

C#

# Переменные

```
public int firstNumber = 3;  
private float secondNumber;
```

# Типы данных

- Int
- Float
- Bool
- String

# Целочисленные типы данных

Тип	Разрядность в битах	Диапазон
<u>byte</u>	8	0...255
<u>sbyte</u>	8	-128...127
<u>short</u>	16	-32768...32767
<u>ushort</u>	16	0...65535
<u>int</u>	32	-2 147 483 648...2 147 483 647
<u>uint</u>	32	0...4 294 967 295
<u>long</u>	64	-9 223 372 036 854 775 808... 9 223 372 036 854 775 807
<u>ulong</u>	64	0...18 446 744 073 709 551 615

# Вещественные типы данных

Тип данных	Разрядность в байтах	Диапазон
<b>float</b>	32	$5 \cdot 10^{-45} \dots 3,4 \cdot 10^{+38}$
<b>double</b>	64	$5 \cdot 10^{-324} \dots$ $1,7 \cdot 10^{+308}$

Десятичный тип данных

**decimal** предназначен для применения в финансовых расчетах

разрядность 128 бит  $1 \cdot 10^{-28} \dots 7,9 \cdot 10^{+28}$ .

позволяет представить числа с точностью до 28 (а иногда и 29) десятичных разрядов

# Задание значения переменной при описании

```
public int firstNumber = 3;  
private float secondNumber = 0x123a;
```

# Логический тип данных

**bool** принимает значения **true** и **false**

Можно написать

```
if (e=true) {}
```

А можно

```
if e {}
```

# Подпрограммы

```
void Start () {  
  ... }  
}
```

```
void Update () {  
  ... }  
}
```



# Условный оператор

```
if ( условие ) {
```

```
}
```

```
else {
```

```
}
```

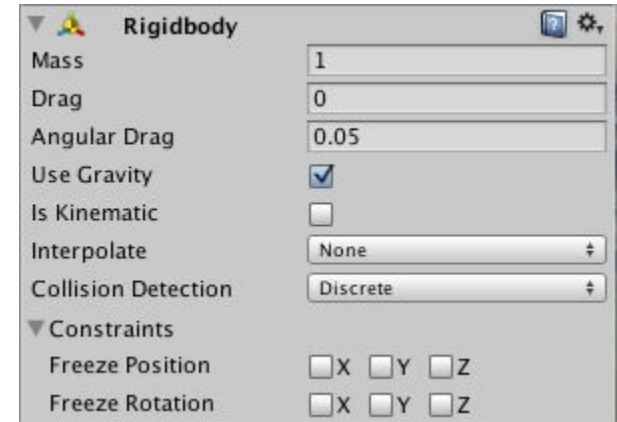
```
if transform.position.x > 10 {
```

```
Destroy(gameObject);
```

```
}
```

