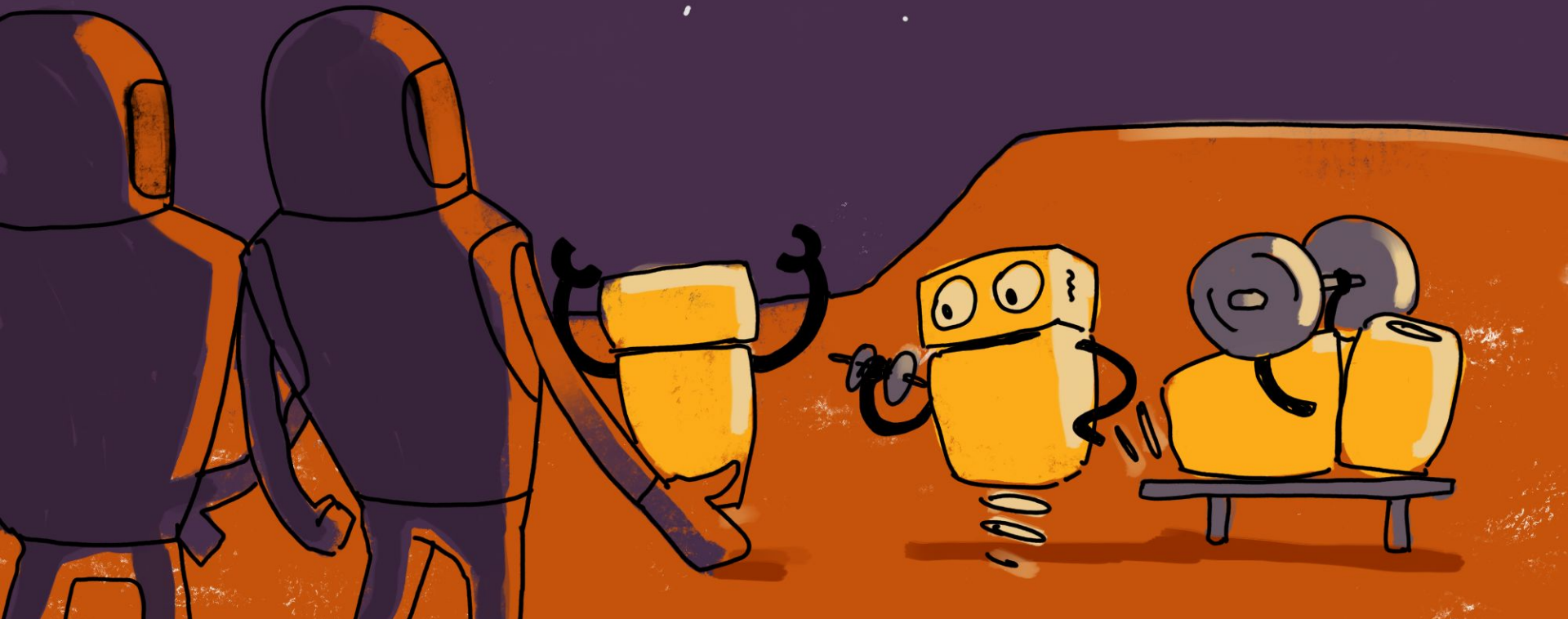
алгоритмика

Международная школа
программирования для детей

Unity 2D

Повтор



На прошлом уроке мы познакомились с Unity а так же узнали, что такое компоненты.

Какие компоненты вы запомнили?

Мы сейчас должны знать 4 компонента

> Компонент **Transform** есть у всех объектов в Unity. Он отвечает за месторасположение объекта на экране, его размеры и поворот.

> Компонент **Sprite Renderer** есть у всех видимых на сцене объектов в Unity. Он отвечает за отображение спрайтов (картинок). Цвет, поворот и прочие.

> Компонент **Rigidbody 2D** отвечает за физику в 2D пространстве. Масса (вес), показатель гравитации, материал и пр.

> Компонент **Box Collider 2D** отвечает за коллизию (столкновение) двух коллайдеров.

Также мы написали с вами скрипт, с помощью которого наш персонаж получил возможность ходить

Помните ли вы, какую функцию мы использовали, для телепортации нашего персонажа?

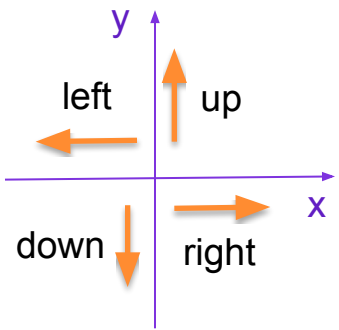
```
if (Input.GetKey(KeyCode.A))
{
    transform.Translate(Vector2.left * Time.deltaTime );
}
```

transform.Translate() функция, телепортирующая нашего персонажа в сторону, указанную в скобках.

```
if (Input.GetKey(KeyCode.A))
{
    transform.Translate(Vector2.left * Time.deltaTime );
}
```

transform.Translate() функция, телепортирующая нашего персонажа в сторону, указанную в скобках.

Vector2.left – направление в двумерной системе координат влево



Сегодня на уроке

- Мы поближе познакомимся с уже знакомыми компонентами
- Познакомимся с новыми видами коллайдеров
- Реализуем прыжок для нашего игрока
- Научим камеру передвигаться за игроком
- Узнаем, что такое Префабы и как с ними работать

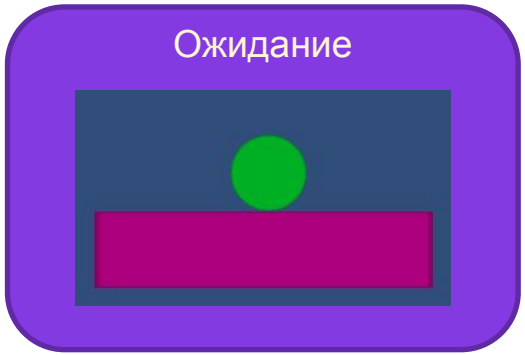
Камера за персонажем...

Как правило, в играх камера всегда следует за персонажем, однако сейчас, наша камера стоит на месте.

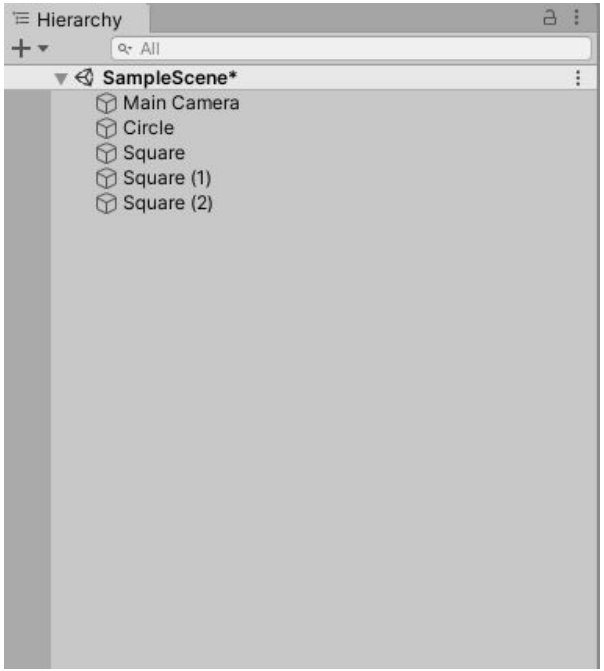
Камера за персонажем...

Как правило, в играх камера всегда следует за персонажем, однако сейчас, наша камера стоит на месте.

Существует множество способов реализации движения камеры за персонажем.
Сегодня мы попробуем самый простой из них.

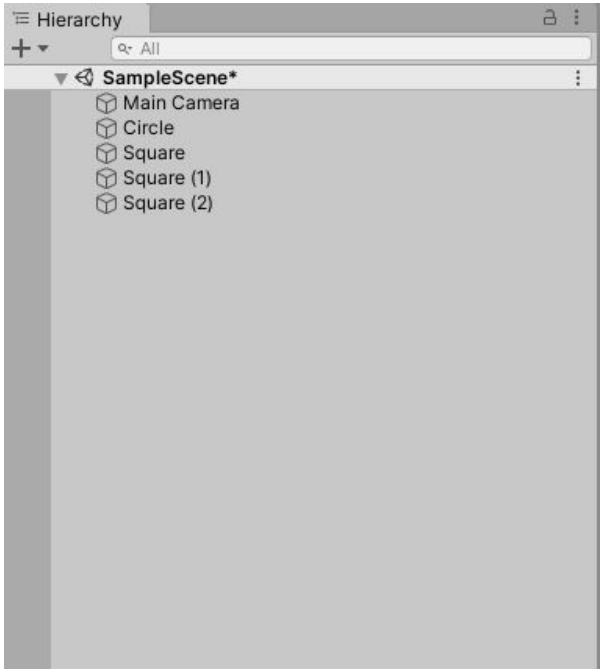


Камера за персонажем...



Обратите внимание на окно **Hierarchy** (Иерархия).

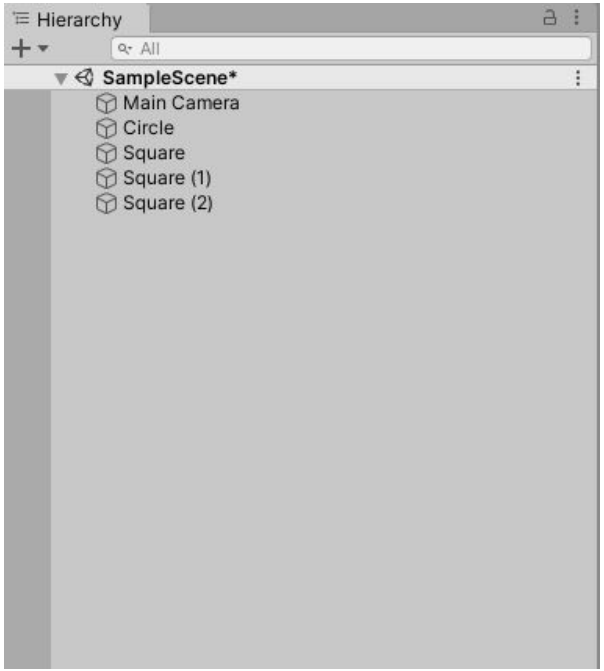
Камера за персонажем...



Обратите внимание на окно **Hierarchy** (Иерархия).

Как мы говорили, в нём отображаются все объекты на нашей сцене. Однако у него есть и другое применение.

Камера за персонажем...

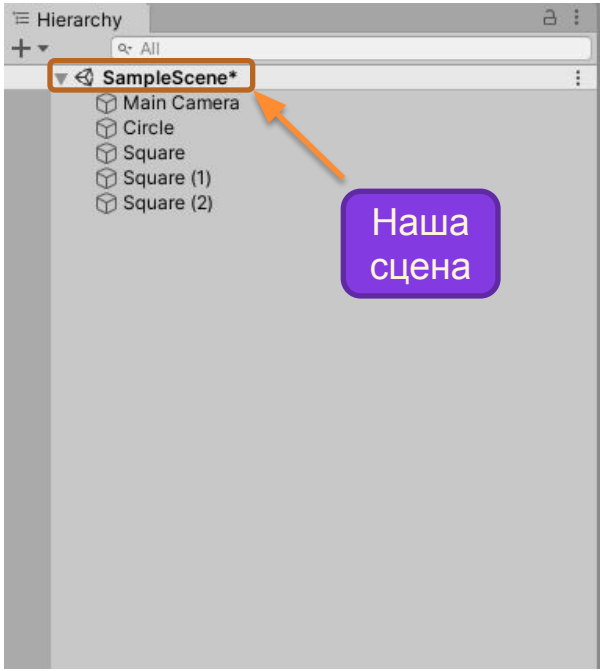


Обратите внимание на окно **Hierarchy** (Иерархия).

Как мы говорили, в нём отображаются все объекты на нашей сцене. Однако у него есть и другое применение.

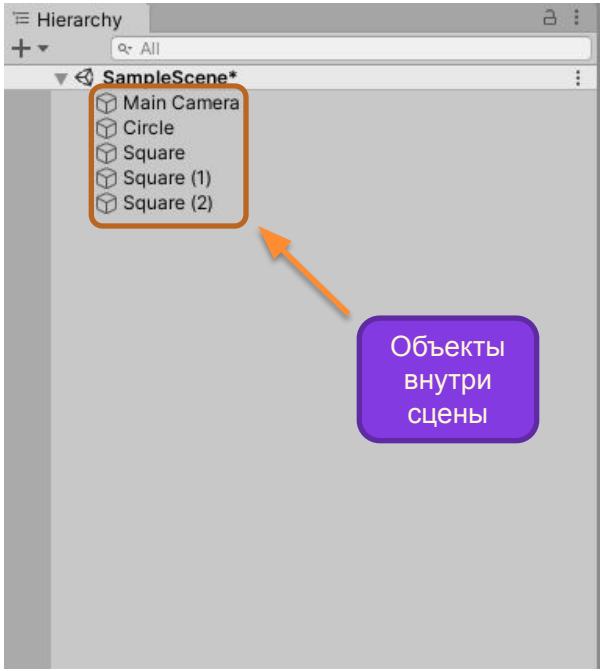
Окно Иерархии позволяет нам узнать, какие объекты находятся внутри других.

