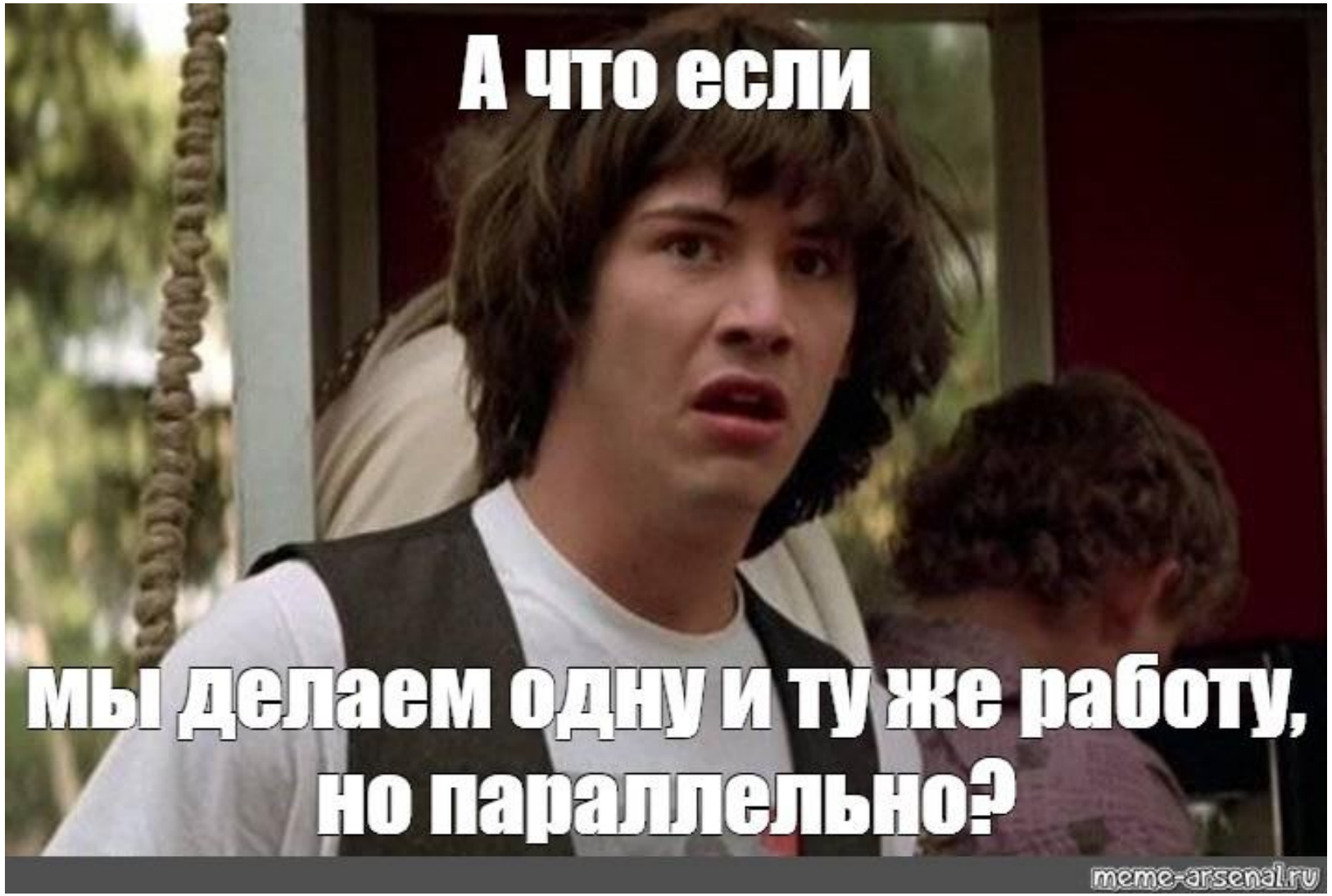
Корутины (**Coroutine**)