# Rigidbody (Твёрдое тело)

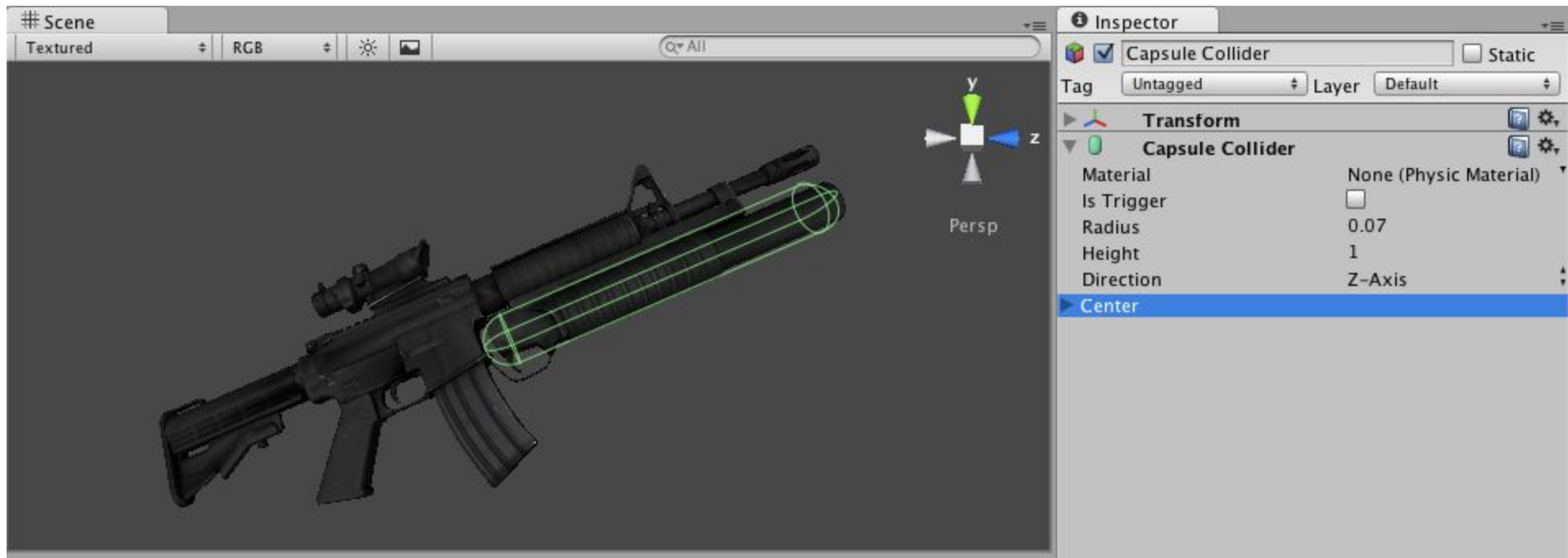
- Rigidbodies позволяют вашим GameObjects действовать под контролем физики.



<b>Mass</b>	Масса
<b>Drag</b>	Какое воздушное сопротивление оказывается на объект пока он перемещается под воздействием этих сил. 0 означает отсутствие сопротивления, а бесконечность (infinity) тут же прекращает перемещение объекта.
<b>Angular Drag</b>	Какое воздушное сопротивление оказывается на объект пока он вращается под воздействием силы вращения. 0 означает отсутствие сопротивления.
<b>Use Gravity</b>	При включении на объект действует гравитация.
<b>Is Kinematic</b>	Если включено, объект не будет управляться физическим движком, но может управляться только изменением Transform.
<b>Constraints</b>	Ограничения движения твёрдого тела:-
- <b>Freeze Position</b>	Выборочно останавливает перемещение твёрдого тела по осям X, Y и Z.
- <b>Freeze Rotation</b>	Выборочно останавливает вращение твёрдого тела по осям X, Y и Z.

# Коллайдеры

- тип компонентов, которые должны быть добавлены наряду с твёрдыми телами, чтобы задействовать столкновения. Если два твёрдых тела врезаются друг в друга, физический движок не будет просчитывать столкновение, пока к обоим объектам не будет назначен коллайдер. **Твёрдые тела не имеющие коллайдеров будут просто проходить сквозь друг друга при расчёте столкновений.**
- коллайдеры определяют физические границы твёрдого тела



# О чем мы говорили сегодня?

- <https://quizizz.com/admin/quiz/5c9ef6255999a001bd6e186/>

# Стрельба путем бросания лучей

- public class **Scr3** : MonoBehaviour {
- **private Camera \_camera;**
- void Start () {
- **\_camera = GetComponent<Camera>();**
- }
- void Update () {
- **if (Input.GetMouseButtonDown(0)) {**
- **Vector3 point = new Vector3(\_camera.pixelWidth / 2, \_camera.pixelHeight / 2, 0);**
- **Ray ray = \_camera.ScreenPointToRay(point);**
- **RaycastHit hit;**
- **if (Physics.Raycast(ray, out hit)) {**
- **Debug.Log("Hit" + hit.point);**
- **}**
- **}**
- }
- }