Камера за персонажем...



Сейчас мы видим с вами активную сцену.

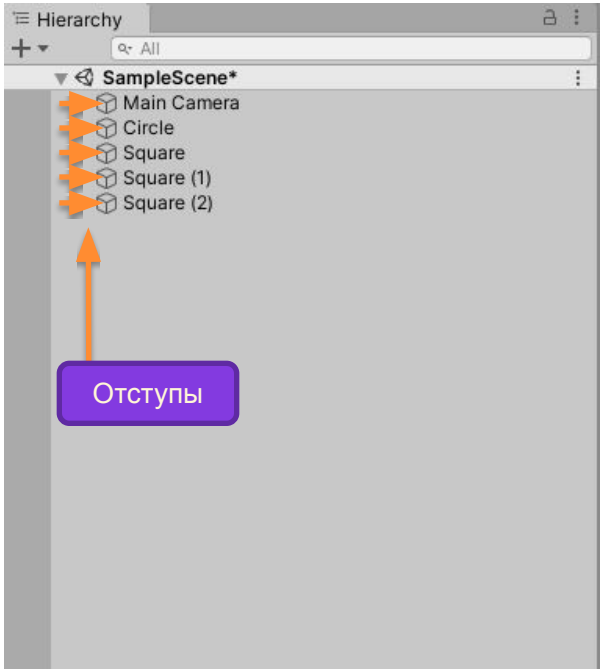
Камера за персонажем...



Сейчас мы видим с вами активную сцену.

В этой сцене, 5 объектов

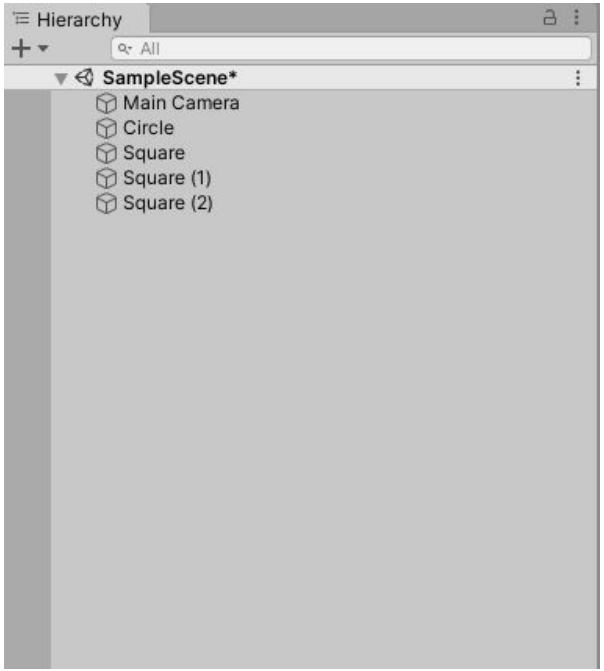
Камера за персонажем...



Сейчас мы видим с вами активную сцену.

В этой сцене, 5 объектов, мы можем это понять, по отступам, у объектов под сценой.

Камера за персонажем...

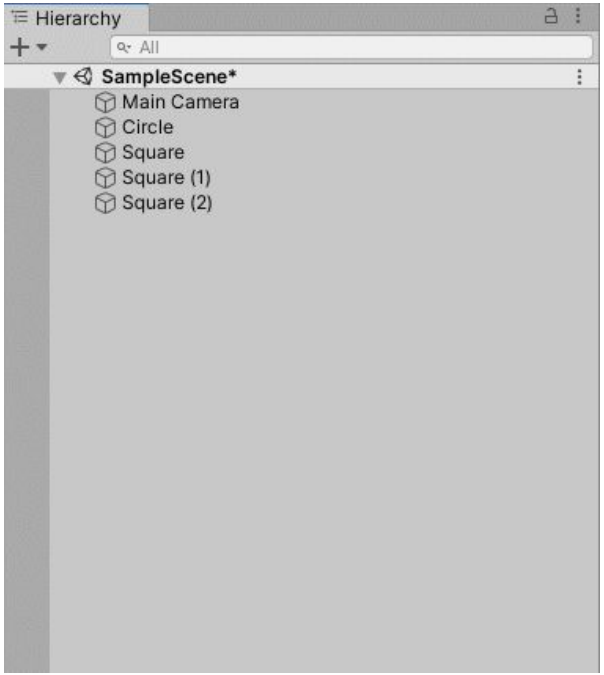


Сейчас мы видим с вами активную сцену.

В этой сцене, 5 объектов, мы можем это понять, по отступам, у объектов под сценой.

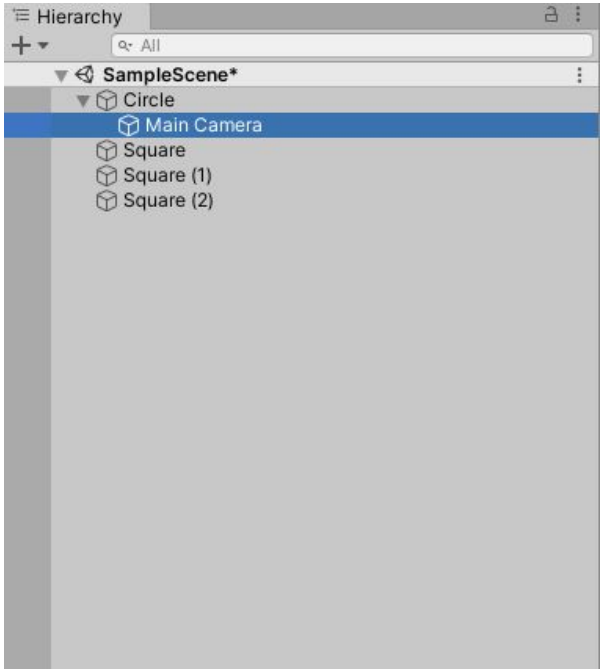
Внутри этих объектов сейчас пусто. Давайте изменим это.

Камера за персонажем...



Зажмите нашу камеру, и перетащите её в нашего персонажа.

Камера за персонажем...



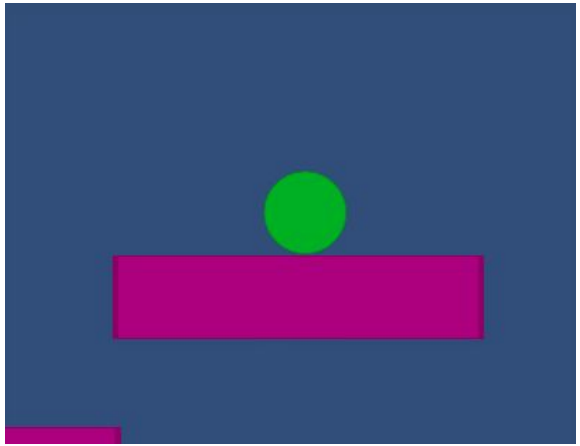
Зажмите нашу камеру, и перетащите её в нашего персонажа.

Теперь наша камера расположена внутри персонажа.

А это значит, что при перемещении нашего персонажа, будет двигаться и камера!

Попробуйте запустить игру!

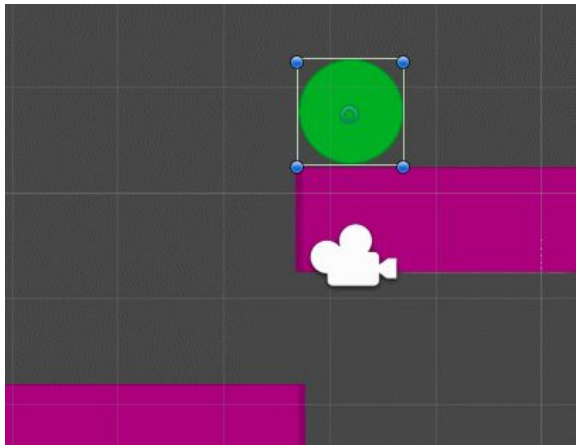
Камера за персонажем...



И вроде бы всё хорошо, но у вас не случалось вот такой проблемы?

Кажется наш персонаж перевернулся, а вместе с ним и камера.

Камера за персонажем...



И вроде бы всё хорошо, но у вас не случалось вот такой проблемы?

Кажется наш персонаж перевернулся, а вместе с ним и камера.

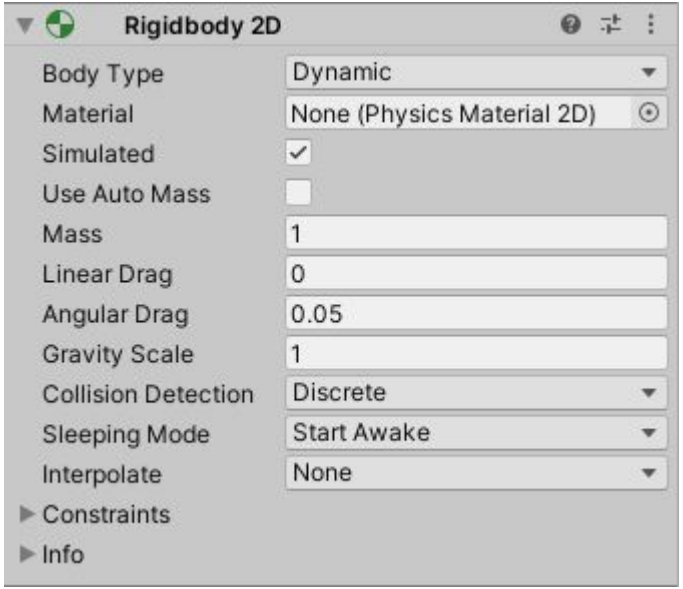
Это довольно таки логично, однако, хотелось бы, чтобы такого не происходило.

Решить такую проблему легко, для этого мы вспомним один уже знакомый нам компонент

Немного о Rigidbody2D

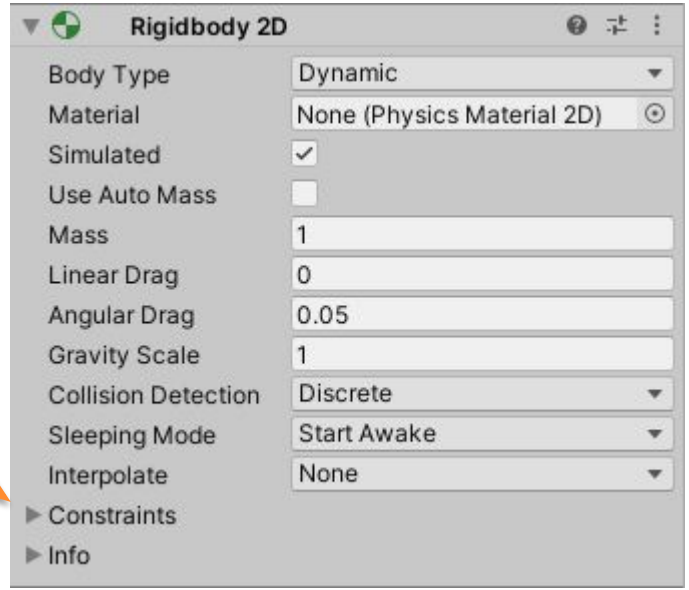
С Rididbody2D Мы уже с вами работали, однако у этого компонента есть множество свойств, которые мы будем изучать в процессе занятий.

Давайте познакомимся с одним из таких свойств!