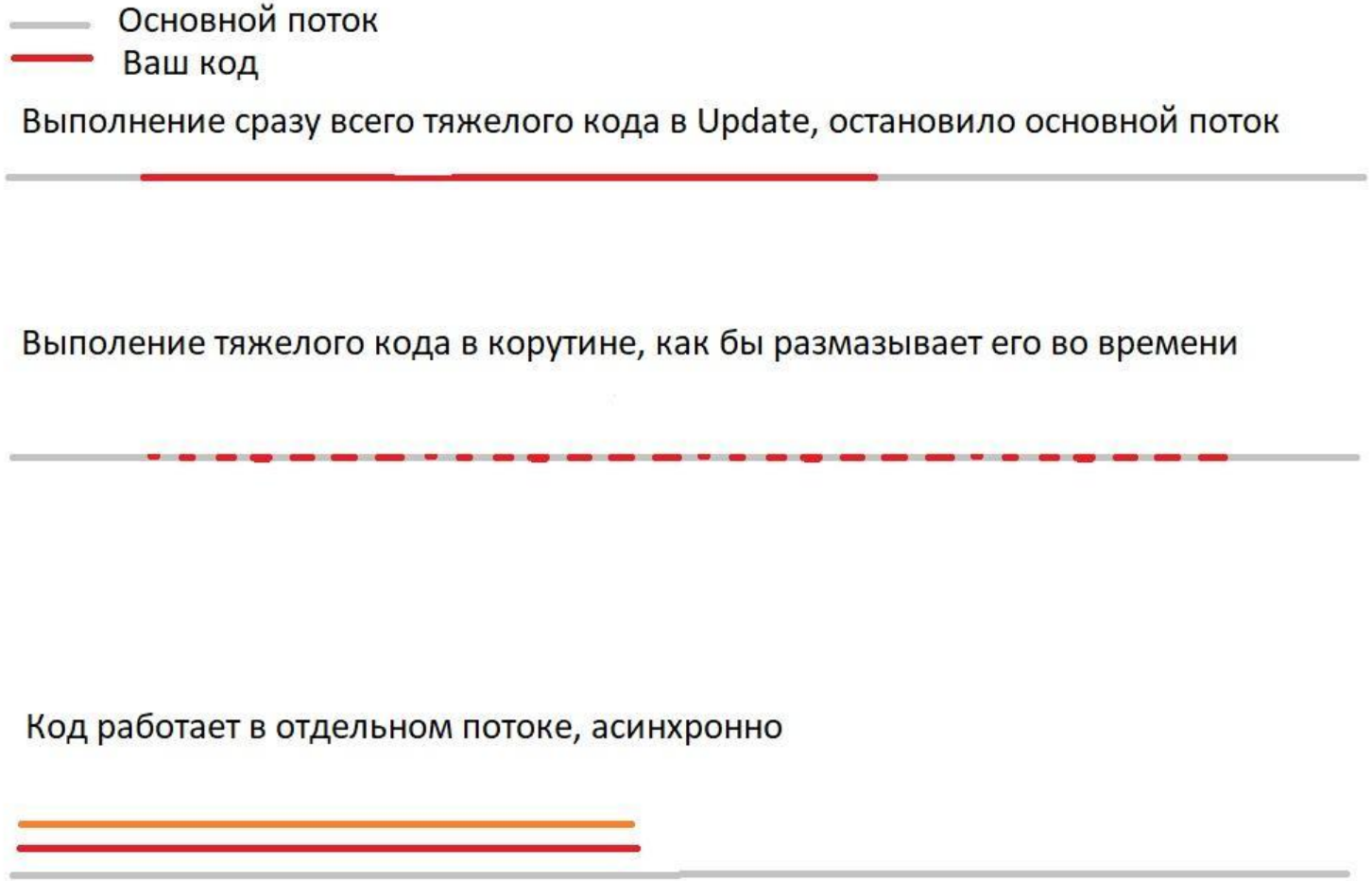
- Каждый скрипт Unity имеет две важные функции:Start() и Update().
- В то время как первый вызывается, когда объект включается после создания, второй вызывается в каждом кадре. По замыслу следующий кадр не может начаться до тех пор, пока Update не прекратил свою работу. Это вводит сильное конструктивное ограничение: Update() не может легко моделировать события, которые длятся более одного кадра.

