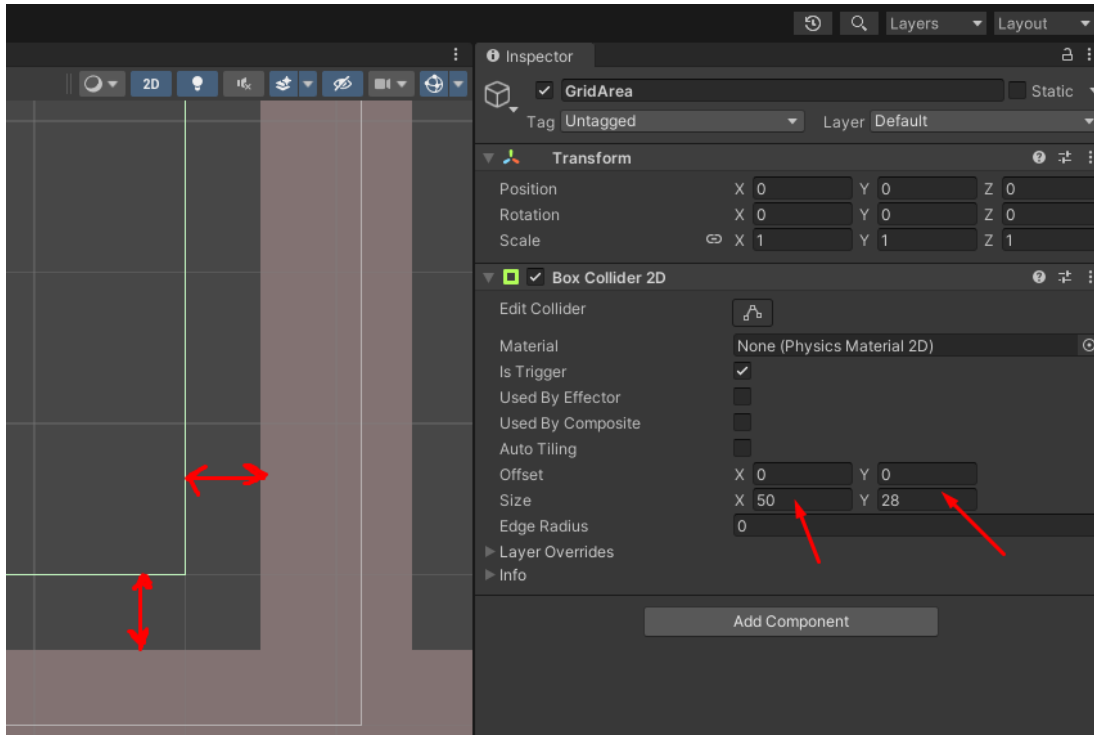
Поменяем **Is Trigger**:



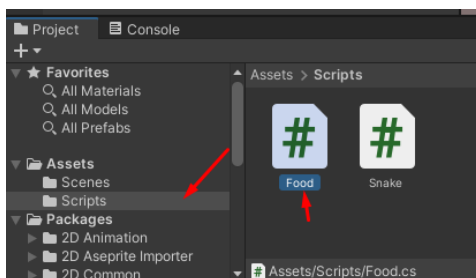
Называем **GridArea**:



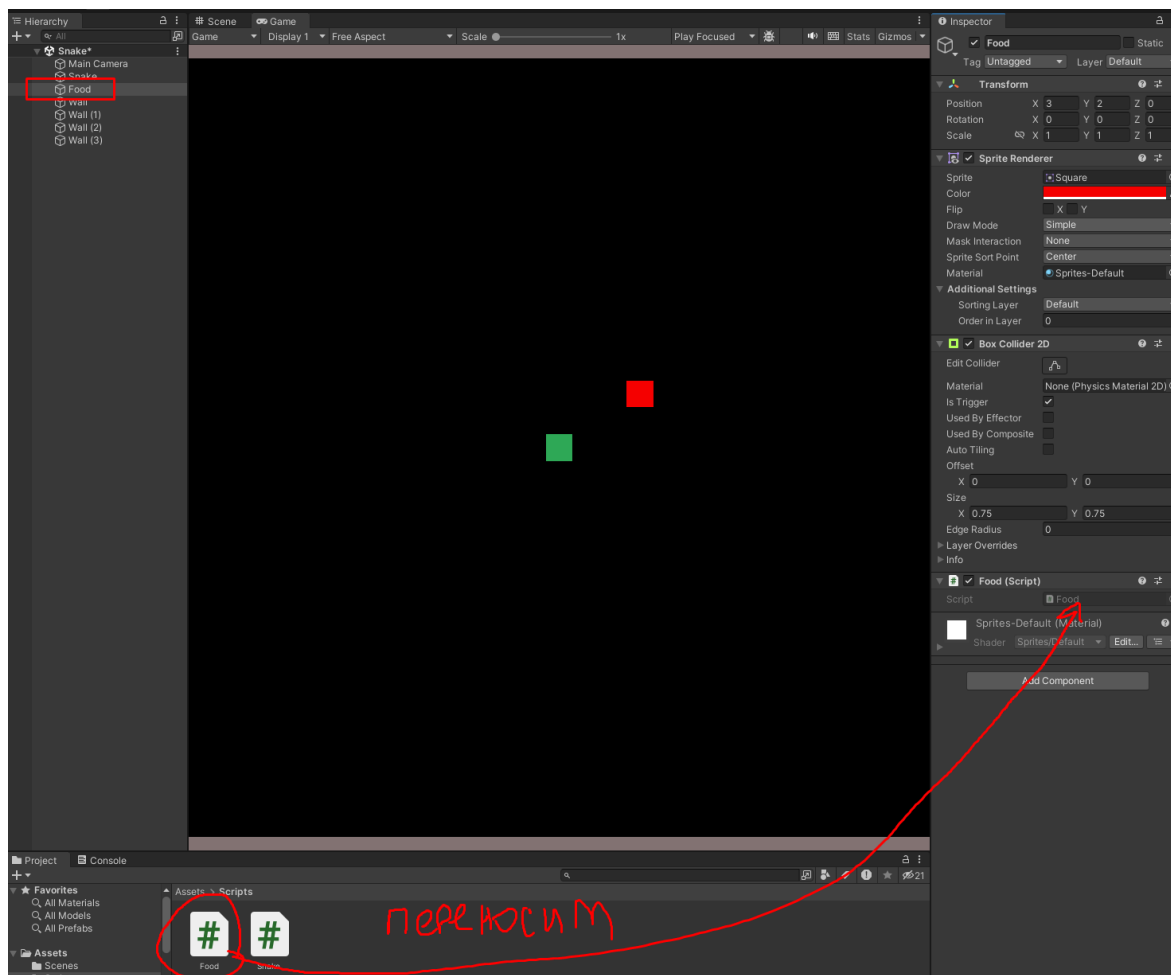
Настроим размер **Box Collider 2D** чтобы он не касался границ наших стен (в моём случае это $x = 50$ и $y = 28$):