Немного о Rigidbody2D

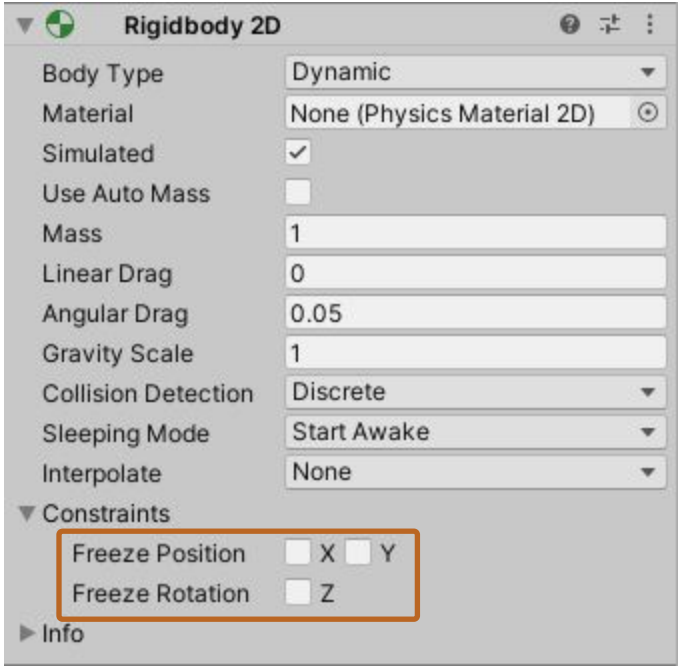
Но чтобы найти это свойство, давайте раскроем вкладку **Constraints** у компонента Rigidbody2D



Немного о Rigidbody2D

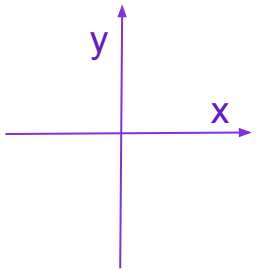
Но чтобы найти это свойство, давайте раскроем вкладку **Constraints** у компонента Rigidbody2D

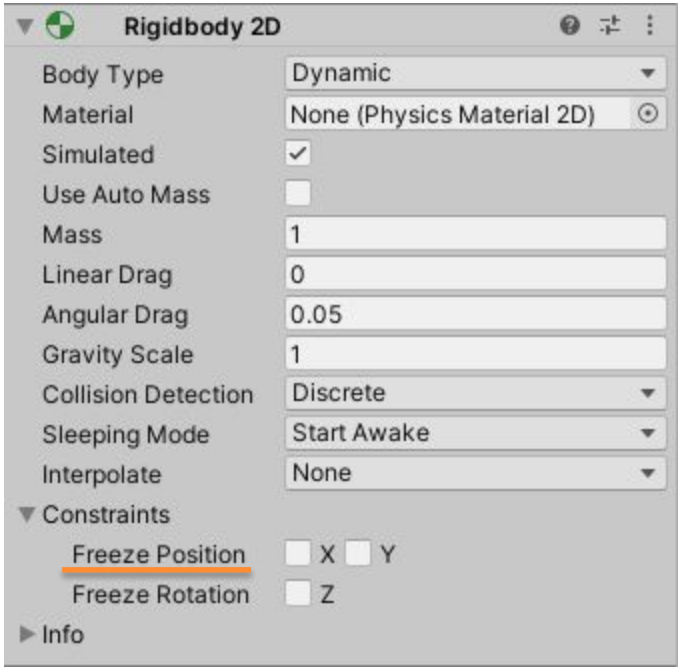
Здесь мы увидим ещё пару свойств Rigidbody: Freeze Position и Freeze Rotation.



Немного о Rigidbody2D

Галочка у Freeze Position запрещает объекту перемещаться по оси **x**, или **y** (Таким образом можно заставить объект парить, не падая на землю)

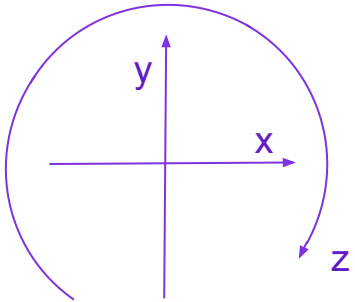


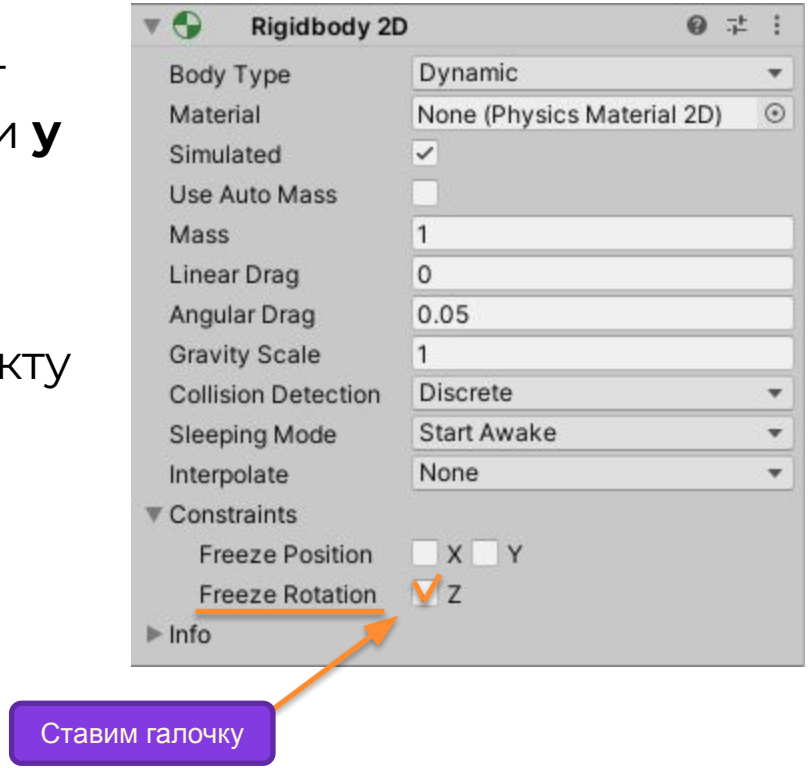


Немного о Rigidbody2D

Галочка у Freeze Position запрещает объекту перемещаться по оси **x**, или **y** (Таким образом можно заставить объект парить, не падая на землю)

А вот Freeze Rotation запретит объекту переворачиваться (крутиться).

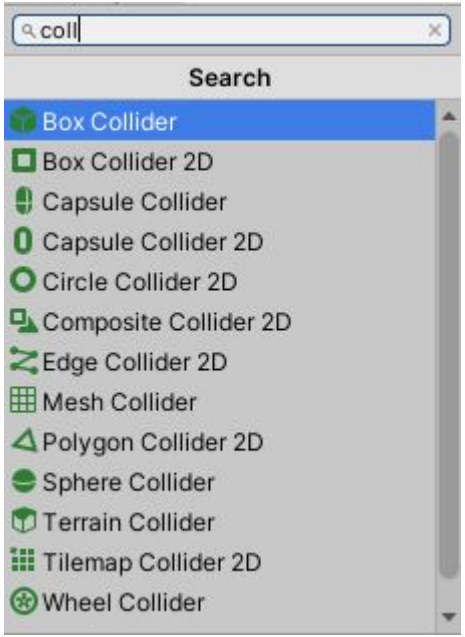




О коллайдерах...

Мы с вами уже знакомы с компонентом Box Collider, однако самые внимательные могли заметить, что в Unity у нас есть ещё несколько коллайдеров.

Давайте с ними познакомимся!



Box Collider – Прямоугольный коллайдер. Отлично подходит, если объект, нуждающийся в коллайдере похож на прямоугольник. Меньше всего затрачивает ресурсы в приложении.

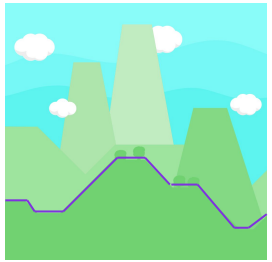
Box Collider – Прямоугольный коллайдер. Отлично подходит, если объект, нуждающийся в коллайдере похож на прямоугольник. Меньше всего затрачивает ресурсы в приложении.

Circle Collider – коллайдер в форме круга. Зачастую используется для круглых, или квадратных спрайтов, чтобы смягчить их углы.

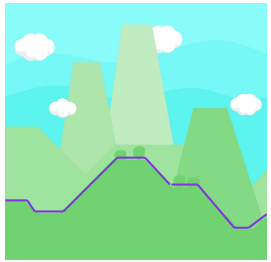
Box Collider – Прямоугольный коллайдер. Отлично подходит, если объект, нуждающийся в коллайдере похож на прямоугольник. Меньше всего затрачивает ресурсы в приложении.

Circle Collider – коллайдер в форме круга. Зачастую используется для круглых, или квадратных спрайтов, чтобы смягчить их углы.

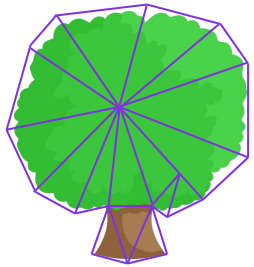
Capsule Collider – коллайдер в форме капсулы. Используется для вытянутых в высоту/ширину персонажей, со смягчёнными углами.



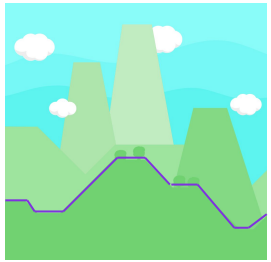
Edge Collider – Выглядит как ломанная линия. Удобен при повторении особенностей ландшафта



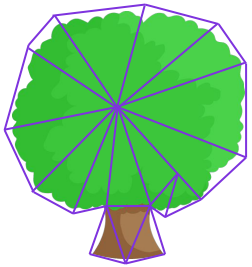
Edge Collider – Выглядит как сломанная линия. Удобен при повторении особенностей ландшафта.



Polygon Collider – Создаётся автоматически. Повторяет форму спрайта. Является плохим выбором, так как ваша игра будет потреблять много ресурсов компьютера/смартфона.



Edge Collider – Выглядит как сломанная линия. Удобен при повторении особенностей ландшафта



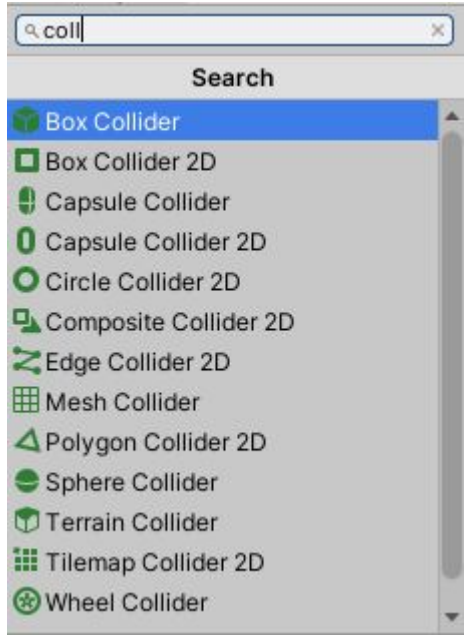
Polygon Collider – Создаётся автоматически. Повторяет форму спрайта. Является плохим выбором, так как ваша игра будет потреблять много ресурсов компьютера/смартфона.

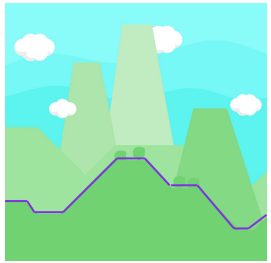
Остальные коллайдеры используются редко, поэтому мы пропустим их.

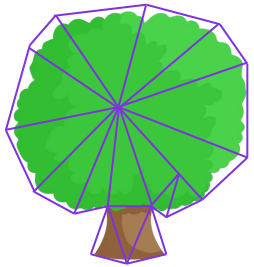
О коллайдерах...

Теперь мы знаем множество новых коллайдеров, и когда их лучше применять!

Постарайтесь в будущем самостоятельно решать, какой коллайдер будет лучше для того, или иного объекта!





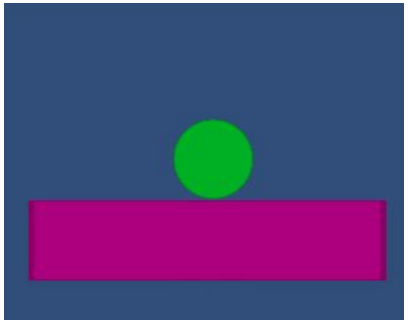


Реализация прыжка

Теперь давайте научим нашего персонажа прыгать!

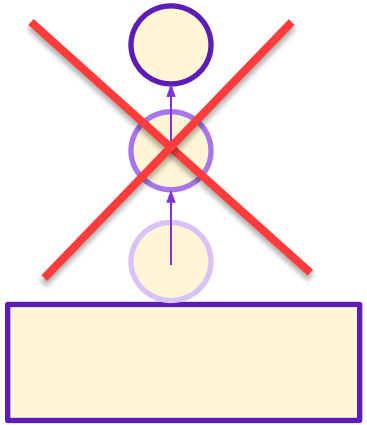
При перемещении мы используем функцию transform.Translate() для телепортации игрока.

Однако для прыжка телепортация совершенно не подходит!



Реализация прыжка

Мы не можем постоянно телепортировать игрока вверх, и назвать это прыжком.



Реализация прыжка

Мы не можем постоянно телепортировать игрока вверх, и назвать это прыжком.

Вместо этого нам нужно придать нашему персонажу едино разовый толчок.

Ведь когда вы прыгаете, вы отталкиваетесь от земли. То же самое должен сделать ваш персонаж.

Реализация прыжка

В Unity есть компонент, который позволит придавать толчок нашему персонажу, и мы уже знакомы с ним.