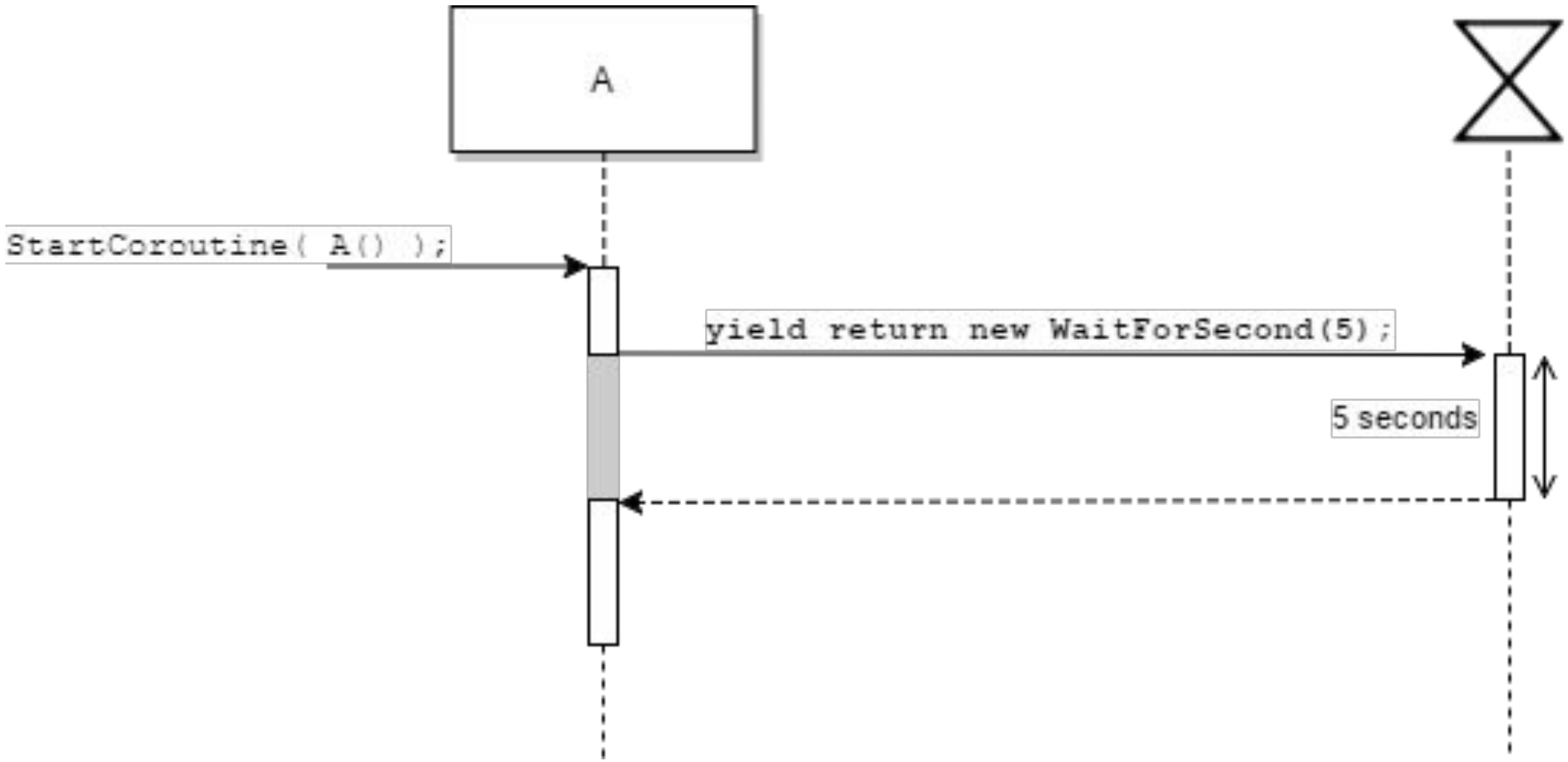
События, происходящие в нескольких кадрах (такие как анимация, диалоги, ожидание и т. д.), кодировать сложнее. Это потому, что их логика не может быть записана в последовательном потоке. Он должен быть фрагментирован, распределен по нескольким кадрам. Это часто приводит к тому, что код не только сложнее написать, но и сложнее поддерживать.


А что если
мы делаем одну и ту же работу,
но параллельно?
meme-arsenal.ru

Корутины (Coroutines, сопрограммы) в Unity — простой и удобный способ запускать функции, которые должны работать параллельно в течение некоторого времени.

Корутины выполняются параллельно с основным кодом, параллельно не значит асинхронно. То есть, если вы напишите свой "тяжелый" код в кучу корутин которые будут работать параллельно, программа **работать быстрее не будет**.