## Добавление визуальных индикаторов для попаданий

- `if (Physics.Raycast(ray, out hit)) {`
- `StartCoroutine(SphereIndicator(hit.point));`
- `}`
- `}`
- `}`
- `private IEnumerator SphereIndicator(Vector3 pos) {`
- `GameObject sphere =`
- `GameObject.CreatePrimitive(PrimitiveType.Sphere);`
- `sphere.transform.position = pos;`
- `yield return new WaitForSeconds(1);`
- `Destroy(sphere);`
- `}`
- `}`

## Создаем индикатор для прицеливания

- `void Start () {`
- `_camera = GetComponent<Camera>();`
- `Cursor.lockState = CursorLockMode.Locked;`
- `Cursor.visible = false;`
- `}`
  
- `private void OnGUI()`
- `{`
- `int size = 12;`
- `float posX = _camera.pixelWidth / 2 - size / 4;`
- `float posY = _camera.pixelHeight / 2 - size / 2;`
- `GUI.Label(new Rect(posX, posY, size, size), "*");`
- `}`

# Создаем активные цели

- Создайте куб (1,2,1), поместите его в точку (0,1,0).
- Дайте ему имя Enemy
- Создайте сценарий ReactiveTarget и присоедините его к объекту.



# Определяем точку попадания

- if (Physics.Raycast(ray, out hit))
- {
- **GameObject hitObject = hit.transform.gameObject;**
- **ReactiveTarget target =**
- **hitObject.GetComponent<ReactiveTarget>();**
- **if (target != null)**
- {
- **Debug.Log("Target hit");**
- }
- **else**
- {
- StartCoroutine(SphereIndicator(hit.point));
- }
- } } }

# Уведомляем цель о попадании

- if (target != null)
  - {
  - **target.ReactToHit();**
  - }
  - else
  - {
  -
- StartCoroutine(SphereIndicator(hit.point));

# Сценарий ReactiveTarget, реализующий смерть врага при попадании

- `public class ReactiveTarget : MonoBehaviour {`
- `public void ReactToHit() {`
- `StartCoroutine(Die());`
- `}`
  
- `private IEnumerator Die()`
- `{`
- `this.transform.Rotate(-75, 0, 0);`
- `yield return new WaitForSeconds(1.5f);`
- `Destroy(this.gameObject);`
- `}`
- `}`

Базовый искусственный интеллект для перемещения по сцене

- `public class WanderingAI : MonoBehaviour {`
- `public float speed = 3.0f;`
- `public float obstacleRange = 5.0f;`
- `void Update () {`
- `transform.Translate(0, 0, speed * Time.deltaTime);`
- `Ray ray = new Ray(transform.position, transform.forward);`
- `RaycastHit hit;`
- `if (Physics.SphereCast(ray, 0.75f, out hit)) {`
- `if (hit.distance < obstacleRange) {`
- `float angle = Random.Range(-110, 110);`
- `transform.Rotate(0, angle, 0);`
- `}}}`

# Слежение за состоянием персонажа (WanderingAI)

- `private bool _alive;`
- `void Start () {`
- `_alive = true;`
- `}`
  
- `void Update () {`
- `if (_alive)`
- `{`
- `transform.Translate(0, 0, speed * Time.deltaTime);`
- `}`
- `... }`
- `//дописать в конце программы`
- `public void SetAlive(bool alive) {`
- `_alive = alive;`
- `}`

Персонаж жив

Открытый метод, позволяющий внешнему коду действовать на «живое» состояние

ReactiveTarget сообщает сценарию WanderingAI, когда наступает  
смерть  
(дописываем в ReactiveTarget)

- public void ReactToHit() {
  - **WanderingAI behavior =**  
**GetComponent<WanderingAI>();**
  - **if (behavior != null) {**
  - **behavior.SetAlive(false);**
  - **}**
  - StartCoroutine(Die());
  - }
- Проверяем  
присоединен ли к  
персонажу сценарий  
WanderingAI