Это **Rigidbody 2D**, который работает с физикой.

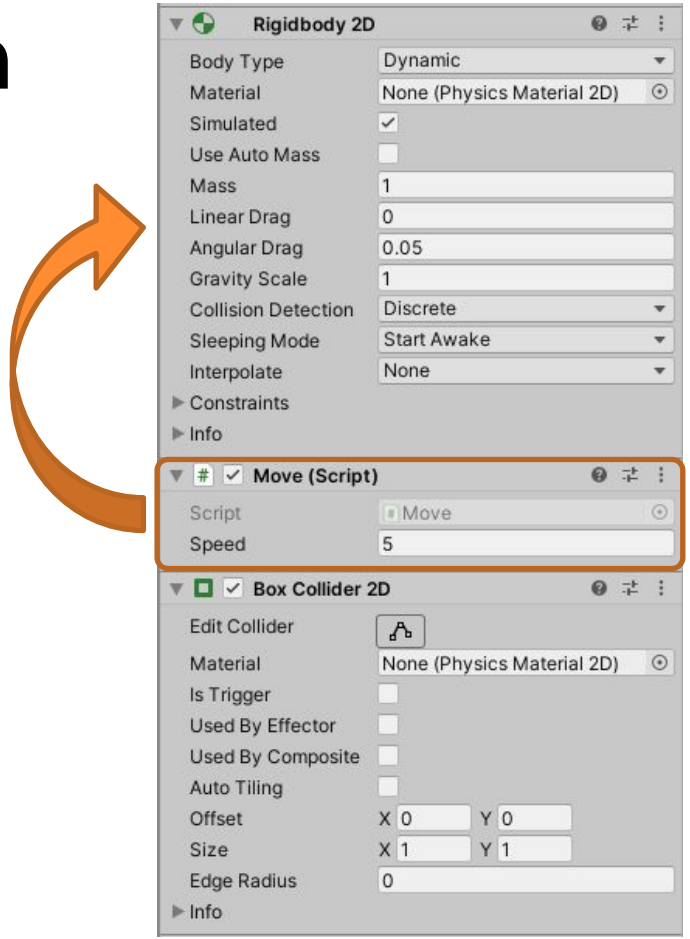
Однако у нас есть одна сложность...

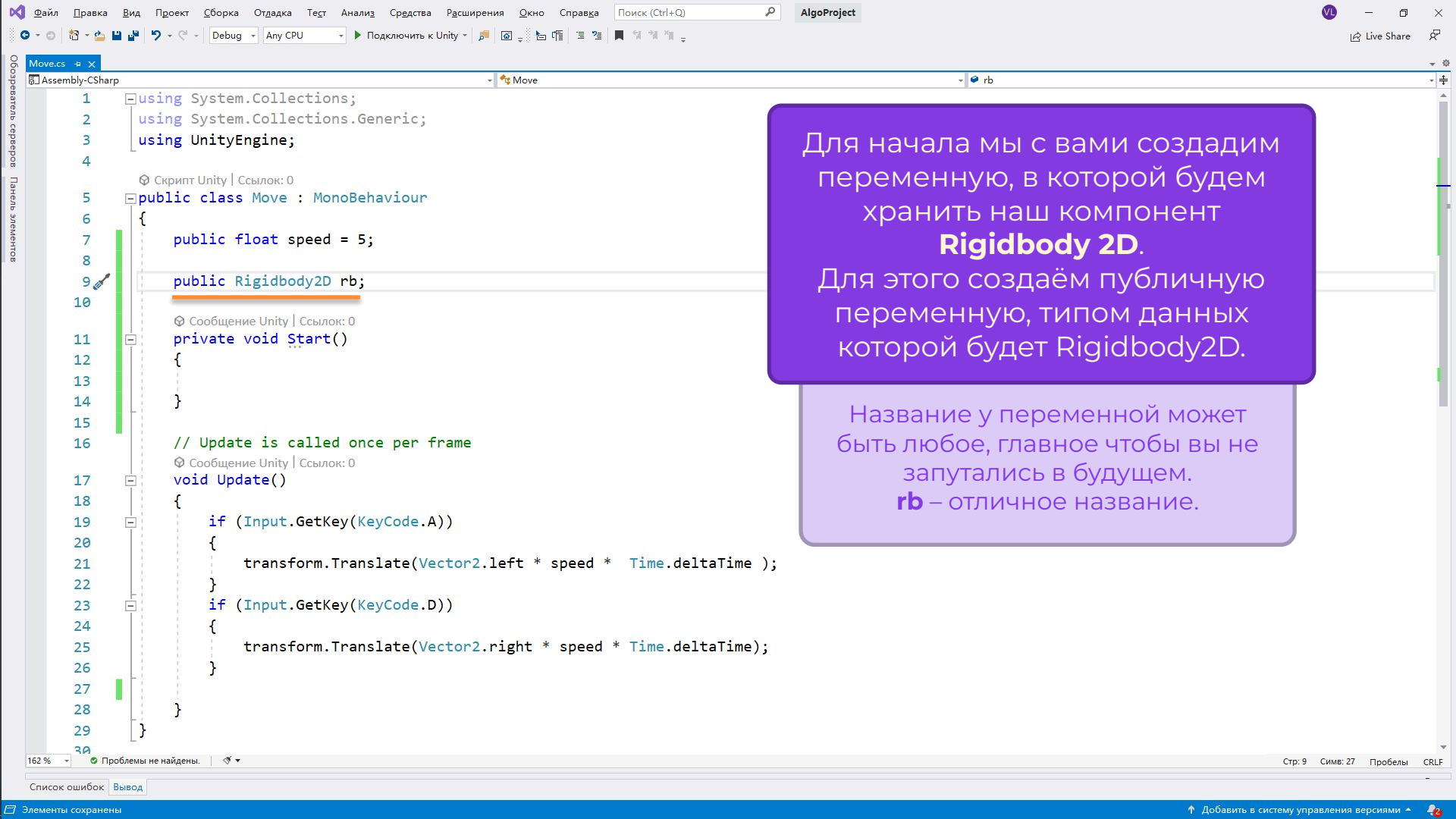
Реализация прыжка

Прыжок мы будем реализовывать в нашем скрипте **Move**.

Однако из скрипта **Move** нам нужно как-то передать команду в **Rigidbody**, чтобы тот активировал толчок.

В решении этой проблемы нам помогут *переменные*!
Смотрите внимательно!

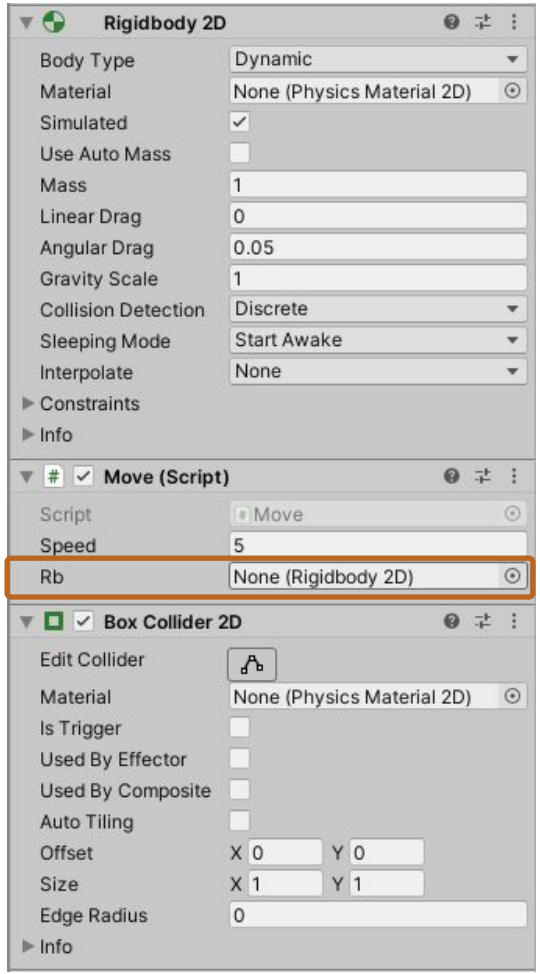



Файл Правка Вид Проект Сборка Отладка Тест Анализ Средства Расширения Окно Справка
Поиск (Ctrl+Q)
AlgoProject
Debug
Any CPU
Подключить к Unity
Live Share
Move.cs
Assembly-CSharp
Move
rb
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
Скрипт Unity | Ссылок: 0
public class Move : MonoBehaviour
{
public float speed = 5;
public Rigidbody2D rb;
Сообщение Unity | Ссылок: 0
private void Start()
{
}
// Update is called once per frame
Сообщение Unity | Ссылок: 0
void Update()
{
if (Input.GetKey(KeyCode.A))
{
transform.Translate(Vector2.left * speed * Time.deltaTime);
}
if (Input.GetKey(KeyCode.D))
{
transform.Translate(Vector2.right * speed * Time.deltaTime);
}
}
}
Для начала мы с вами создадим переменную, в которой будем хранить наш компонент Rigidbody 2D.
Для этого создаём публичную переменную, типом данных которой будет Rigidbody2D.
Название у переменной может быть любое, главное чтобы вы не запутались в будущем.
rb – отличное название.
Проблемы не найдены.
Список ошибок
Вывод
Элементы сохранены
Добавить в систему управления версиями

Реализация прыжка

После сохранения скрипта, в нашем компоненте **Move** появится новая строчка.

Это наша пустая переменная **rb**, которой нужно дать значение, ведь по умолчанию она пустая (none).