A
StartCoroutine(A());
yield return new WaitForSecond(5);
5 seconds

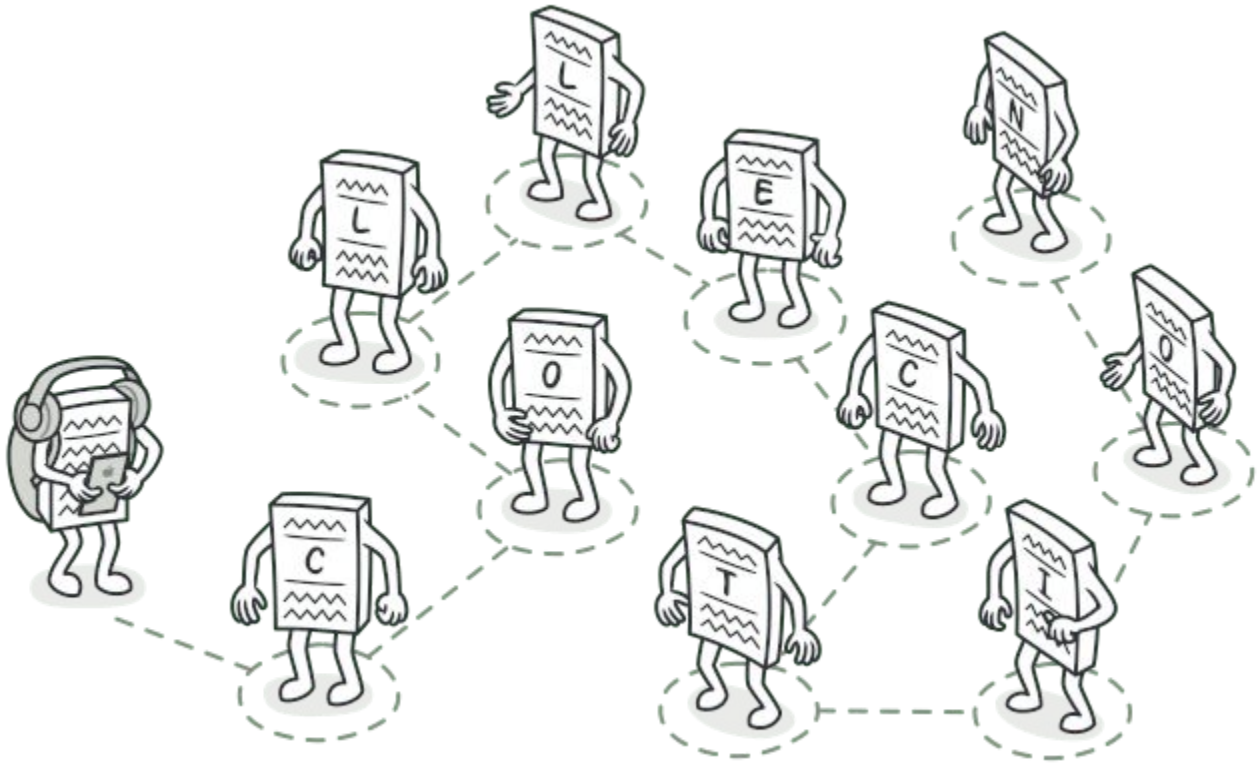
Пример: корутина, которая уменьшает яркость объекта

```
IEnumerator Fade() {
for (float ft = 1f; ft >= 0; ft -= 0.1f) {
Color c = renderer.material.color;
c.a = ft;
renderer.material.color = c;
yield return null;}
}
void Update() {
 if (Input.GetKeyDown("f")) {
 StartCoroutine("Fade"); }
}
```