18. Создаём скрипт с именем **Food**:

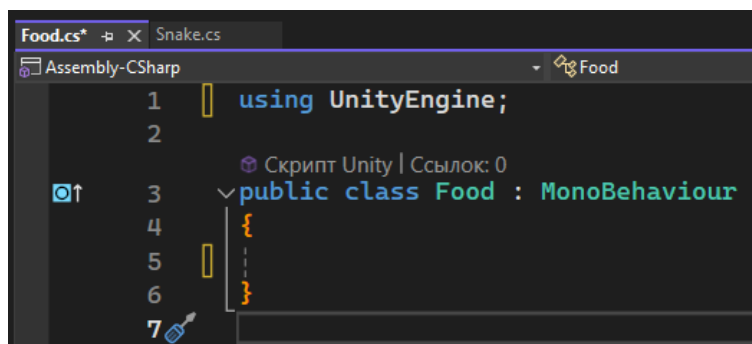


И добавляем его к еде (нажимаем на объект Food и переносим на него скрипт, чтобы он у нас отображался):



Открываем его.

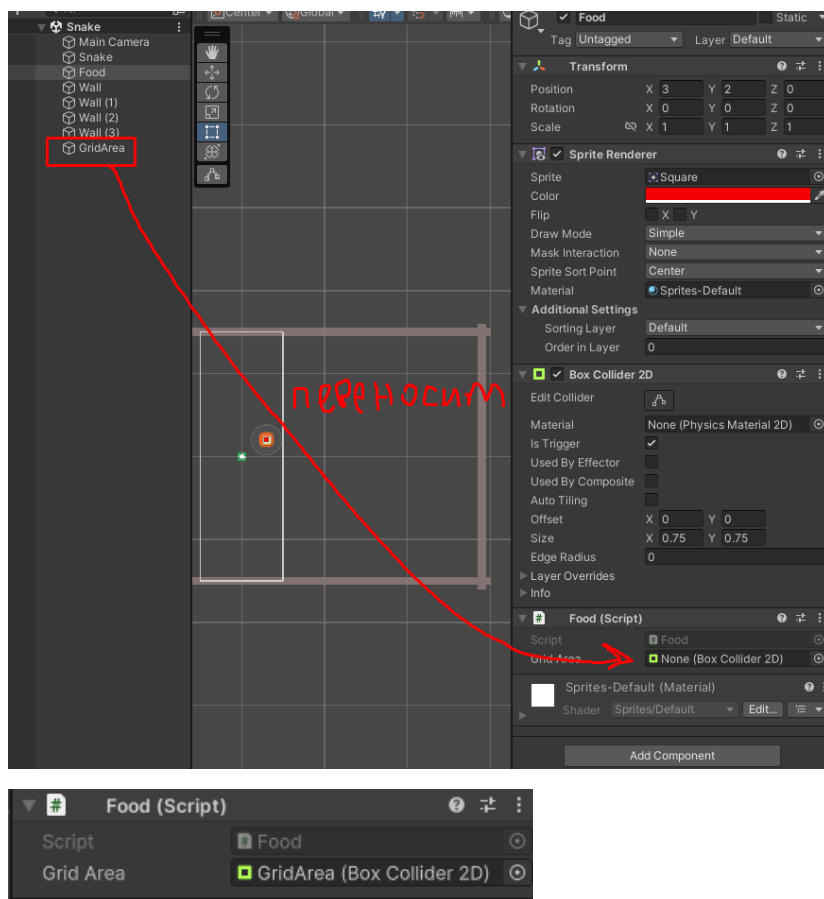
18. Удаляем ненужные строки кода:



Объявляем публичное поле **GridArea**:

```
public BoxCollider2D GridArea; // Объявляет публичное поле
GridArea типа BoxCollider2D. Это поле будет использоваться
для определения области, в пределах которой будет случайно
размещаться объект Food.
```

Затем нам нужно перенести объект **GridArea** в поле скрипта:



19. Напишем метод, который будет случайным образом задавать позицию нашей еды в пределах области **GridArea**:

```
private void RandomizePosition() // Метод, который
    случайным образом задает позицию объекта Food в пределах
    области GridArea.
{
    Bounds bounds = this.GridArea.bounds; // Получает
    границы области GridArea.

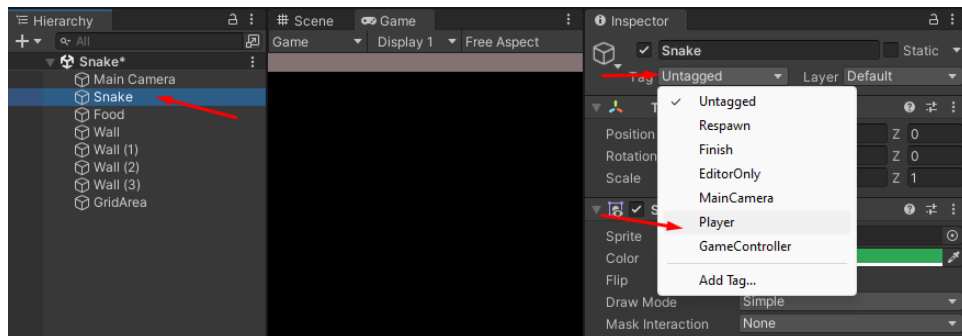
    float x = Random.Range(bounds.min.x, bounds.max.x);
    // Генерирует случайное значение x в пределах границ
    GridArea.
    float y = Random.Range(bounds.min.y, bounds.max.y);
    // Генерирует случайное значение y в пределах границ
    GridArea.

    this.transform.position = new
    Vector3(Mathf.Round(x), Mathf.Round(y), 0.0f); //
    Устанавливает позицию объекта Food на случайные координаты
    (x, y) с округлением до целых чисел.
}
```

Добавим вызов объекта, при начале игры:

```
private void Start() // Метод Start вызывается один раз
при инициализации объекта. В данном случае он вызывает
метод RandomizePosition, чтобы задать начальную позицию
объекта Food.
{
    RandomizePosition();
}
```

Сделаем так, чтобы при съедании еды она будет пропадать и появляться в новом месте. Поставим для змейки тег **Player** чтобы по нему сверять столкновение с едой:



Затем напомним код:

```
private void OnTriggerEnter2D(Collider2D other) // Метод,
который вызывается при столкновении объекта Food с другим
объектом.
{
    if (other.tag == "Player") // Проверяет, имеет ли
    объект, с которым произошло столкновение, тег "Player".
    {
        RandomizePosition(); // Если условие выполнено,
        вызывается метод RandomizePosition, чтобы переместить
        объект Food на новую случайную позицию.
    }
}
```