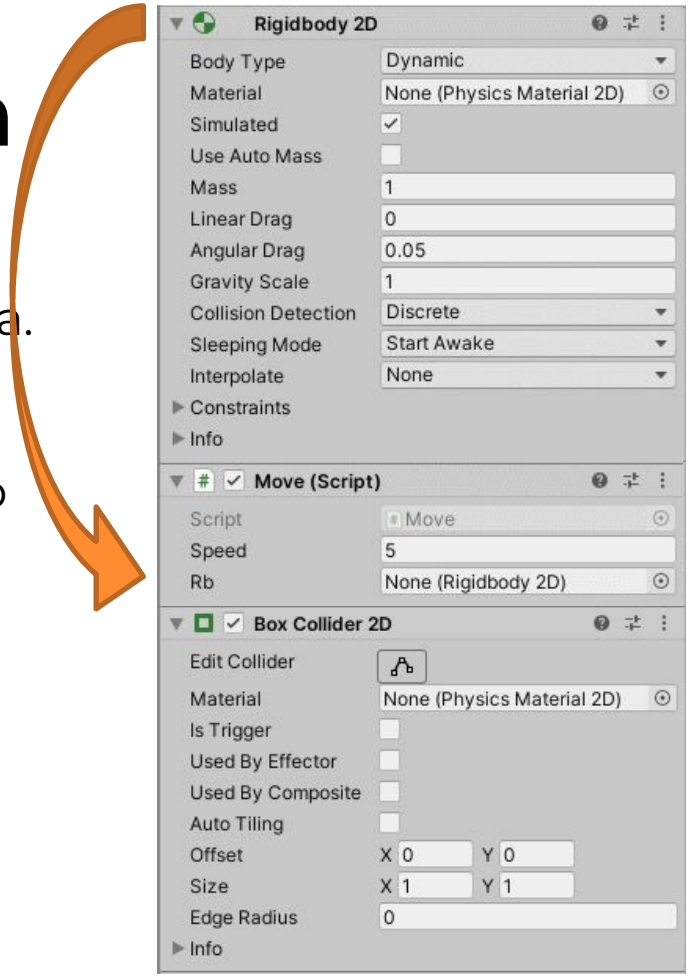


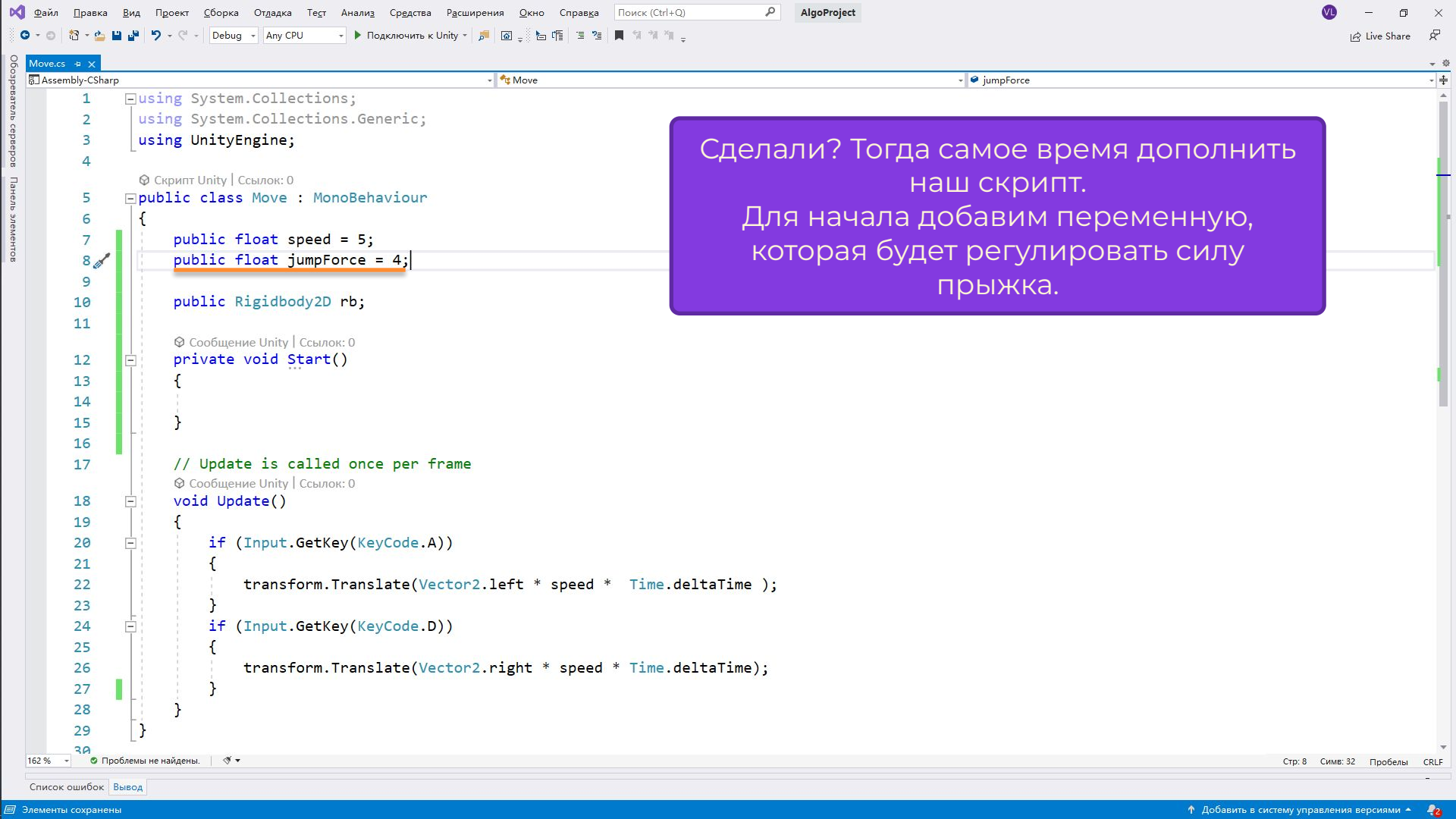
Реализация прыжка

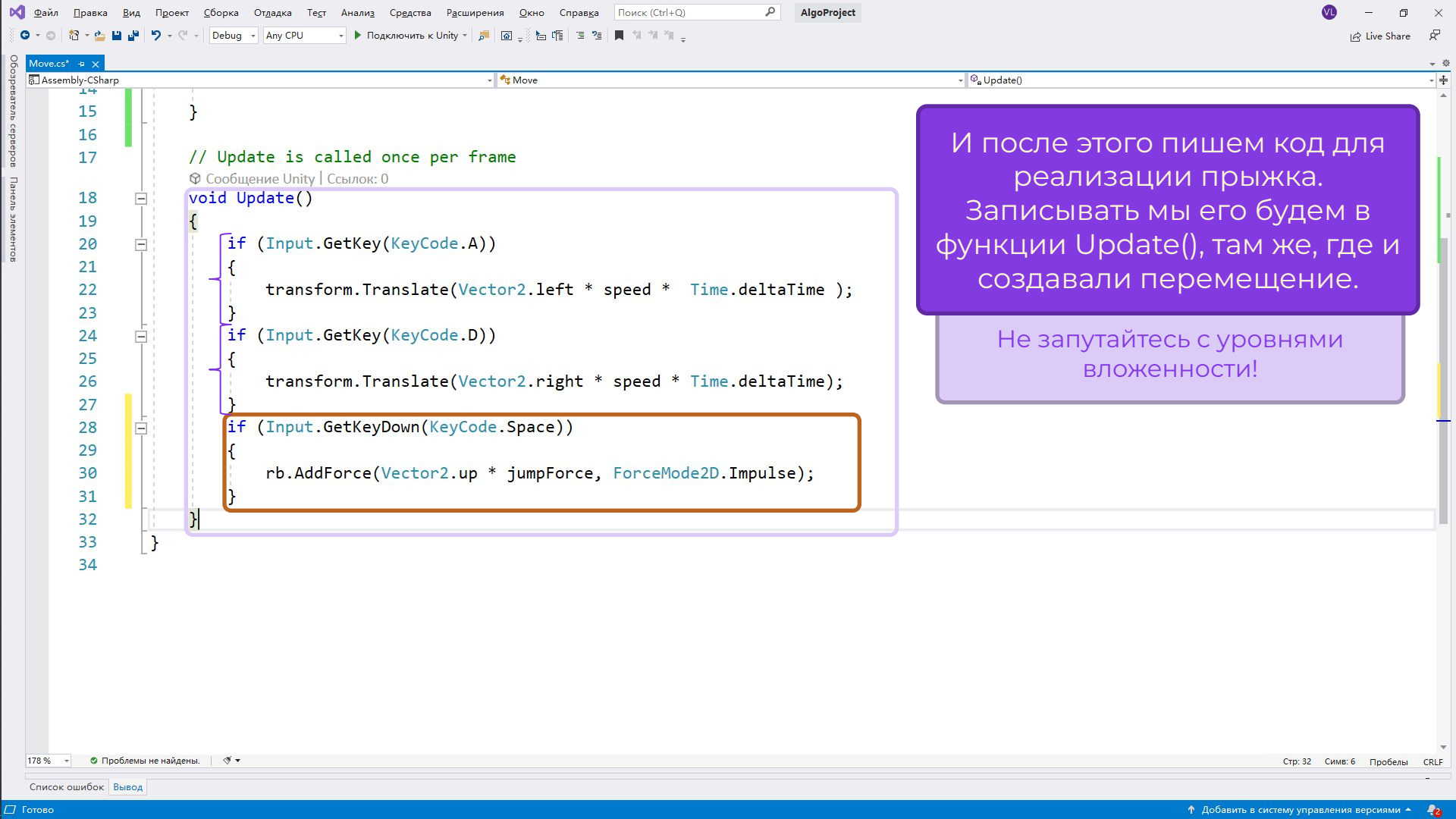
После сохранения скрипта, в нашем компоненте **Move** появится новая строчка.

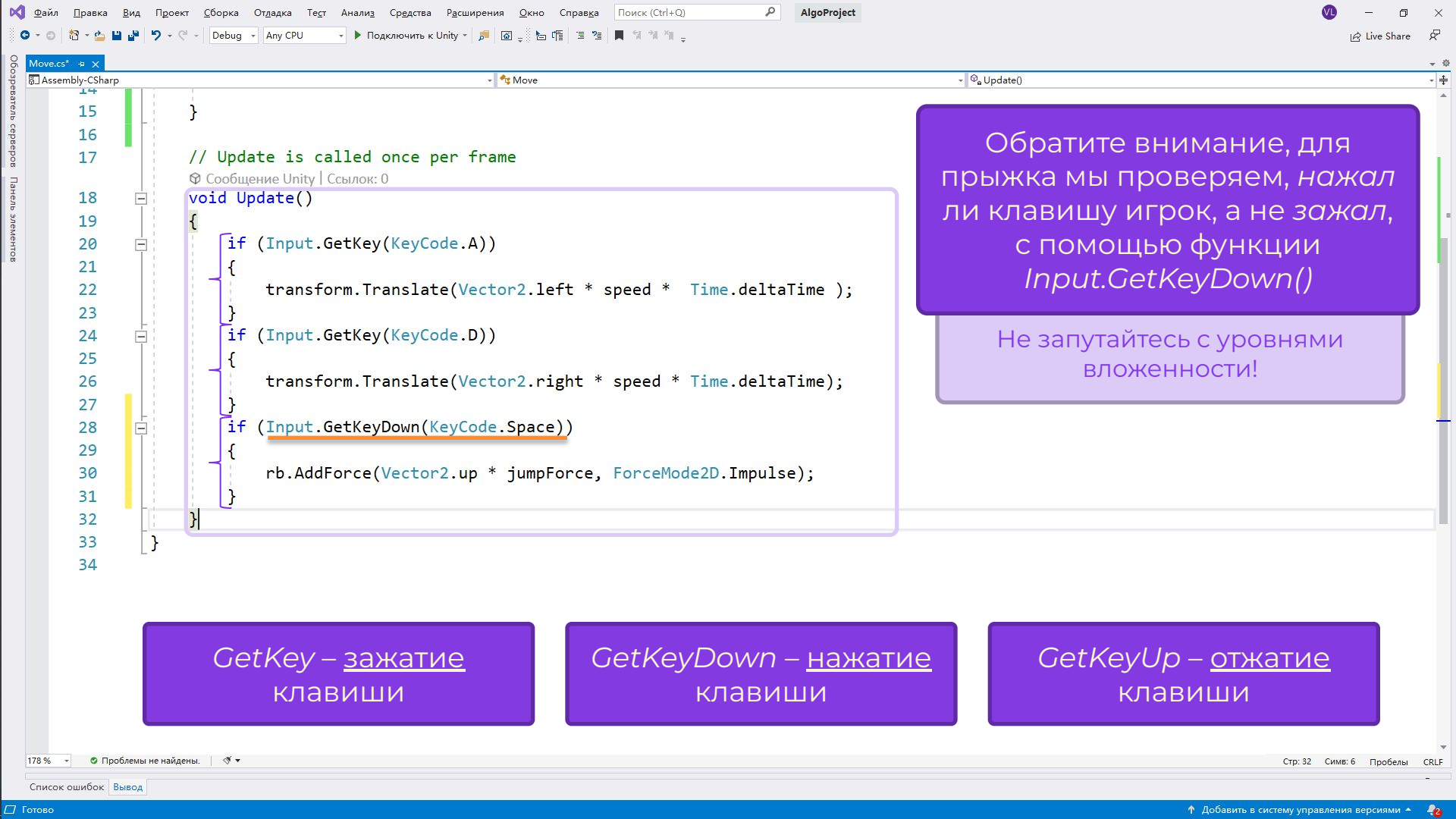
Это наша пустая переменная **rb**, которой нужно дать значение, ведь по умолчанию она пустая (none).

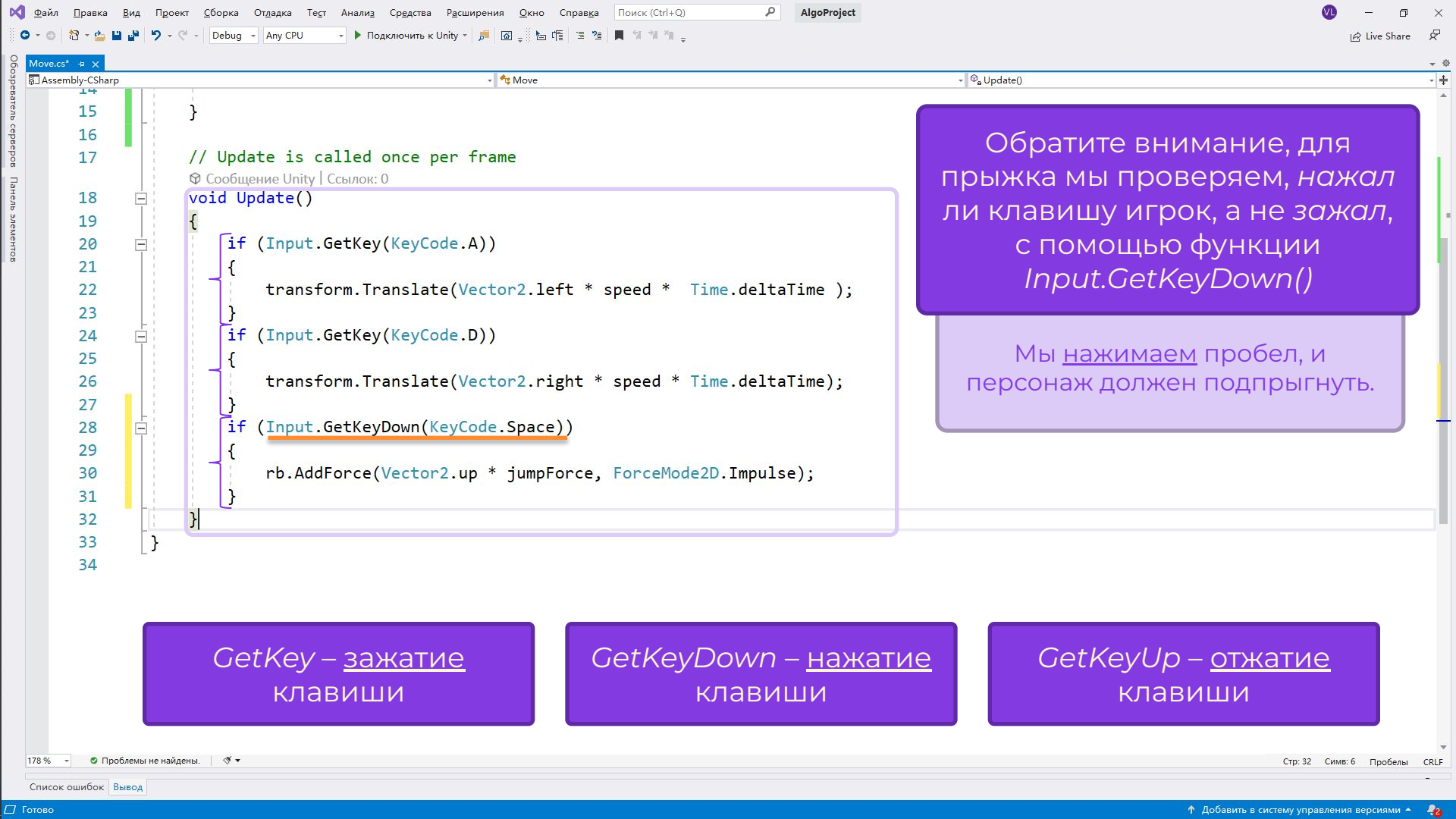
Для того, чтобы дать ей значение, достаточно просто *перетащить* нужный нам компонент на пустое значение переменной.