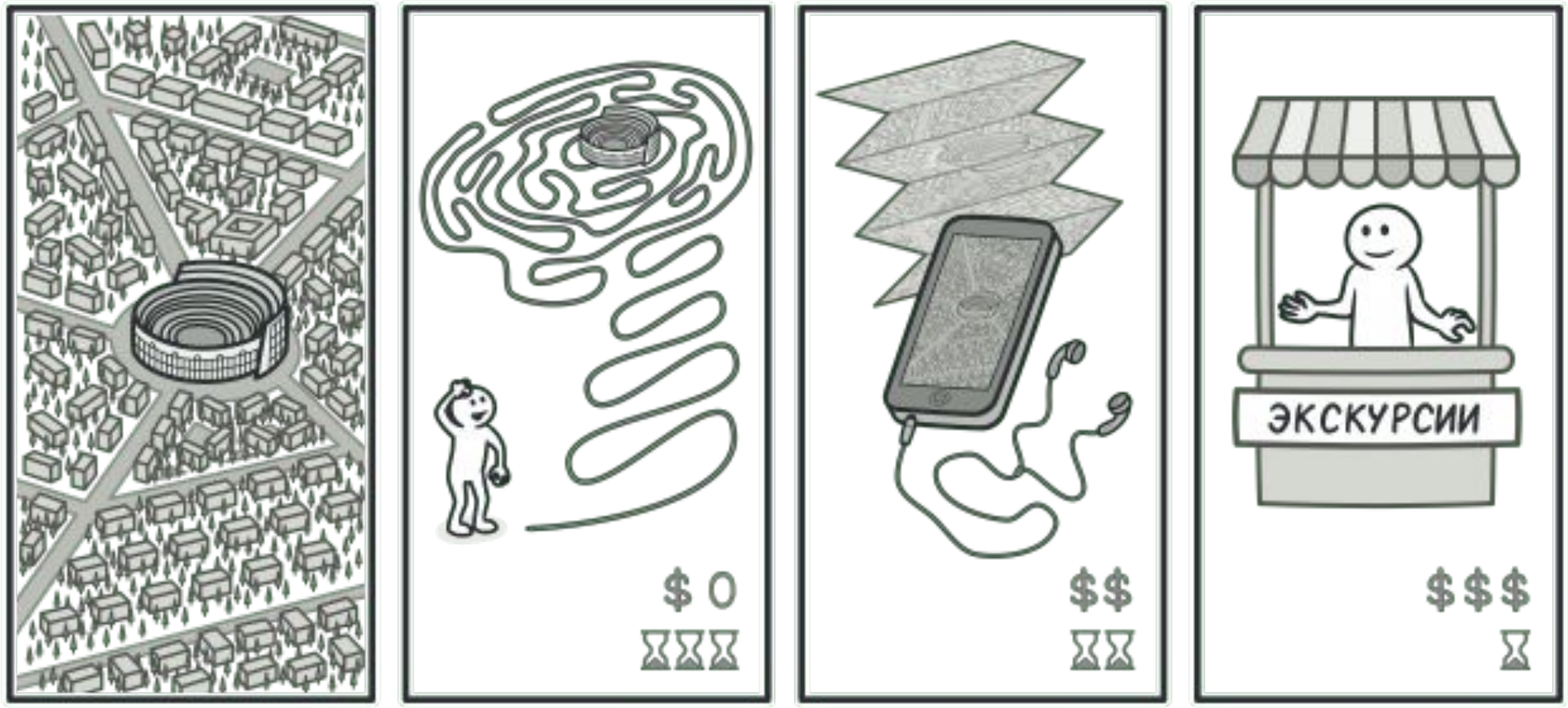
```
IEnumerator YouCoroutin(){
//описание функции
yield return …. ;
//здесь несколько вариантов}
```

Корутины представляют собой простые C# итераторы, возвращающие **IEnumerator** и использующие ключевое слово **yield**.

Итератор — это поведенческий паттерн проектирования, который даёт возможность последовательно обходить элементы составных объектов, не раскрывая их внутреннего представления.



Аналогия из жизни –прогулка по не знакомому городу



Yield, что это?