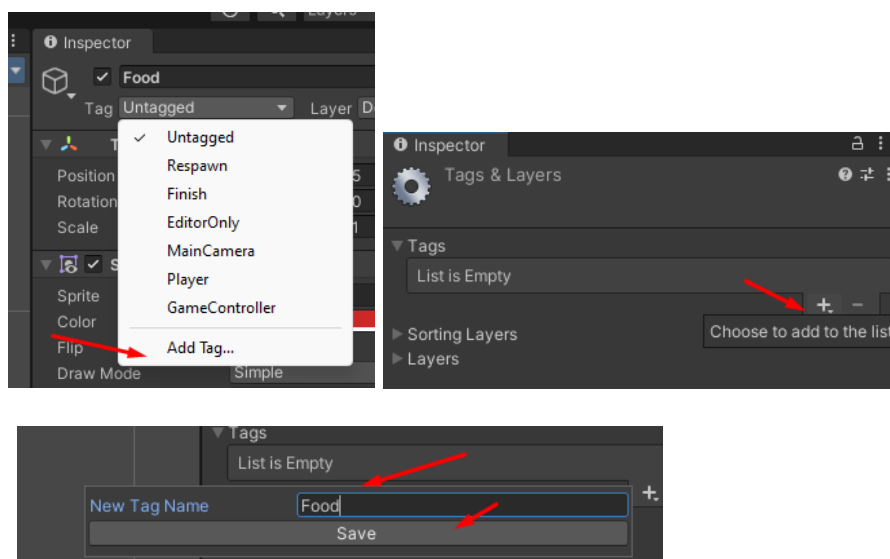
В итоге полностью готовый скрипт на еду будет выглядеть следующим образом:

```

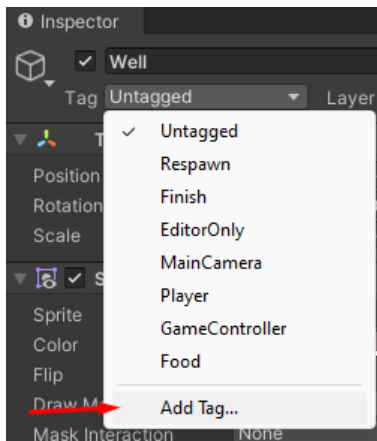
1  using UnityEngine;
2
3  Скрипт Unity (1 ссылка на ресурсы) | Ссылок: 0
4  public class Food : MonoBehaviour
5  {
6      public BoxCollider2D GridArea;
7
8      Ссылка: 2
9      private void RandomizePosition()
10     {
11         Bounds bounds = this.GridArea.bounds;
12
13         float x = Random.Range(bounds.min.x, bounds.max.x);
14         float y = Random.Range(bounds.min.y, bounds.max.y);
15
16         this.transform.position = new Vector3(Mathf.Round(x),
17             Mathf.Round(y), 0.0f);
18     }
19
20     Сообщение Unity | Ссылка: 0
21     private void Start()
22     {
23         RandomizePosition();
24     }
25
26     Сообщение Unity | Ссылка: 0
27     private void OnTriggerEnter2D(Collider2D other)
28     {
29         if (other.tag == "Player")
30         {
31             RandomizePosition();
32         }
33     }
34 }

```

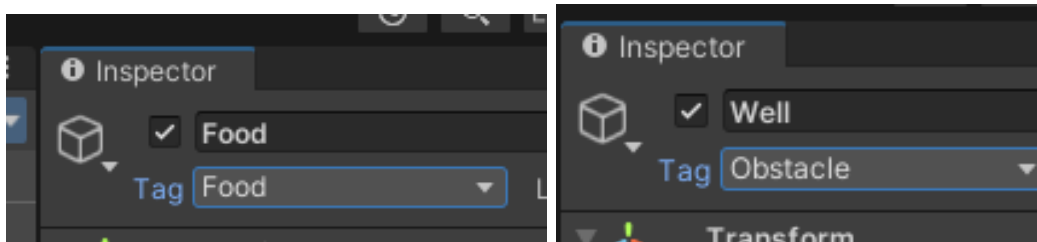
20. Добавим новый тег **Food** для нашего объекта – еды:



Добавим новый тег **Obstacle** для наших объектов – препятствий:

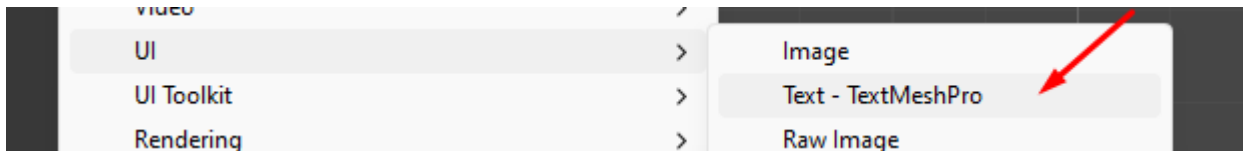


И не забудем их выбрать (для каждой стены и для **Food**)!

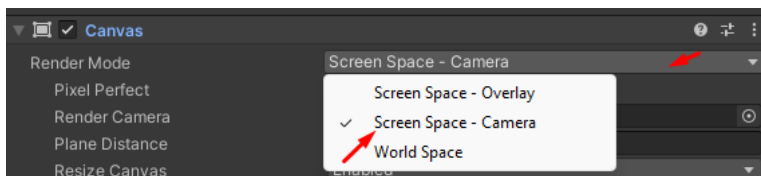


21. Теперь добавим отображение очков в нашу игру.

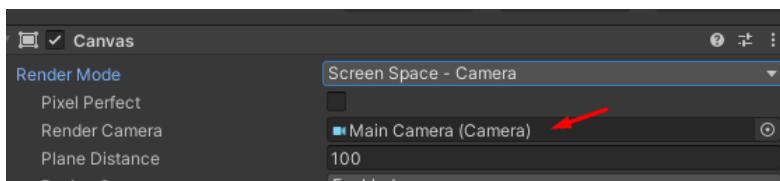
Щёлкаем правой кнопкой мыши в иерархии и добавляем **UI – Text – TextMeshPro**:



В **Inspector** для **Canvas** выбираем в модели рендера **Screen Space-Camera**:



Переносим **Main Camera** в рендер:



Подгоняем размер текста, под нашу игровую область. Меняем размер шрифта, цвет и текст на **Score**:



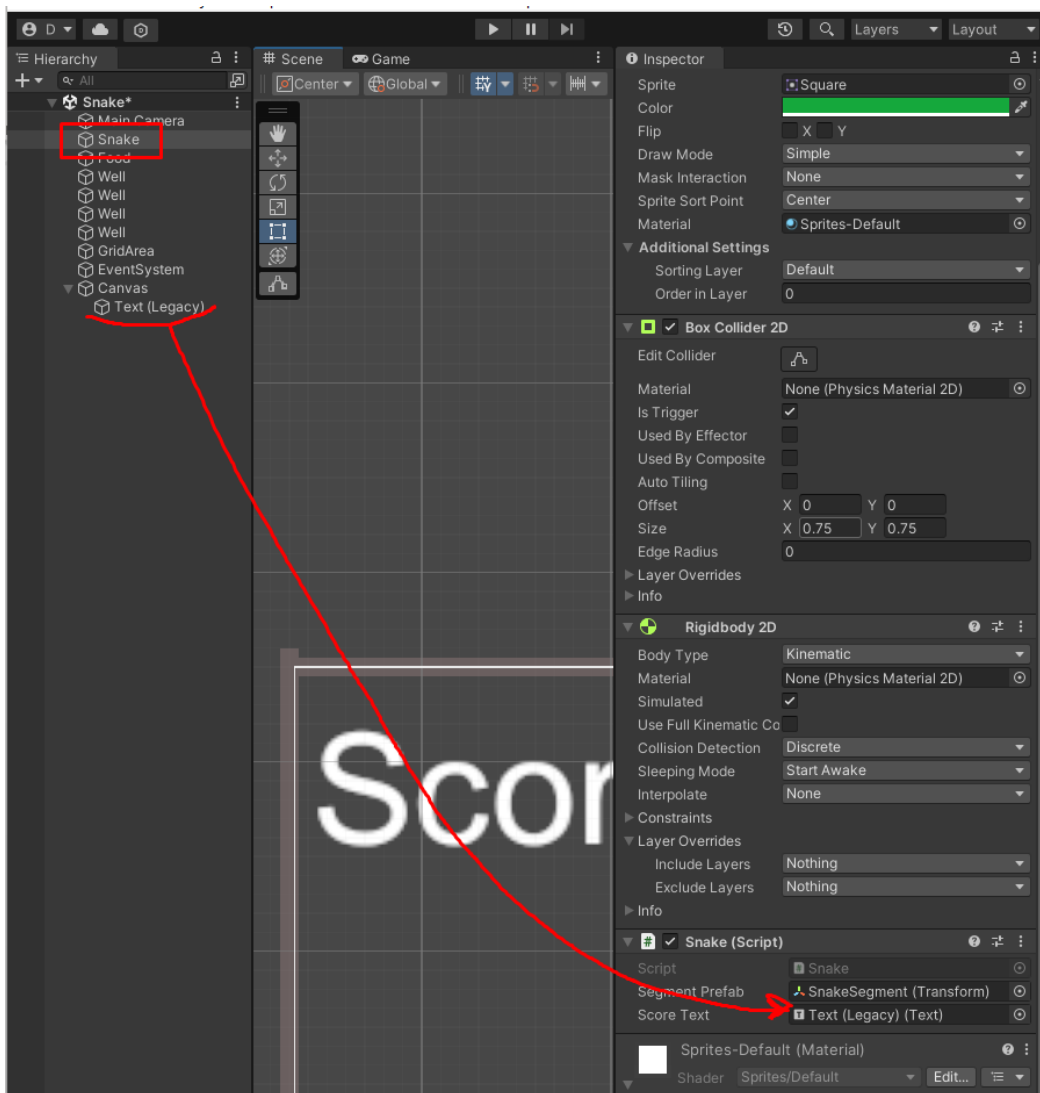
22. Переходим в скрипт **Snake**. Подключаем пространство имен для работы с **UI**:

```
using TMPro;
```

Объявим переменную и добавим ссылку на UI:

```
public TextMeshProUGUI scoreText;  
private int score = 0;
```

Затем перенесём в **Unity** в переменную **Score Text** наш текст:



Пропишем в самом конце новый метод **UpdateScoreText**:

```
private void UpdateScoreText() // обновляет текстовое поле  
scoreText для отображения текущего счета  
{  
    scoreText.text = "Score: " + score.ToString();  
}
```

23. Напишем метод для сброса игры:

```
private void ResetGame() // метод для сброса игры
{
    transform.position = Vector3.zero; // устанавливаем
    координаты змейки в начало
    score = 0; // обнуляем счёт
    UpdateTextScore(); // запускаем метод подсчёта и вывода
    очков
}
```

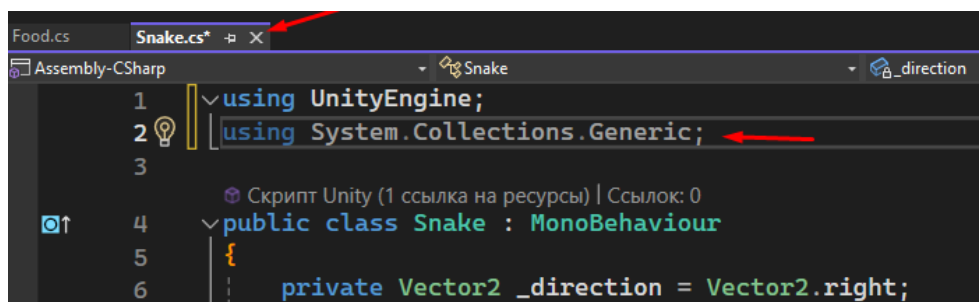
И пропишем метод для проверки столкновения с тегом еды и препятствиями:

```
private void OnTriggerEnter2D(Collider2D other)
{
    if (other.CompareTag("Food"))
    {
        score++;
        UpdateTextScore();
    }
    else if (other.CompareTag("obstacle"))
        ResetGame();
}
```

24. Теперь сделаем увеличение сегментов змейки. Для этого возвращаемся в скрипт со змейкой (**Snake.cs**)

Для начала подключим коллекции в начале скрипта:

```
using System.Collections.Generic; // Подключает
пространство имен, которое содержит коллекции, такие как
списки (List).
```



Создаём приватное поле для наших сегментов:

```
private List<Transform> _segments; // Объявляет приватное
поле _segments типа List<Transform>, которое будет
содержать сегменты змейки.
```

```

public class Player : MonoBehaviour
{
    private Vector2 _direction;
    public TextMeshProUGUI scoreText;
    private int score = 0;
    private List<Transform> _segments;

    void Update()

```

Создадим метод **Start**, вызывающийся при запуске игры. Инициализируем список и добавим первый сегмент (голову змейки):

```

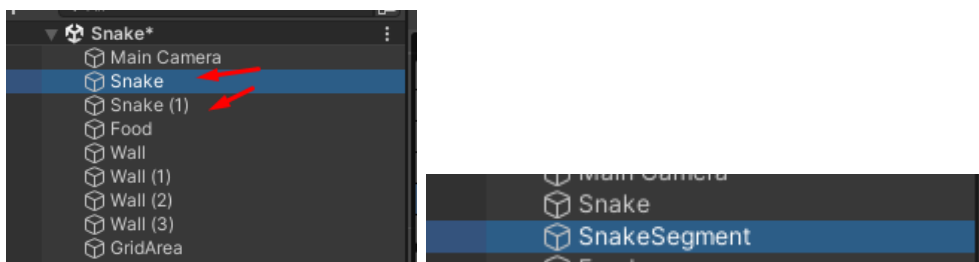
private void Start() // Метод Start вызывается один раз
при инициализации объекта.
{
    _segments = new List<Transform>(); //
Инициализирует список _segments.
    _segments.Add(this.transform); // Добавляет первый
сегмент (голову змейки) в список _segments.
}