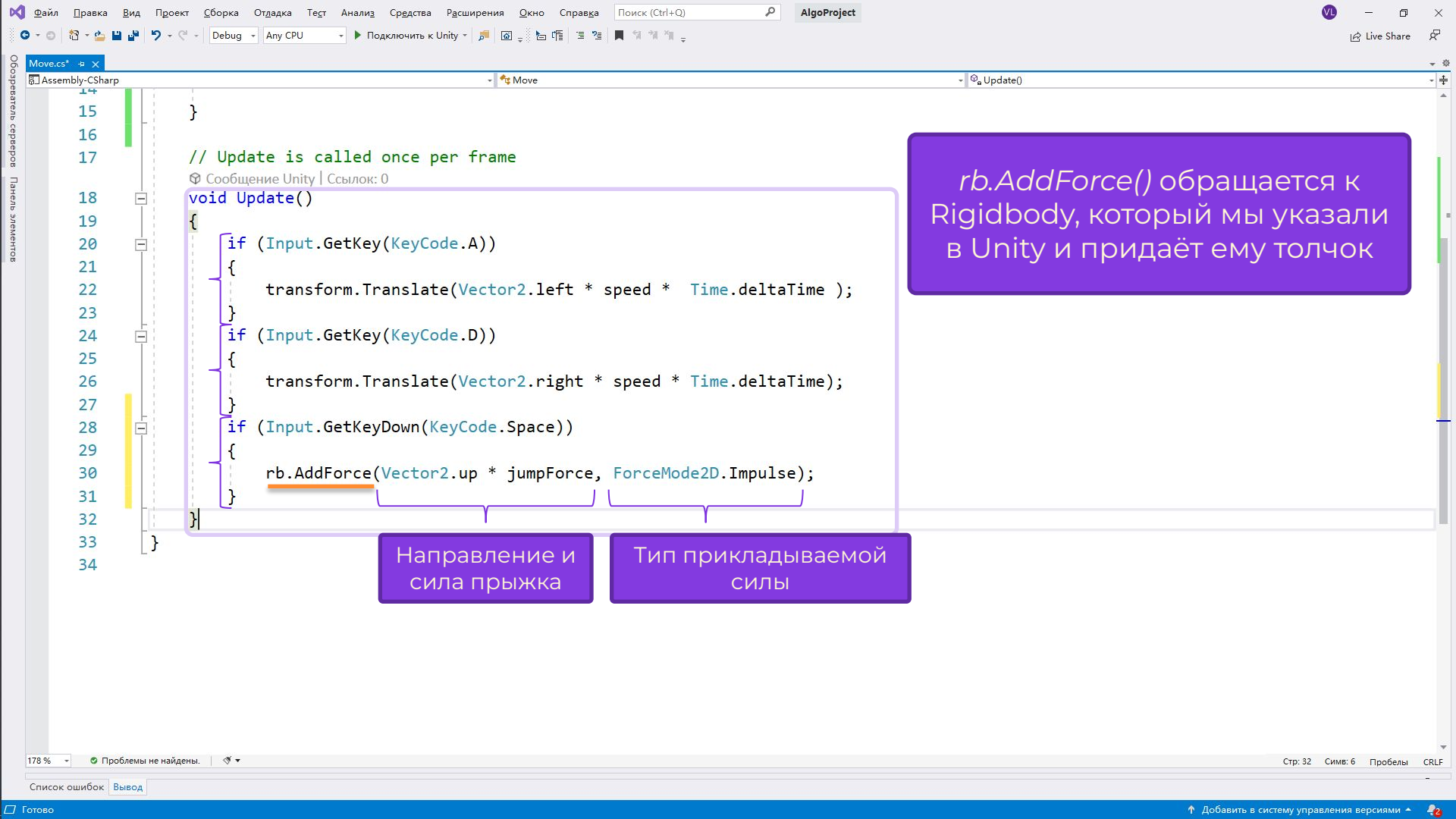



Файл Правка Вид Проект Сборка Отладка Тест Анализ Средства Расширения Окно Справка
Поиск (Ctrl+Q)
AlgoProject
Debug
Any CPU
Подключить к Unity
Live Share
Move.cs
Assembly-CSharp
Move
jumpForce
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
Скрипт Unity | Ссылок: 0
public class Move : MonoBehaviour
{
public float speed = 5;
public float jumpForce = 4;
public Rigidbody2D rb;
Сообщение Unity | Ссылок: 0
private void Start()
{
}
// Update is called once per frame
Сообщение Unity | Ссылок: 0
void Update()
{
if (Input.GetKey(KeyCode.A))
{
transform.Translate(Vector2.left * speed * Time.deltaTime);
}
if (Input.GetKey(KeyCode.D))
{
transform.Translate(Vector2.right * speed * Time.deltaTime);
}
}
}
Сделали? Тогда самое время дополнить наш скрипт.
Для начала добавим переменную, которая будет регулировать силу прыжка.
162 %
Проблемы не найдены.
Стр: 8
Симв: 32
Пробелы
CRLF
Список ошибок
Вывод
Элементы сохранены
Добавить в систему управления версиями


Файл Правка Вид Проект Сборка Отладка Тест Анализ Средства Расширения Окно Справка
Поиск (Ctrl+Q)
AlgoProject
Debug
Any CPU
Подключить к Unity
Live Share
Move.cs*
Assembly-CSharp
Move
Update()
}
// Update is called once per frame
Сообщение Unity | Ссылок: 0
void Update()
{
if (Input.GetKey(KeyCode.A))
{
transform.Translate(Vector2.left * speed * Time.deltaTime);
}
if (Input.GetKey(KeyCode.D))
{
transform.Translate(Vector2.right * speed * Time.deltaTime);
}
if (Input.GetKeyDown(KeyCode.Space))
{
rb.AddForce(Vector2.up * jumpForce, ForceMode2D.Impulse);
}
}
}
И после этого пишем код для реализации прыжка. Записывать мы его будем в функции Update(), там же, где и создавали перемещение.
Не запутайтесь с уровнями вложенности!
178 %
Проблемы не найдены.
Стр: 32
Симв: 6
Пробелы
CRLF
Список ошибок
Вывод
Готово
Добавить в систему управления версиями


Файл Правка Вид Проект Сборка Отладка Тест Анализ Средства Расширения Окно Справка
Поиск (Ctrl+Q)
AlgoProject
Debug
Any CPU
Подключить к Unity
Live Share
Move.cs*
Assembly-CSharp
Move
Update()
}
// Update is called once per frame
Сообщение Unity | Ссылок: 0
void Update()
{
if (Input.GetKey(KeyCode.A))
{
transform.Translate(Vector2.left * speed * Time.deltaTime);
}
if (Input.GetKey(KeyCode.D))
{
transform.Translate(Vector2.right * speed * Time.deltaTime);
}
if (Input.GetKeyDown(KeyCode.Space))
{
rb.AddForce(Vector2.up * jumpForce, ForceMode2D.Impulse);
}
}
}
Обратите внимание, для прыжка мы проверяем, нажал ли клавишу игрок, а не зажал, с помощью функции Input.GetKeyDown()
Не запутайтесь с уровнями вложенности!
GetKey – зажатие клавиши
GetKeyDown – нажатие клавиши
GetKeyUp – отжатие клавиши
Проблемы не найдены.
Стр: 32 Симв: 6 Пробелы CRLF
Список ошибок Вывод
Готово
Добавить в систему управления версиями

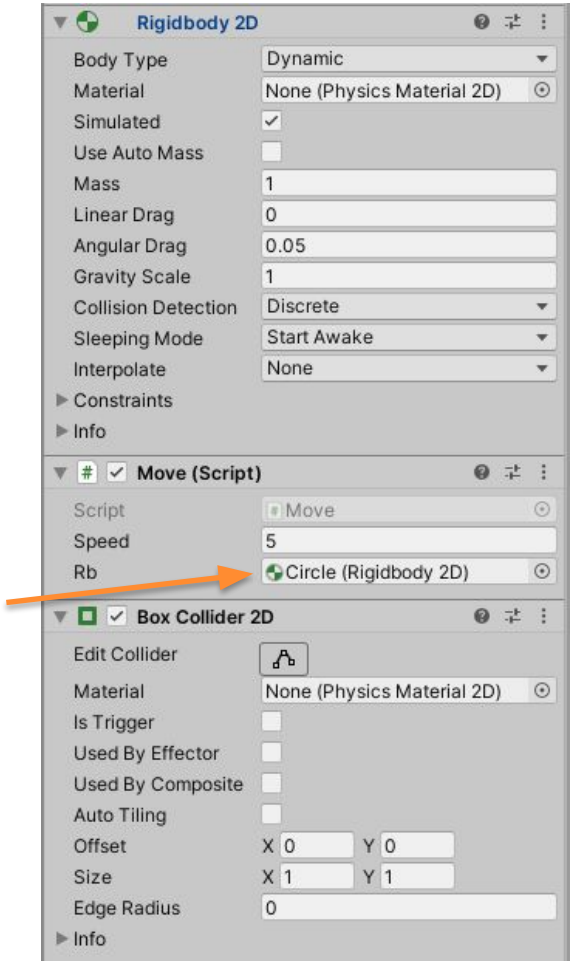

Файл Правка Вид Проект Сборка Отладка Тест Анализ Средства Расширения Окно Справка
Поиск (Ctrl+Q)
AlgoProject
Debug
Any CPU
Подключить к Unity
Live Share
Move.cs*
Assembly-CSharp
Move
Update()
}
// Update is called once per frame
Сообщение Unity | Ссылок: 0
void Update()
{
if (Input.GetKey(KeyCode.A))
{
transform.Translate(Vector2.left * speed * Time.deltaTime);
}
if (Input.GetKey(KeyCode.D))
{
transform.Translate(Vector2.right * speed * Time.deltaTime);
}
if (Input.GetKeyDown(KeyCode.Space))
{
rb.AddForce(Vector2.up * jumpForce, ForceMode2D.Impulse);
}
}
}
Обратите внимание, для прыжка мы проверяем, нажал ли клавишу игрок, а не зажал, с помощью функции Input.GetKeyDown()
Мы нажимаем пробел, и персонаж должен подпрыгнуть.
GetKey – зажатие клавиши
GetKeyDown – нажатие клавиши
GetKeyUp – отжатие клавиши
Проблемы не найдены.
Стр: 32 Симв: 6 Пробелы CRLF
Список ошибок Вывод
Готово
Добавить в систему управления версиями


Файл Правка Вид Проект Сборка Отладка Тест Анализ Средства Расширения Окно Справка
Поиск (Ctrl+Q)
AlgoProject
Debug
Any CPU
Подключить к Unity
Live Share
Move.cs*
Assembly-CSharp
Move
Update()
}
// Update is called once per frame
Сообщение Unity | Ссылок: 0
void Update()
{
if (Input.GetKey(KeyCode.A))
{
transform.Translate(Vector2.left * speed * Time.deltaTime);
}
if (Input.GetKey(KeyCode.D))
{
transform.Translate(Vector2.right * speed * Time.deltaTime);
}
if (Input.GetKeyDown(KeyCode.Space))
{
rb.AddForce(Vector2.up * jumpForce, ForceMode2D.Impulse);
}
}
}
rb.AddForce() обращается к Rigidbody, который мы указали в Unity и придаёт ему толчок
Направление и сила прыжка
Тип прикладываемой силы
178 %
Проблемы не найдены.
Стр: 32
Симв: 6
Пробелы
CRLF
Список ошибок
Вывод
Готово
Добавить в систему управления версиями