# Что такое шаблон экземпляров?

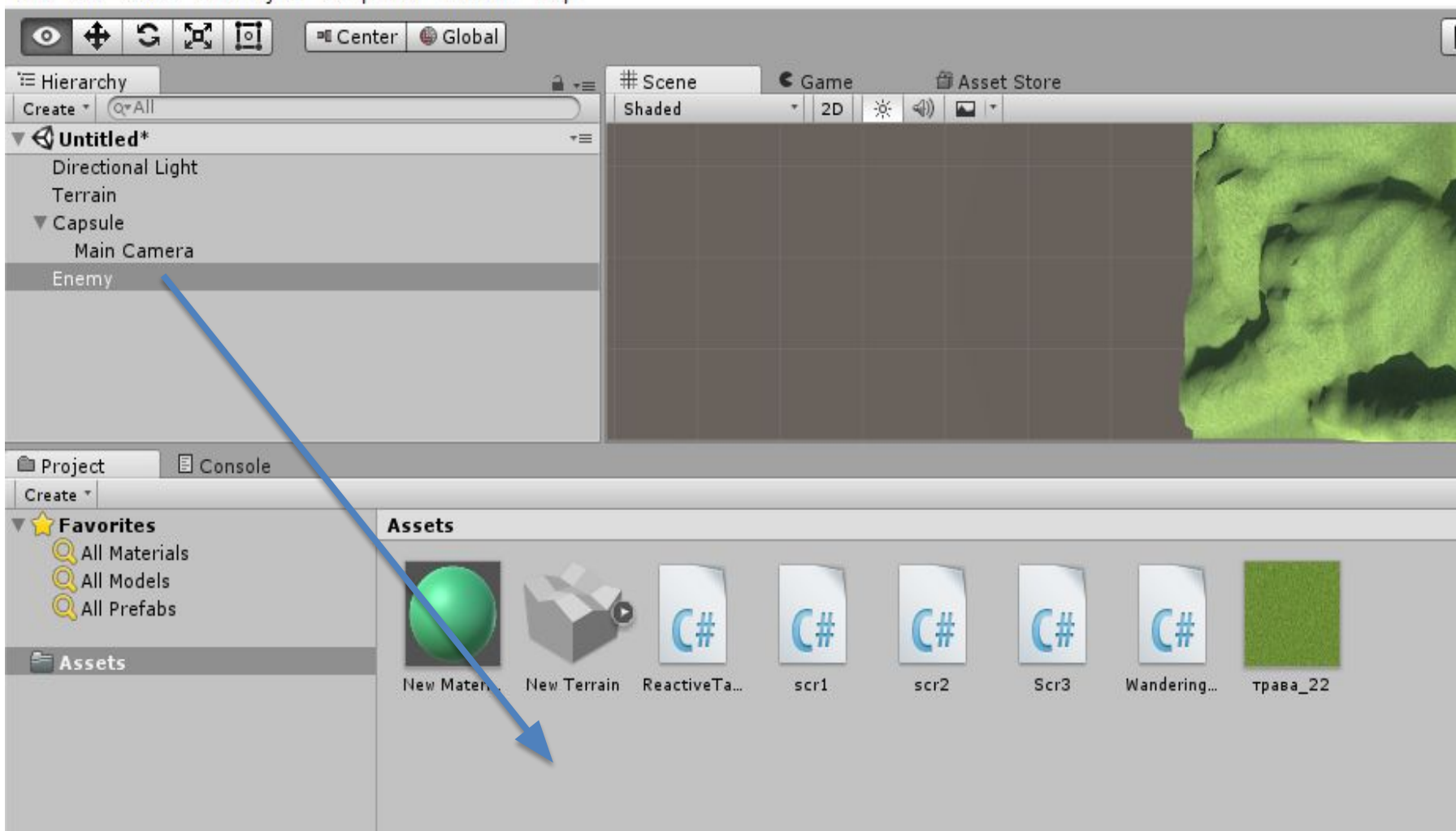
- Это полностью сформированный игровой объект (с уже присоединенными компонентами), существующий не внутри конкретной сцены, а в виде ресурса, который может быть скопирован в любую сцену.
- Копии объектов могут создаваться вручную и порождаться кодом