```

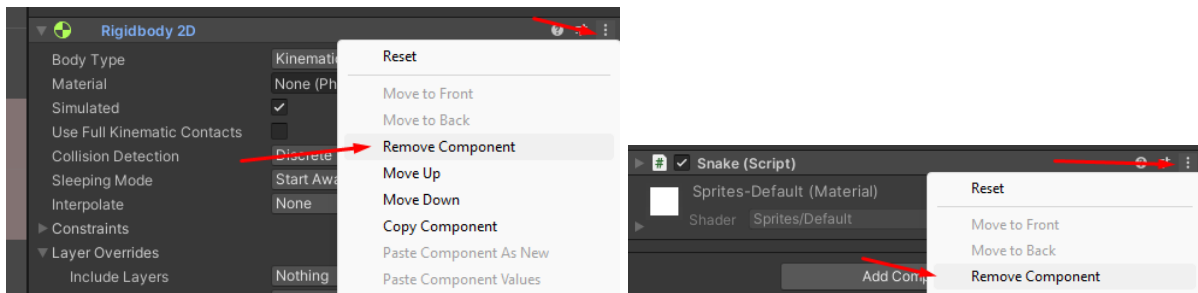
25. Теперь нам нужно создать новый объект – **Prefab**.

Prefab - это шаблон для объекта в игровом движке Unity. С помощью префабов можно создать «образец» предмета с определенными свойствами, а потом использовать такие предметы на всей игровой сцене. Если изменить префаб, то изменятся все объекты, созданные на его основе.

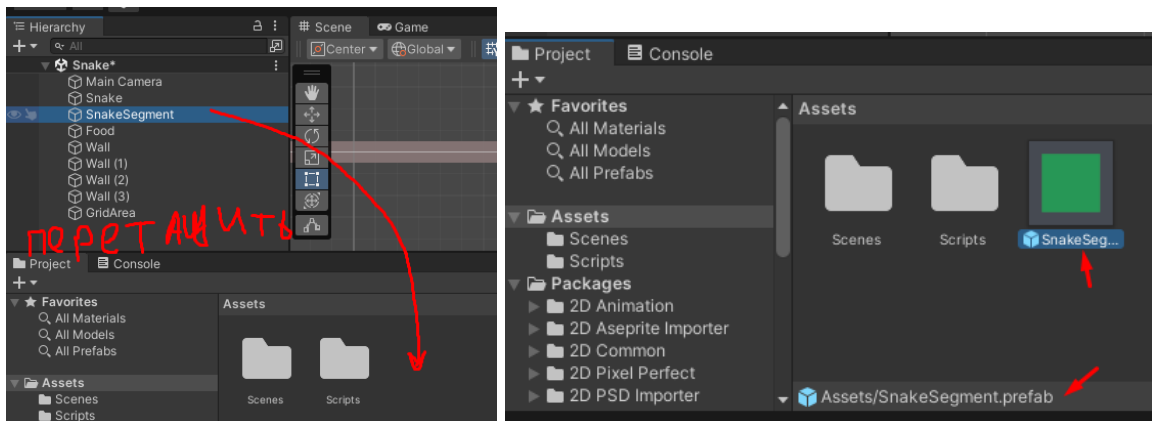
Для этого выбираем наш объект **Snake** и нажимаем **Ctrl+D**, чтобы сделать его дубликат и называем его **SnakeSegment**:



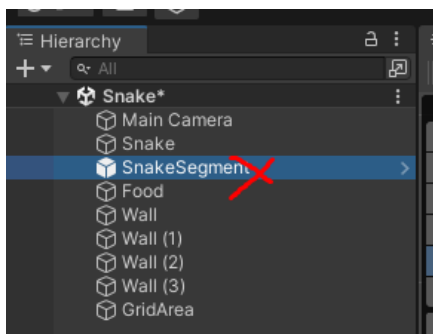
У **SnakeSegment** удаляем **Rigidbody 2D** и скрипт:



Перетаскиваем **SnakeSegment** в наши **Assets** (в итоге он появится как новый тип объекта и будет иметь расширение **.prefab**):



После удаляем объект из нашей иерархии:

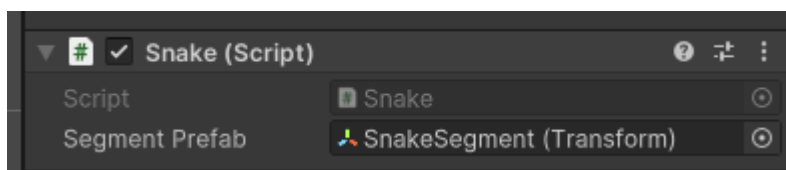
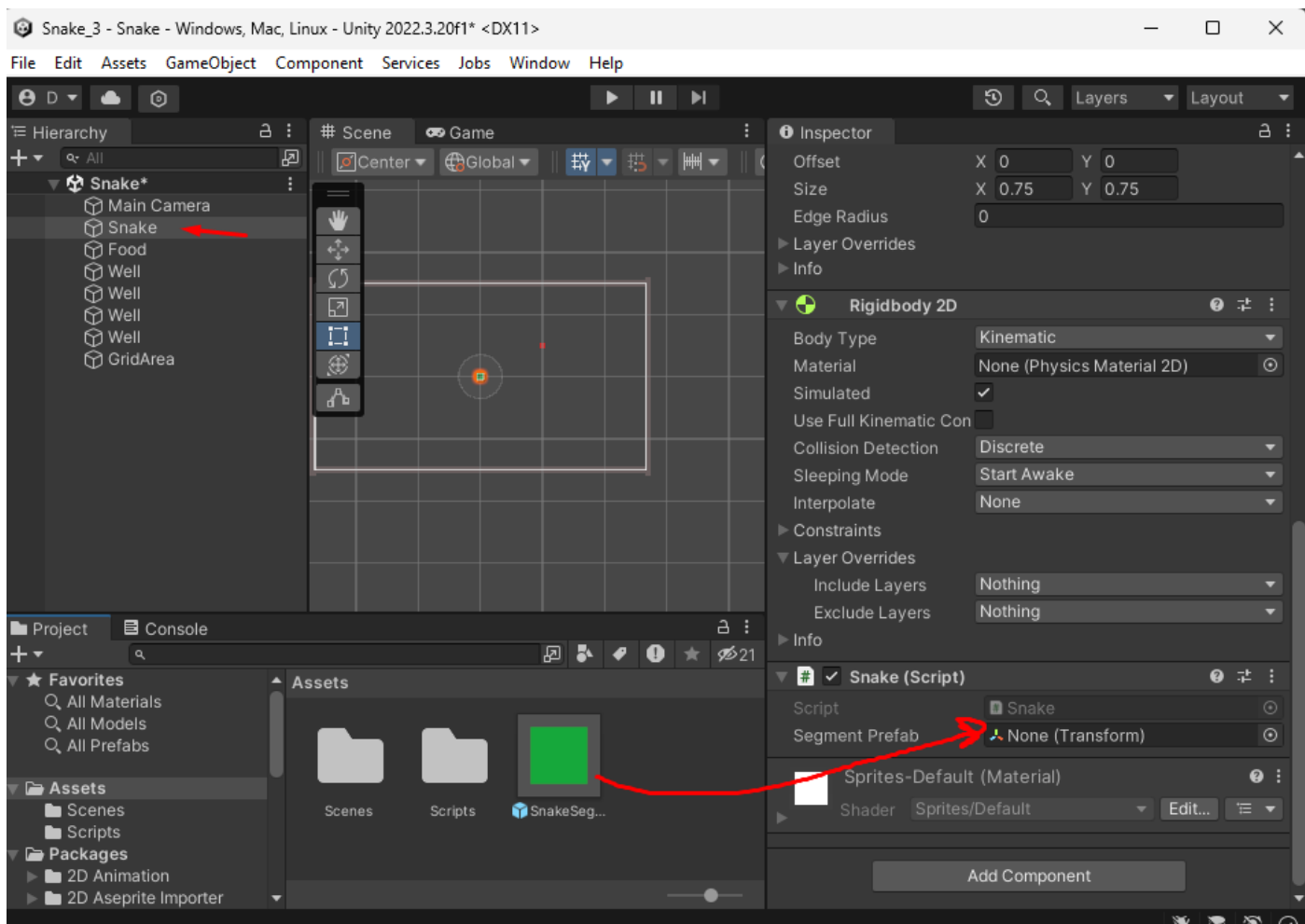