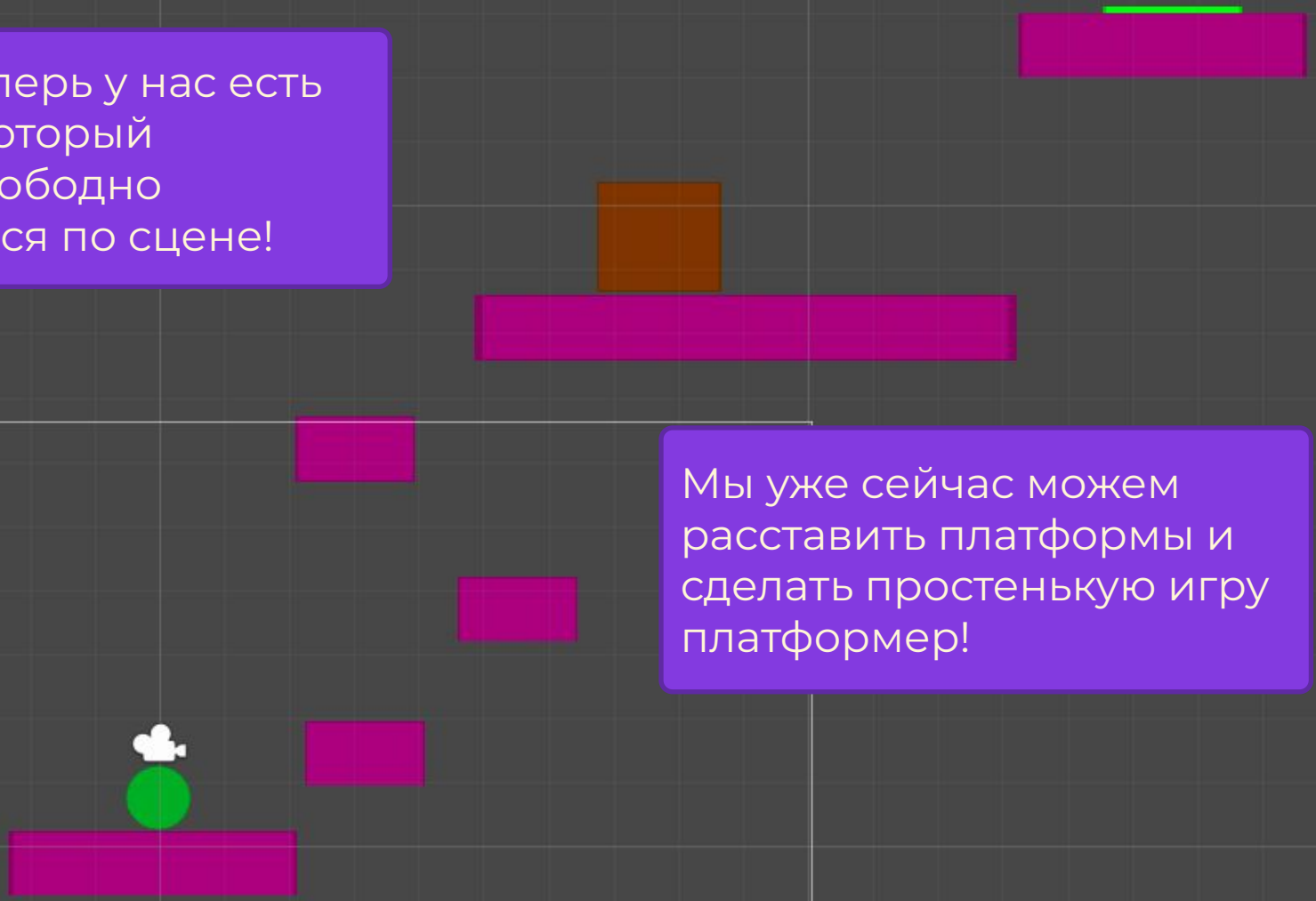
Реализация прыжка

Если вы сделали все правильно, то ваш персонаж должен прыгать, при нажатии клавиши *Space*

Не забудьте убедиться, что вы дали значение переменной *rb*



Отлично! Теперь у нас есть персонаж, который способен свободно перемешаться по сцене!



Мы уже сейчас можем расставить платформы и сделать простенькую игру платформер!

Немного о префабах

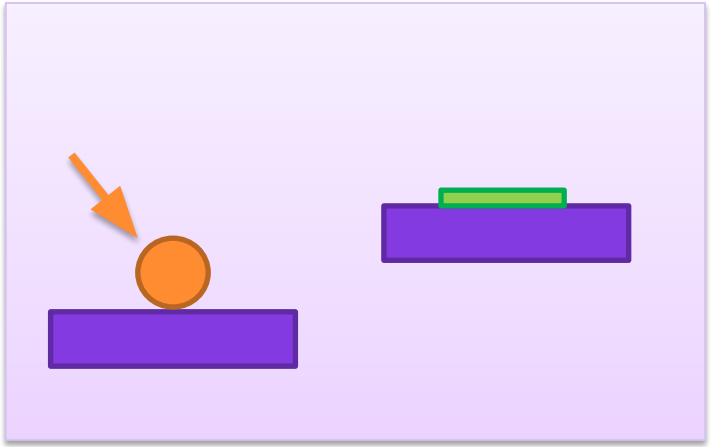
Последнее, что мы сегодня с вами узнаем – это что такое *Префабы*.

Префаб – это копия какого-либо объекта, сохраняющая все свойства и настройки, которую можно многократно использовать в **Unity**

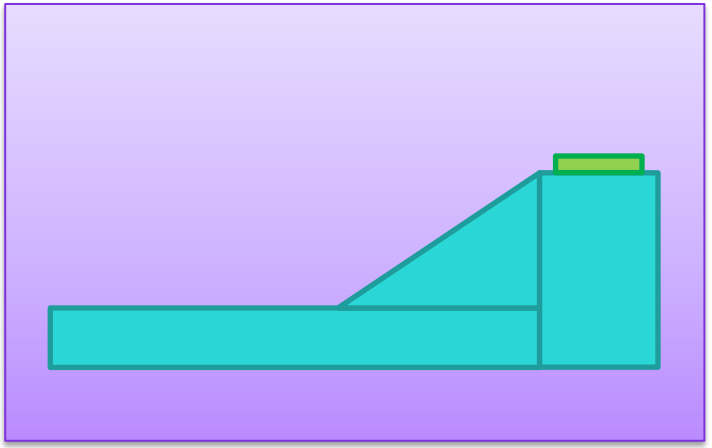
Немного о префабах

Наглядно: Представьте, что у нас есть две сцены, на которых должен быть один и тот же объект.

Сцена 1



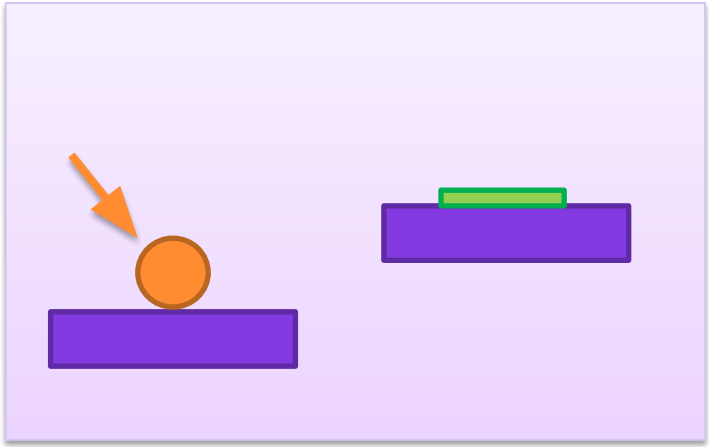
Сцена 2



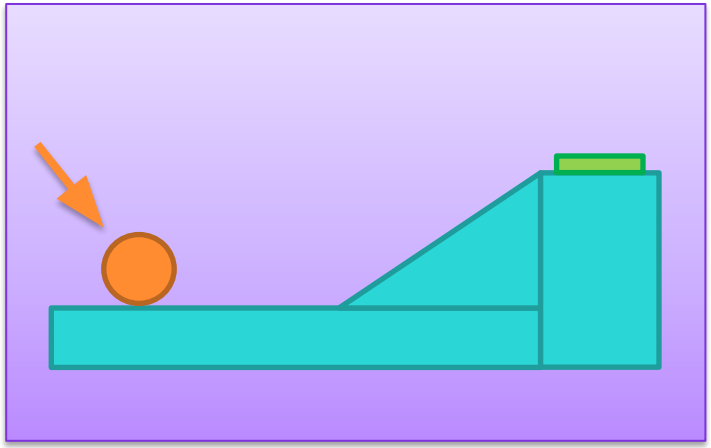
Немного о префабах

Если сохранить его в *префаб*, то мы сможем добавить такой же объект и на другие сцены