# Увеличение количества врагов

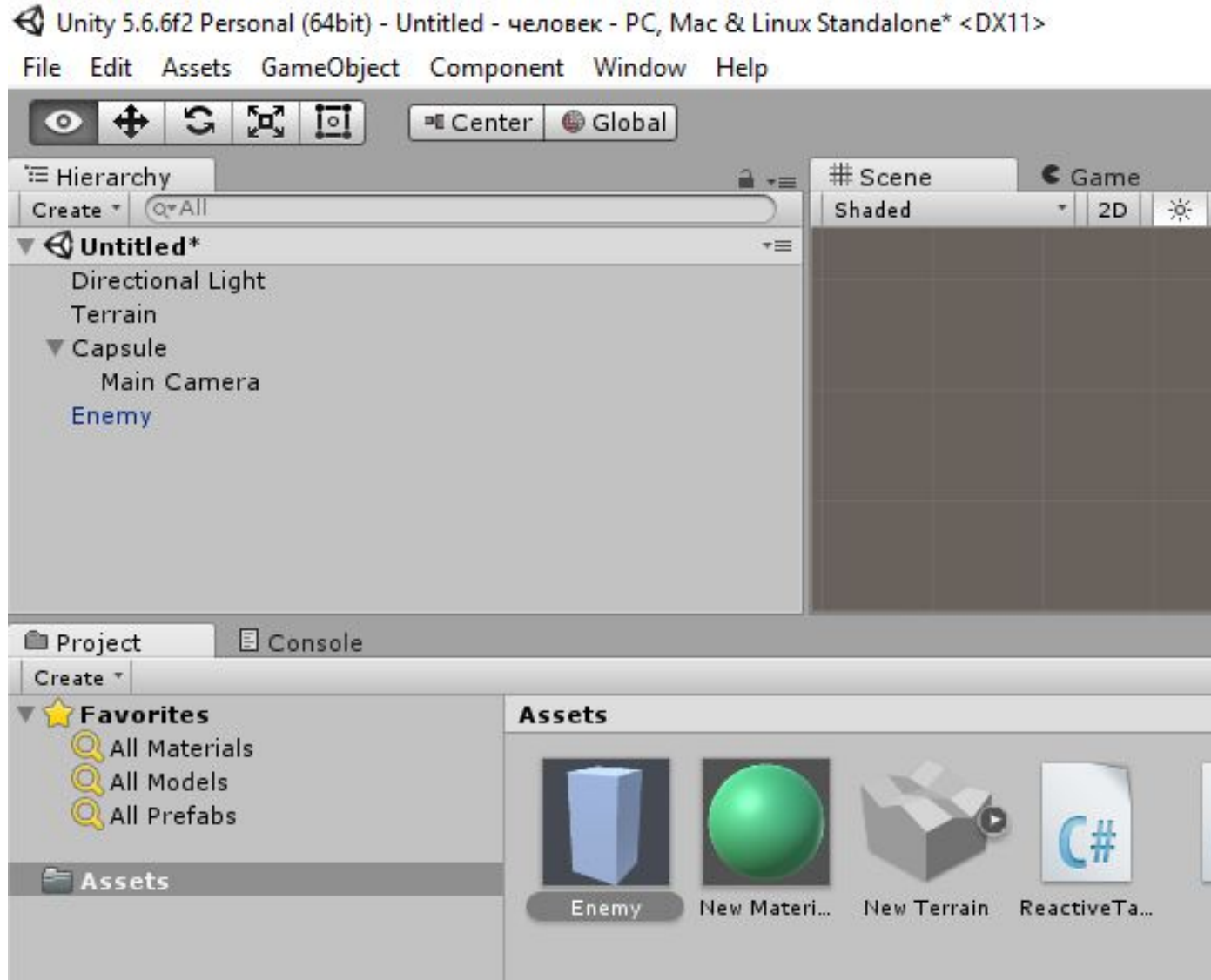
Unity 5.6.6f2 Personal (64bit) - Untitled - человек - PC, Mac & Linux Standalone\* <DX11>