26. Возвращаемся в наш скрипт. Создадим публичное поле для нашего префаба:

```
public Transform segmentPrefab; // Объявляет публичное поле
segmentPrefab типа Transform, которое будет использоваться
для создания новых сегментов змейки.
```

```
private Vector2 _direction;
public TextMeshProUGUI scoreText;
private int score = 0;
private List<Transform> _segments;
public Transform segmentPrefab;
```

Перетаскиваем наш префаб в поле **Snake**:



Далее создадим новый метод **Grow**, который будет добавлять сегменты к змейке:

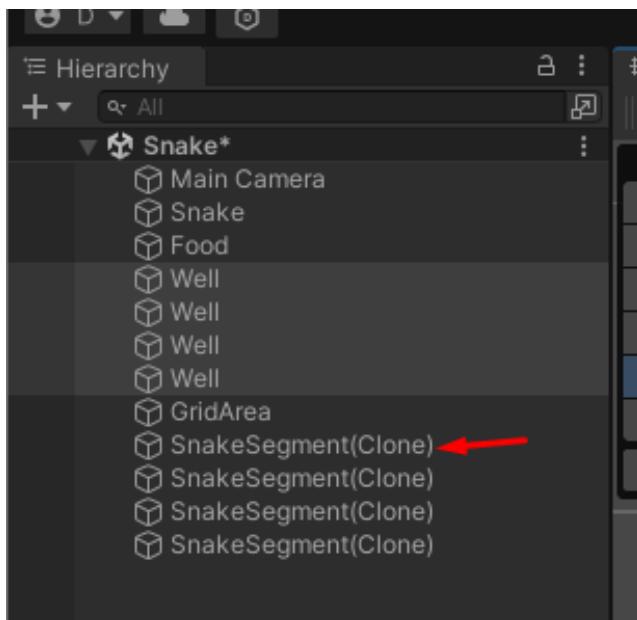
```
private void Grow() // Метод, который добавляет новый
сегмент к змейке.
{
    Transform segmentNew =
Instantiate(this.segmentPrefab); // Создает новый сегмент.
    segmentNew.position = _segments[_segments.Count -
1].position; // Устанавливает позицию нового сегмента на
позицию последнего сегмента змейки.

    _segments.Add(segmentNew); // Добавляет новый
сегмент в список _segments.
}
```

27. Затем добавим в метод **OnTriggerEnter2D** вызов метода **Grow**:

```
private void OnTriggerEnter2D(Collider2D other)
{
    if (other.CompareTag("Food"))
    {
        Grow();
        score++;
        UpdateTextScore();
    }
    else if (other.CompareTag("obstacle"))
        ResetGame();
}
```

28. Если запустить игру, то может показаться странным, что сегменты вроде как не добавляются, но на самом деле они появляются, в этом можно убедиться при столкновении с едой глянув в иерархию мы увидим новые префабы:



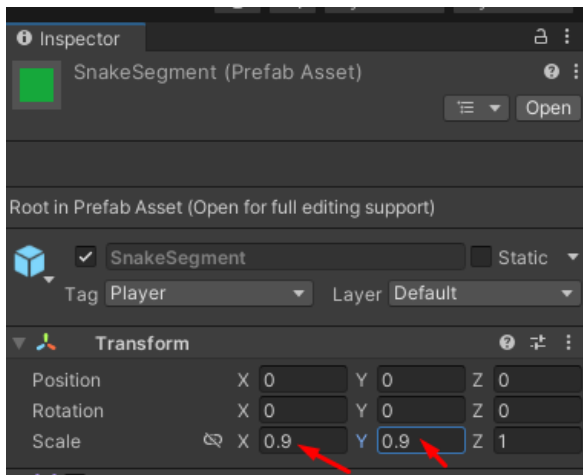
Решим эту проблему, для этого добавим цикл перемещения сегментов змейки на позицию предыдущего в начало метода **FixedUpdate**:

```
for (int i = _segments.Count - 1; i > 0; i--) // Цикл,
    который перемещает каждый сегмент змейки на позицию
    предыдущего сегмента.
{
    _segments[i].position = _segments[i -
1].position;
}
```

```
private void FixedUpdate()
{
    for (int i = _segments.Count - 1; i > 0; i--)
        _segments[i].position = _segments[i - 1].position;

    transform.position = new Vector3(
        transform.position.x + _direction.x,
        transform.position.y + _direction.y,
        0.0f);
}
```

Для лучшей визуализации, поменяем у префаба размеры на **0.9**:



29. Теперь нам нужно дописать метод **ResetGame**:

```
private void ResetGame() // Метод, который сбрасывает
игру.
{
    // Цикл, который уничтожает все сегменты змейки,
    кроме головы.
    for (int i = 1; i < _segments.Count; i++)
    {
        Destroy(_segments[i].gameObject);
    }

    _segments.Clear(); // Очищает список _segments.
    _segments.Add(this.transform); // Добавляет голову
    змейки в список _segments.

    this.transform.position = Vector3.zero;
    //Устанавливает позицию головы змейки в начало координат.
}
```