Сцена 1

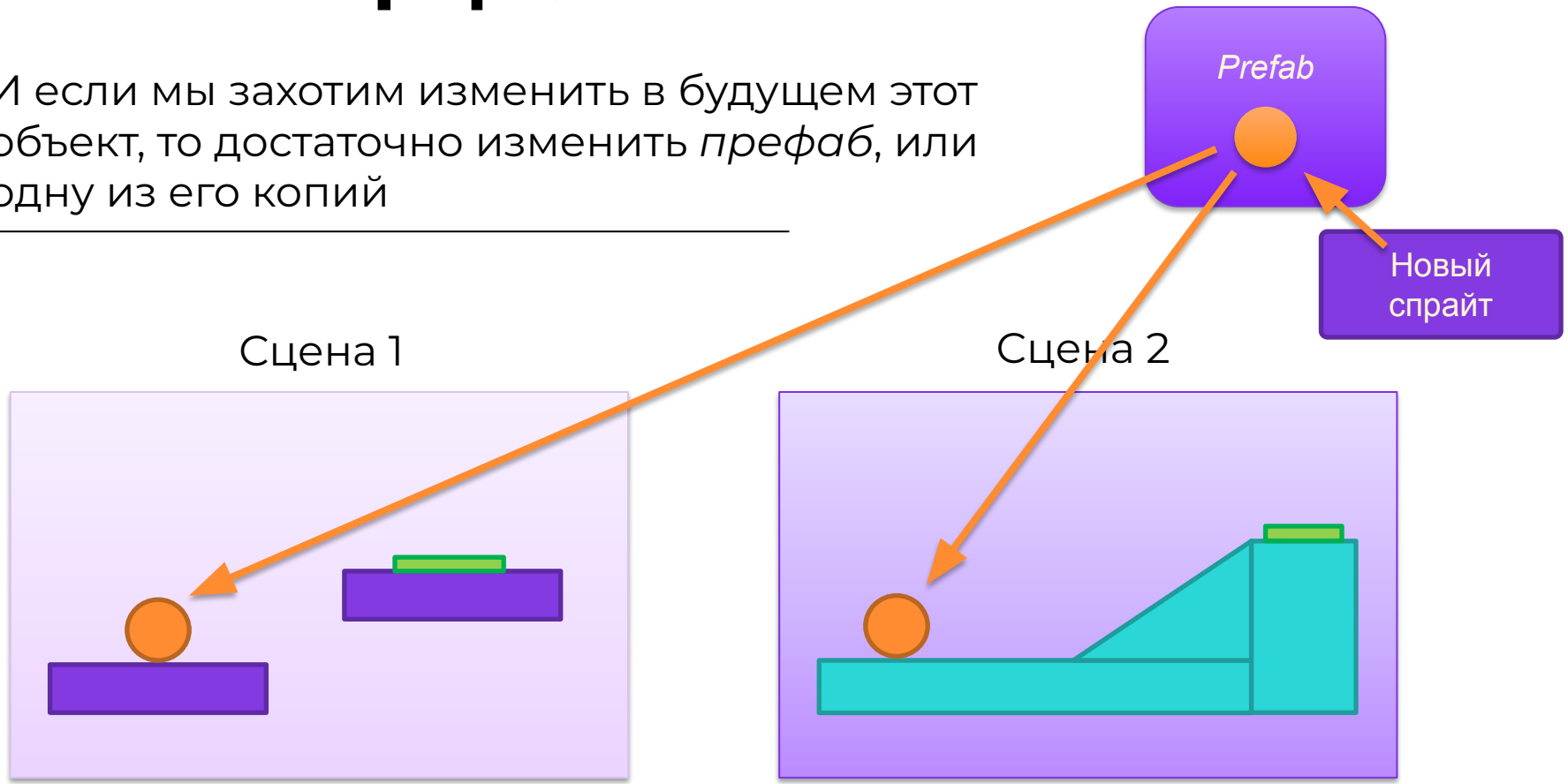


Сцена 2



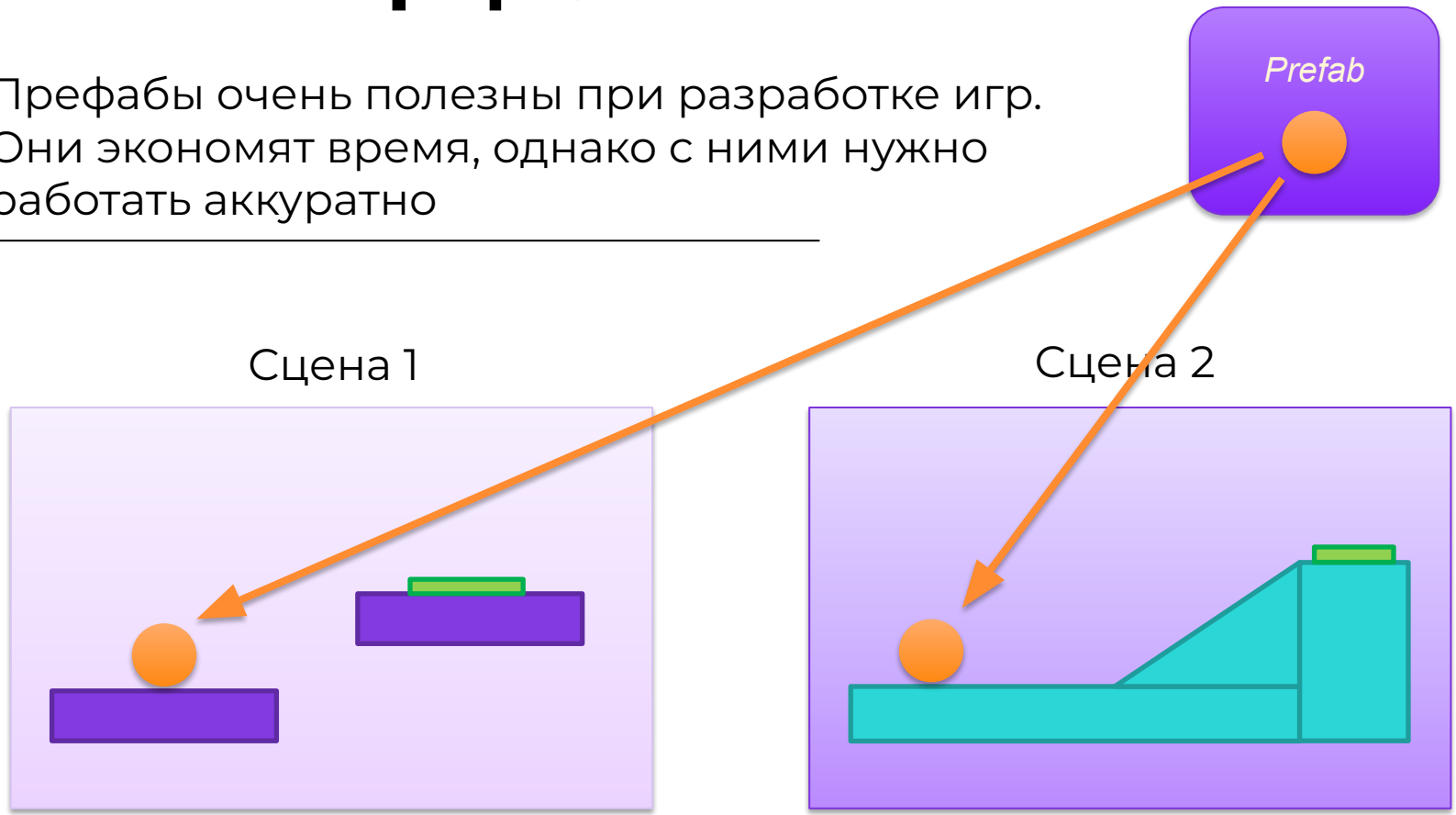
Немного о префабах

И если мы захотим изменить в будущем этот объект, то достаточно изменить *префаб*, или одну из его копий



Немного о префабах

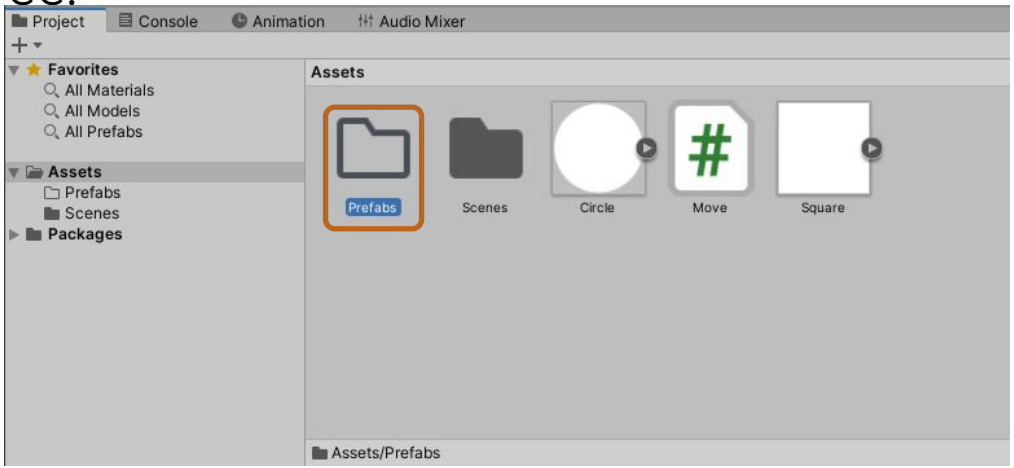
Префабы очень полезны при разработке игр. Они экономят время, однако с ними нужно работать аккуратно



Немного о префабах

Обычно, для префабов создают отдельную папку в проекте, поэтому давайте так и сделаем

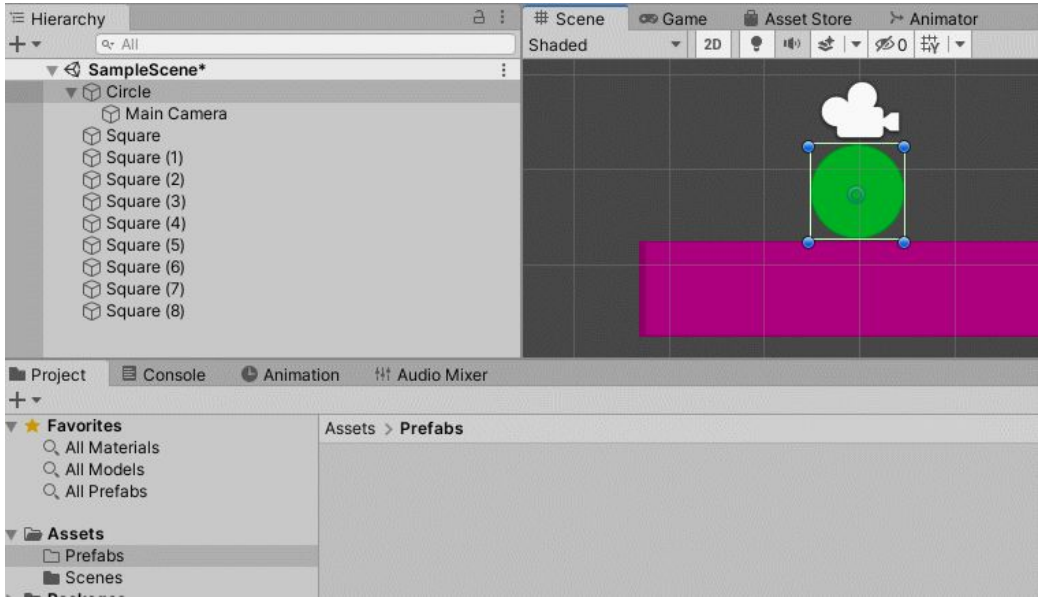
Создайте папку *Prefabs* в вашем проекте и откройте её.



Немного о префабах

Для создания префаба необходимо перетащить нужный объект из окна Иерархии в окно Проекта.

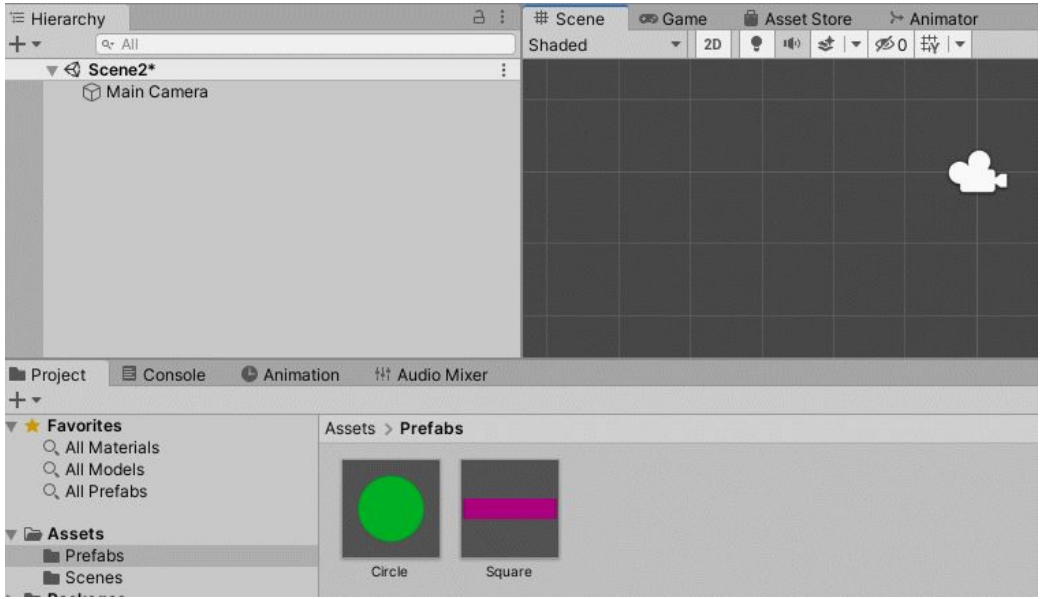
Попробуйте создать префаб нашего персонажа.



Немного о префабах

Для повторного использования просто достаньте эти префабы на сцену.

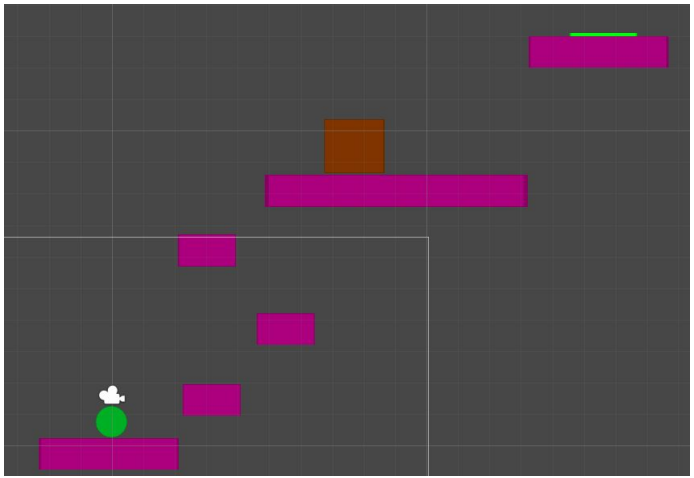
Если у вас 2 камеры на сцене, вам нужно будет удалить лишнюю.



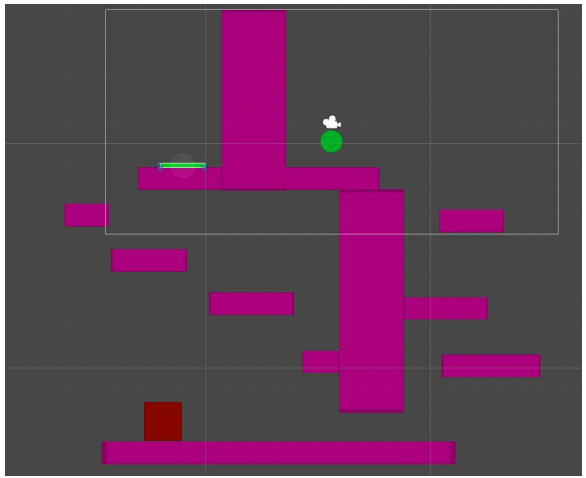
Конец урока

В будущем мы ещё столкнёмся с префабами и научимся редактировать их, а пока давайте с вами в качестве домашнего задания создадим вторую сцену и с помощью префабов обустроим её!

Сцена 1



Сцена 2



До встречи на следующем уроке!

Спасибо!