File Edit Assets GameObject Component Window Help



# Получится вот так...

## После удаляем его из Hierarchy



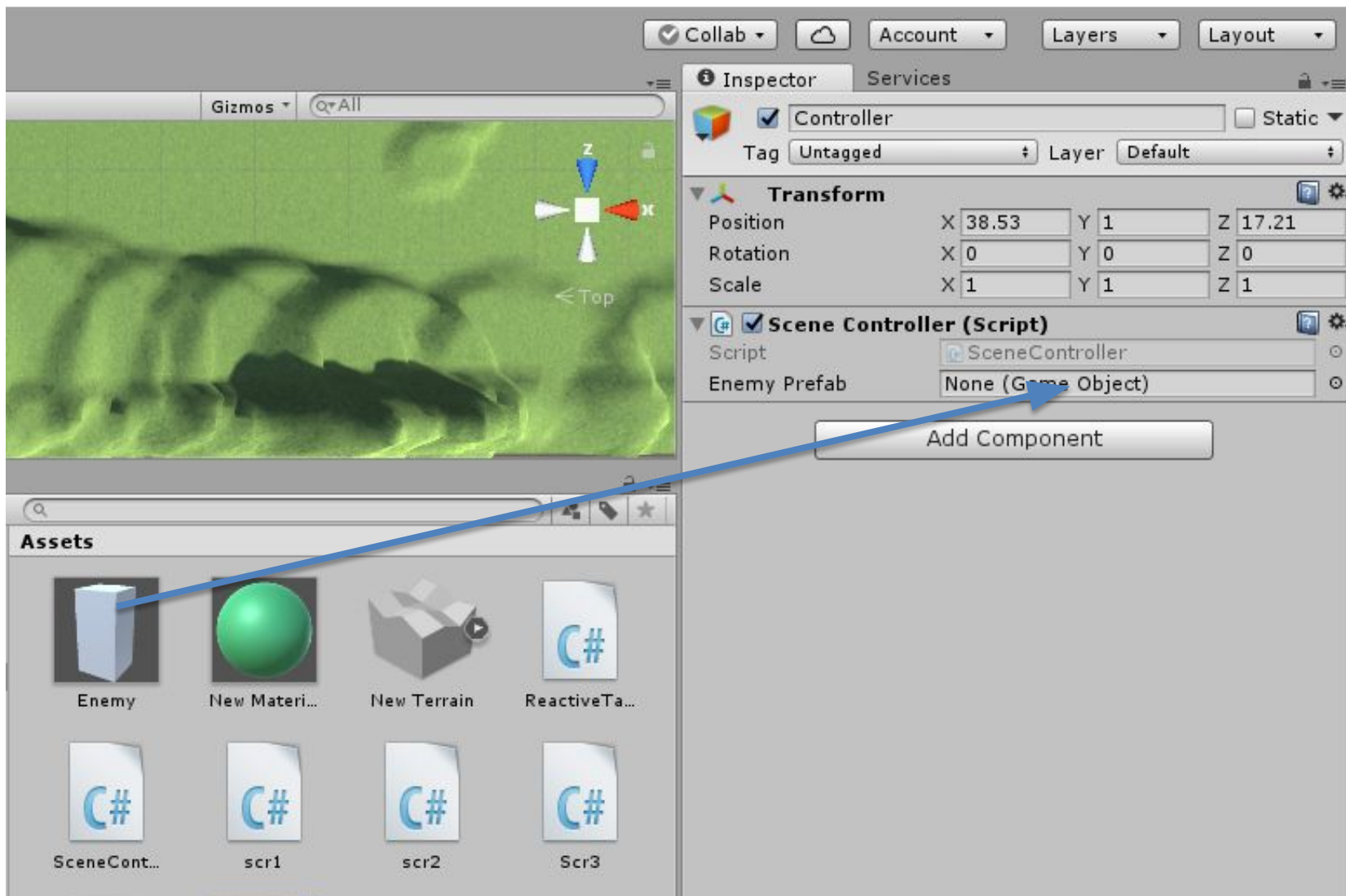
К какому объекту присоединить сценарий, размножающий врагов?