Затем добавим в наш метод **OnTriggerEnter2D** условие для вызова метода:

```

private void OnTriggerEnter2D(Collider2D other) //
Метод, который вызывается при столкновении с другим
объектом.
{
    // Если объект имеет тег "Food", вызывается метод
Grow.
    if (other.tag == "Food")
    {
        Grow();
    }
    // Если объект имеет тег "Obstacle", вызывается
метод ResetGame.
    else if (other.tag == "Obstacle")
    {
        ResetGame();
    }
}

```

Рефакторинг

30. Теперь решим проблему возможности змейки двигаться в противоположном направлении. Для этого немного изменим код в методе **Update**:

```

private void Update()
{
    if (Input.GetKeyDown(KeyCode.W) && _direction !=
Vector2.down)
        _direction = Vector2.up;
    else if (Input.GetKeyDown(KeyCode.S) && _direction !=
Vector2.up)
        _direction = Vector2.down;
    else if (Input.GetKeyDown(KeyCode.A) && _direction !=
Vector2.right)
        _direction = Vector2.left;
    else if (Input.GetKeyDown(KeyCode.D) && _direction !=
Vector2.left)
        _direction = Vector2.right;
}

```

31. При столкновении змейки с самой собой, будем вызывать перезагрузку уровня. Для этого добавим в **FixedUpdate** проверку:

```

private void FixedUpdate()
{
    for (int i = _segments.Count - 1; i > 0; i--)
        _segments[i].position = _segments[i - 1].position;

    transform.position = new Vector3(
        transform.position.x + _direction.x,
        transform.position.y + _direction.y,
        0.0f);

    // Проверка на столкновение с сегментами змейки
    for (int i = 1; i < _segments.Count; i++)
    {
        if (transform.position == _segments[i].position)
        {
            ResetGame();
            break;
        }
    }
}

```

32. Давайте добавим возможности выходить из игры по нажатию клавиши **ESC**.

Для этого допишем в методе **Update**:

```

else if (Input.GetKeyDown(KeyCode.Escape))
    Application.Quit();

```

```

private void Update()
{
    if (Input.GetKeyDown(KeyCode.W) && _direction != Vector2.down)
        _direction = Vector2.up;
    else if (Input.GetKeyDown(KeyCode.S) && _direction != Vector2.up)
        _direction = Vector2.down;
    else if (Input.GetKeyDown(KeyCode.A) && _direction != Vector2.right)
        _direction = Vector2.left;
    else if (Input.GetKeyDown(KeyCode.D) && _direction != Vector2.left)
        _direction = Vector2.right;
    else if (Input.GetKeyDown(KeyCode.Escape))
        Application.Quit();
}

```

33. Разделим в методе **FixedUpdate** код на 3 метода:

```

private void FixedUpdate()
{
    MoveSegments();
    MovePlayer();
    CheckSelfCollision();
}

```

Сами методы:

Ссылка: 1

```
private void MoveSegments()
{
    for (int i = _segments.Count - 1; i > 0; i--)
    {
        _segments[i].position = _segments[i - 1].position;
    }
}
```

Ссылка: 1

```
private void MovePlayer()
{
    transform.position = new Vector2(
        Mathf.Round(transform.position.x) + _direction.x,
        Mathf.Round(transform.position.y) + _direction.y
    );
}
```

Ссылка: 1

```
private void CheckSelfCollision()
{
    for (int i = 1; i < _segments.Count; i++)
    {
        if (transform.position == _segments[i].position)
        {
            ResetGame();
            break;
        }
    }
}
```

34. Можно переписать обработку ввода через конструкцию **switch**, и вынести её отдельным методом:

```
void Update()
{
    HandleInput();
}
```

Код:

```
private void HandleInput()
{
    switch (Input.inputString)
    {
        case "a" or "φ" when _direction != Vector2.right:
            _direction = Vector2.left; break;
        case "d" or "В" when _direction != Vector2.left:
            _direction = Vector2.right; break;
        case "w" or "Ц" when _direction != Vector2.down:
            _direction = Vector2.up; break;
        case "s" or "ы" when _direction != Vector2.up:
            _direction = Vector2.down; break;
        // Escape key
        case "\u001B": Application.Quit(); break;
    }
}
```

35. Для увеличения скорости змейки по мере набора очков мы можем увеличить параметр **Time.fixedDeltaTime** в Unity. Этот параметр управляет тем, как часто вызывается **FixedUpdate** и тем самым влияет на скорость игры.

Добавляем инициализацию начальной скорости для **fixedDeltaTime** и переменную для увеличения скорости:

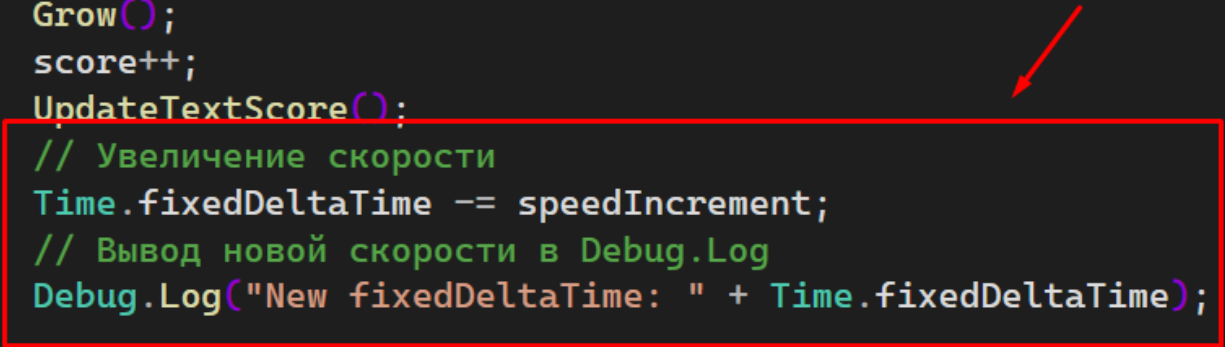
```
Скрипт Unity (1 ссылка на ресурсы) | Ссылок: 0
public class Player : MonoBehaviour
{
    private float initialFixedDeltaTime;
    private float speedIncrement = 0.001f; // Насколько увеличивать
    скорость за каждую еду
}
```

Сохраняем начальное значение **Time.fixedDeltaTime**, чтобы можно было сбросить его при перезапуске игры:

```
Сообщение Unity | Ссылок: 0
private void Start()
{
    _segments = new List<Transform>();
    segments.Add(transform);
    initialFixedDeltaTime = Time.fixedDeltaTime;
}
```