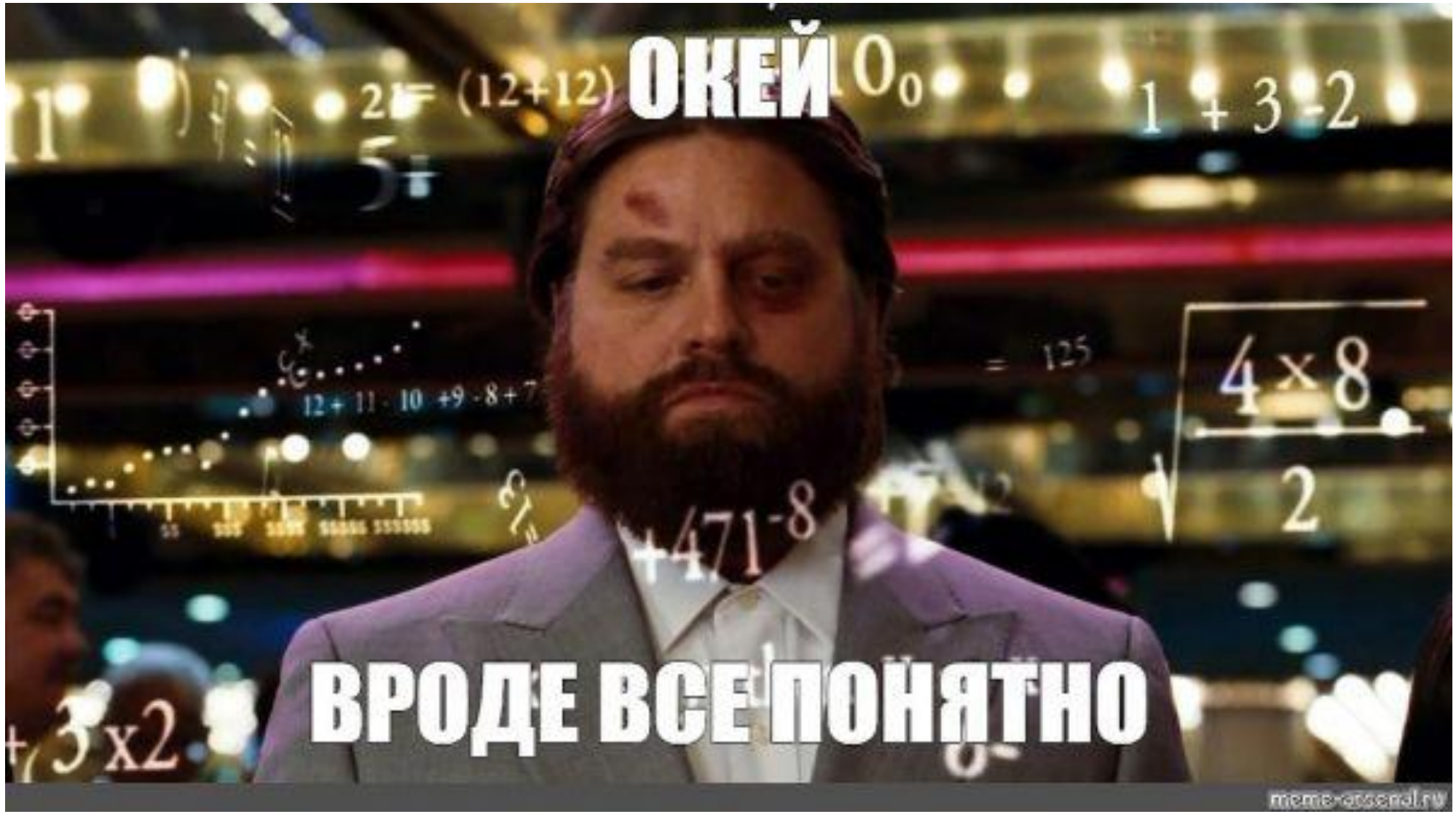
- Эта команда, отдает процессорное время основному потоку и продолжает выполнение корутины с этого места когда произойдет необходимое событие

При использовании **WaitForSeconds** создается долго существующий объект в памяти (управляемой куче), поэтому его использование в быстрых циклах может быть плохой идеей.

Как завершить корутину?

Иногда корутину нужно завершить преждевременно. Для это есть несколько путей.

- **StopAllCoroutines()**
- **StopCoroutine(«YourCoroutine»)**
- Уничтожить родительский GameObject


ОКЕЙ
ВРОДЕ ВСЕ ПОНЯТНО
meme-arsenal.ru

- https://refactoring.guru/ru/design-patterns/iterator
- https://professorweb.ru/my/csharp/charp_theory/level12/12_20.php
- https://docs.microsoft.com/ru-ru/dotnet/csharp/iterators
- https://you-hands.ru/2020/11/25/korutiny-v-unity-chto-eto-i-kak-ispolzovat/
- https://www.
- alanzucconi.com/2017/02/15/ne
- ted-coroutines-in-unity/
- https://docs.unity3d.com/ru/2019.4/Manual/Coroutines.html

Почирпапавптать:

Варианты

1.Продолжить после следующего FixedUpdate

```
yield return new WaiЗадание.
tForFixedUpdate();
```

2. Продолжитv

```
yield return new WaitForEndOfFrame();
```

4. Продолжить после другой корутины:

```
yield return StartCoroutine(AnotherCoroutine());
```

5. После текущего Update

```
yield return null;
```

6. По завершении скачивания

```
yield return new WWW(someLink);
```

7. Завершить корутину

```
yield return break;
```

Задание.

- Проверить заготовку игры по чеклисту с прошлого занятия
- Написать характеристику героя по схеме (см картинку в гугл диске)
- Написать заготовку CharecterController для ГГ, создать все необходимые переменные (минимум: здоровье, выносливость, ключи), а так же написать реакцию на встречу с монстрами и паверапами(см тему взаимодействие с объектами)
- Подумать, где в нашем коде мы можем использовать корутины