Создаем пустой объект. Даем ему имя Controller  
Пишем сценарий SceneController, порождающий экземпляры врагов

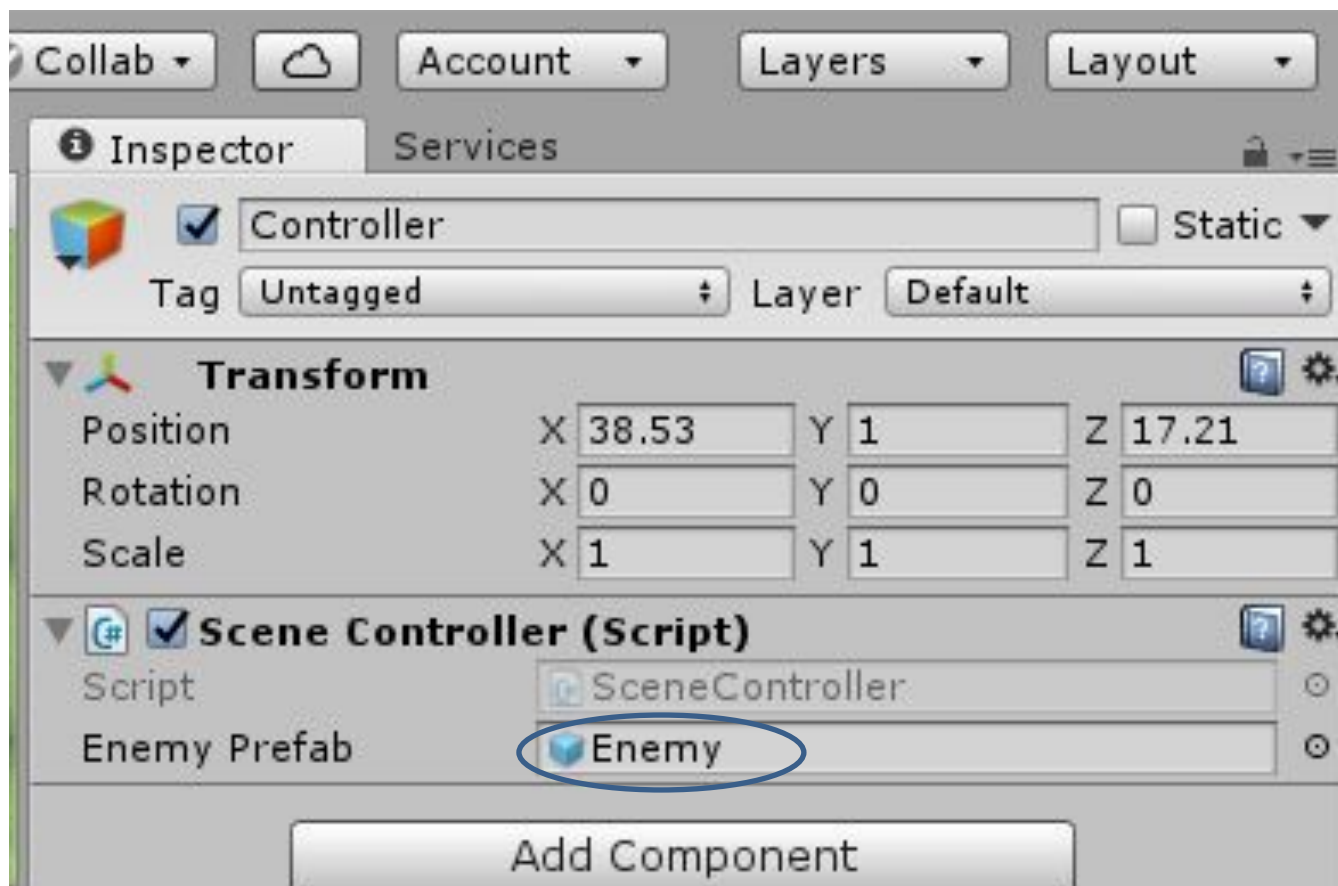
- public class SceneController : MonoBehaviour {
- **[SerializeField] private GameObject enemyPrefab;** Сериализованная переменная для связи с объектом-шаблоном
- **private GameObject \_enemy;** Закрытая переменная для слежения за экземпляром врага в сцене
- void Update () { Порождаем нового врага только если в сцене враги отсутствуют
- **if (\_enemy == null) {**
- **\_enemy = Instantiate(enemyPrefab) as GameObject;** Метод, копирующий объект-шаблон
- **\_enemy.transform.position = new Vector3(0, 1, 0);**
- **float angle = Random.Range(0, 360);**
- **\_enemy.transform.Rotate(0, angle, 0);**
- **}}}**

Присоедините сценарий SceneController к пустому объекту Controller.