Увеличиваем скорость в **OnTriggerEnter2D** при сборе еды:

```
private void OnTriggerEnter2D(Collider2D other)
{
    if (other.CompareTag("Food"))
    {
        Grow();
        score++;
        UpdateTextScore();
        // Увеличение скорости
        Time.fixedDeltaTime -= speedIncrement;
        // Вывод новой скорости в Debug.Log
        Debug.Log("New fixedDeltaTime: " + Time.fixedDeltaTime);
    }
    else if (other.CompareTag("obstacle"))
        ResetGame();
}
```

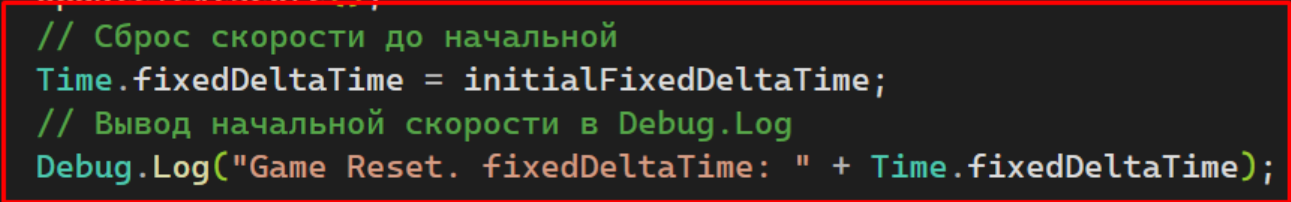


Сбрасываем скорость до начального значения при перезапуске игры в **ResetGame**:

```
private void ResetGame()
{
    for (int i = 1; i < _segments.Count; i++)
    {
        Destroy(_segments[i].gameObject);
    }
    _segments.Clear();
    _segments.Add(transform);

    transform.position = Vector3.zero;

    score = 0;
    UpdateTextScore();
    // Сброс скорости до начальной
    Time.fixedDeltaTime = initialFixedDeltaTime;
    // Вывод начальной скорости в Debug.Log
    Debug.Log("Game Reset. fixedDeltaTime: " + Time.fixedDeltaTime);
}
```



36. Бонус! Выход за пределы экрана.

Чтобы сделать так, чтобы объекты, выходящие за границы экрана, появлялись с противоположной стороны, можно использовать функцию ограничения координат в пределах экрана. Сделать это можно, проверяя позицию игрока в методе **FixedUpdate** и изменяя её, если она выходит за пределы экрана.

Напишем метод **WrapAroundScreen**, он будет проверять, выходит ли змейка за границы экрана, и перемещает её на противоположную сторону, если это происходит:

```
private void WrapAroundScreen()
{
    Vector3 position = transform.position;
    float screenWidth = Camera.main.orthographicSize *
Camera.main.aspect;
    float screenHeight = Camera.main.orthographicSize;

    if (position.x > screenWidth) position.x = -
screenWidth;
    else if (position.x < -screenWidth) position.x =
screenWidth;

    if (position.y > screenHeight) position.y = -
screenHeight;
    else if (position.y < -screenHeight) position.y =
screenHeight;

    transform.position = position;
}
```

Добавим его вызов в **FixedUpdate**:

```
private void FixedUpdate()
{
    MoveSegments();
    MovePlayer();
    CheckSelfCollision();
    WrapAroundScreen();
}
```

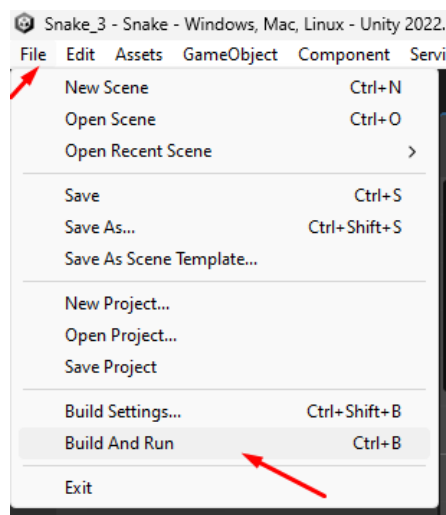
Удалим проверку на столкновение со стенами:

```

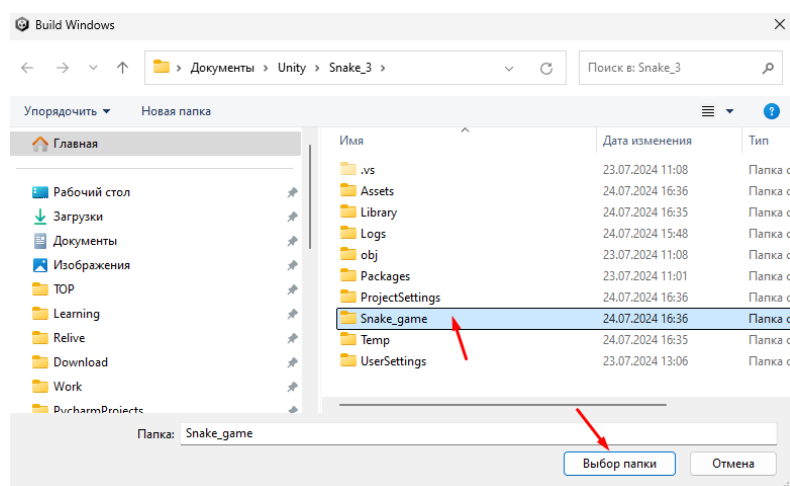
private void OnTriggerEnter2D(Collider2D other)
{
    if (other.CompareTag("Food"))
    {
        Grow();
        score++;
        UpdateTextScore();
        // Увеличение скорости
        Time.fixedDeltaTime -= speedIncrement;
        // Вывод новой скорости в Debug.Log
        Debug.Log("New fixedDeltaTime: " + Time.fixedDeltaTime);
    }
    else if (other.CompareTag("obstacle"))
        ResetGame();
}

```

35. Осталось только скомпилировать нашу игру. Переходим в **File – Build And Run**:



Выбираете любую папку куда хотите сохранить игру, или создаёте новую папку:



После можете запустить игру через .exe:

Snake_game

×

+

> Документы > Unity > Snake_3 > Snake_game >

Сортировать

Просмотреть

...

Имя	Дата изменения	Тип	Размер
<div>Папка</div> MonoBleedingEdge	24.07.2024 16:37	Папка с файлами	
<div>Папка</div> Snake_3_BurstDebugInformation_DoNot...	24.07.2024 16:37	Папка с файлами	
<div>Папка</div> Snake_3_Data	24.07.2024 16:37	Папка с файлами	
<div>Иконка приложения</div> Snake_3.exe	24.07.2024 16:37	Приложение	651 КБ
<div>Иконка приложения</div> UnityCrashHandler64.exe	24.07.2024 16:37	Приложение	1 089 КБ
<div>Иконка расширения</div> UnityPlayer.dll	24.07.2024 16:37	Расширение при...	30 251 КБ