Перетащите шаблон врага на пустое поле переменной



# Получится так



# Стрельба

- Создайте шар
- Переименуйте его в **Fireball**
- Создайте новый сценарий с именем **Fireball** и присоедините его к сфере
- Создайте материал `flame`. Выберите для него оранжевый цвет и поменяйте параметр **Emission** на **0.3**, что бы сделать материал более ярким.
- Превращаем наш огненный шар в шаблон, перетащив его со вкладки **Hierarchy** на вкладку **Project**.

Что бы код распознавал игрока, создадим сценарий

PlayerCharacter (для игрока)



# Откроем сценарий WanderingAI

...

```
[SerializeField] private GameObject fireballPrefab;
```

```
private GameObject _fireball; //стилизованная переменная для связи с объектом  
шаблоном
```

...

```
if (Physics.SphereCast(ray, 0.75f, out hit)) {
```

```
    GameObject hitObject = hit.transform.gameObject;
```

```
    if (hitObject.GetComponent < PlayerCharacter>())
```

```
{
```

```
    if (_fireball == null) {
```

```
        _fireball = Instantiate(fireballPrefab) as GameObject;
```

```
        _fireball.transform.position = transform.TransformPoint(Vector3.forward * 1.5f);
```

```
        _fireball.transform.rotation = transform.rotation
```

```
    }
```

```
    }
```

```
else
```

```
    if (hit.distance < obstacleRange) {
```

...

Щелкните на шаблоне врага в вкладке **Project**.  
В **Inspector** появятся его свойства.  
Перетащите шаблон огненного шара со вкладки **Project** на поле **Fireball Prefab** панели **Inspector**



# Сценарий Fireball, реагирующий на СТОЛКНОВЕНИЯ

```
• public class Fireball : MonoBehaviour {  
•     public float speed = 10.0f;  
•     public int damage = 1;  
  
• void Update () {  
•     transform.Translate(0, 0, speed * Time.deltaTime);  
• }  
  
• private void OnTriggerEnter(Collider other)  
• {  
•     PlayerCharacter player = other.GetComponent<PlayerCharacter>();  
•     if (player != null) {  
•         Debug.Log("Player hit");  
•     }  
•     Destroy(this.gameObject);  
• }  
• }
```

Эта функция вызывается  
автоматически при столкновении  
объекта

# Что бы сработал триггер

- Установить флажок **Is Trigger** в разделе **Sphere Collider**
- Добавьте огненному шару компонент **Rigidbody** и сбросьте у него флажок **Use Gravity**

## Заставим игрока реагировать на попадания (сценарий PlayerCharacter)

- `private int _health;`
- `void Start () {`
- `_health = 5;`
- `}`
- `public void Hurt (int damage){`
- `_health -=damage;`
- `Debug.Log(“Здоровье: “ + _health);`
- `}`
- `}`

# Возвращаемся к сценарию Fireball

- `public class Fireball : MonoBehaviour {`
- `public float speed = 10.0f;`
- `public int damage = 1;`
  
- `void Update () {`
- `transform.Translate(0, 0, speed * Time.deltaTime);`
- `}`
  
- `private void OnTriggerEnter(Collider other)`
- `{`
- `PlayerCharacter player = other.GetComponent<PlayerCharacter>();`
- `if (player != null) {`
- `Debug.Log("Player hit"); player.Hurt(damage);`
- `}`
- `Destroy(this.gameObject);`
- `}`
- `}`

Сообщает игроку о попадании