Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования «Кубанский государственный университет»

Кафедра информационных технологий

**ОТЧЕТ**

о выполнении лабораторной работы №7

по дисциплине «Технологии проектирования программного обеспечения»

Выполнил: ст. гр. 4ИТ

Хижний Е.Г.

Проверил: доц. каф. ИТ

Полетайкин А.Н.

Краснодар

2021

**Тема**: Тестирование программной системы

**Цель**: Освоение методики тестирования разработанной программной системы в ручном режиме.

**Задание**

1. Выполнить системное пользовательское тестирование работоспособности ПС посредством воздействия на её интерфейсную часть, разработанную при выполнении лабораторной работы №6. При возникновении несоответствий задокументировать их, произвести поиск логических ошибок, разработать тесты для их обнаружения, устранить ошибки.
2. Выполнить нагрузочное тестирование программы и оценить эффективность разработанных при выполнении [лабораторной работы №6](#_Лабораторная_работа_№4_) автоматизированных функций ПС, запросов к БД. В качестве критерия эффективности использовать время выполнения функции. Предварительно подготовить 5 массивов исходных данных на 10, 50, 100, 500 и 1000 записей. Построить графики зависимости времени вычислений от объема исходных данных.
3. Выполнить стрессовое тестирование ПС. В случае обнаружения некорректных ситуаций описать их, выявить причины и принять меры к их устранению.
4. Разработать модульный тест для проверки работоспособности кода, реализующего автоматизированные функции ПС. Результаты тестирования сохранить в файле test.log.

**Тема проекта**: Разработка имитационной модели ОДД на участке УДС крупного города.

**1 Нагрузочное тестирование**

В качестве нагрузочного теста был выполнен подсчет FPS(частота кадров в секунду) для модели с количеством объектов 1, 5, 10, 15, 20. Данные по тестированию приведены на рис. 1.

Рисунок 1 ­– График среднего FPS в зависимости от количества объектов

Для запуска тестирования необходимо открыть Window->General->Test Runner, как показанное на рис. 2

В Unity есть 2 вида тестирования (рис. 3):

1. Вкладка «PlayMode» используется для тестов, выполняемых в режиме Play (когда игра выполняется в реальном времени);
2. Вкладки «EditMode» выполняются вне режима Play.

Тесты для данной работы будут проводиться в режиме «PlayMode».



Рисунок 2 ­– Запуск тестирования в Unity

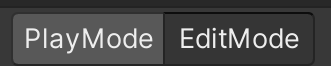


Рисунок 3 ­– Виды тестирования в Unity

Для создания тестирования перейдем на вкладку PlayMode, нажмем на кнопку “Create PlayMode Test Assembly Folder” (рис. 4), назовём ее Test 1 (рис. 5).

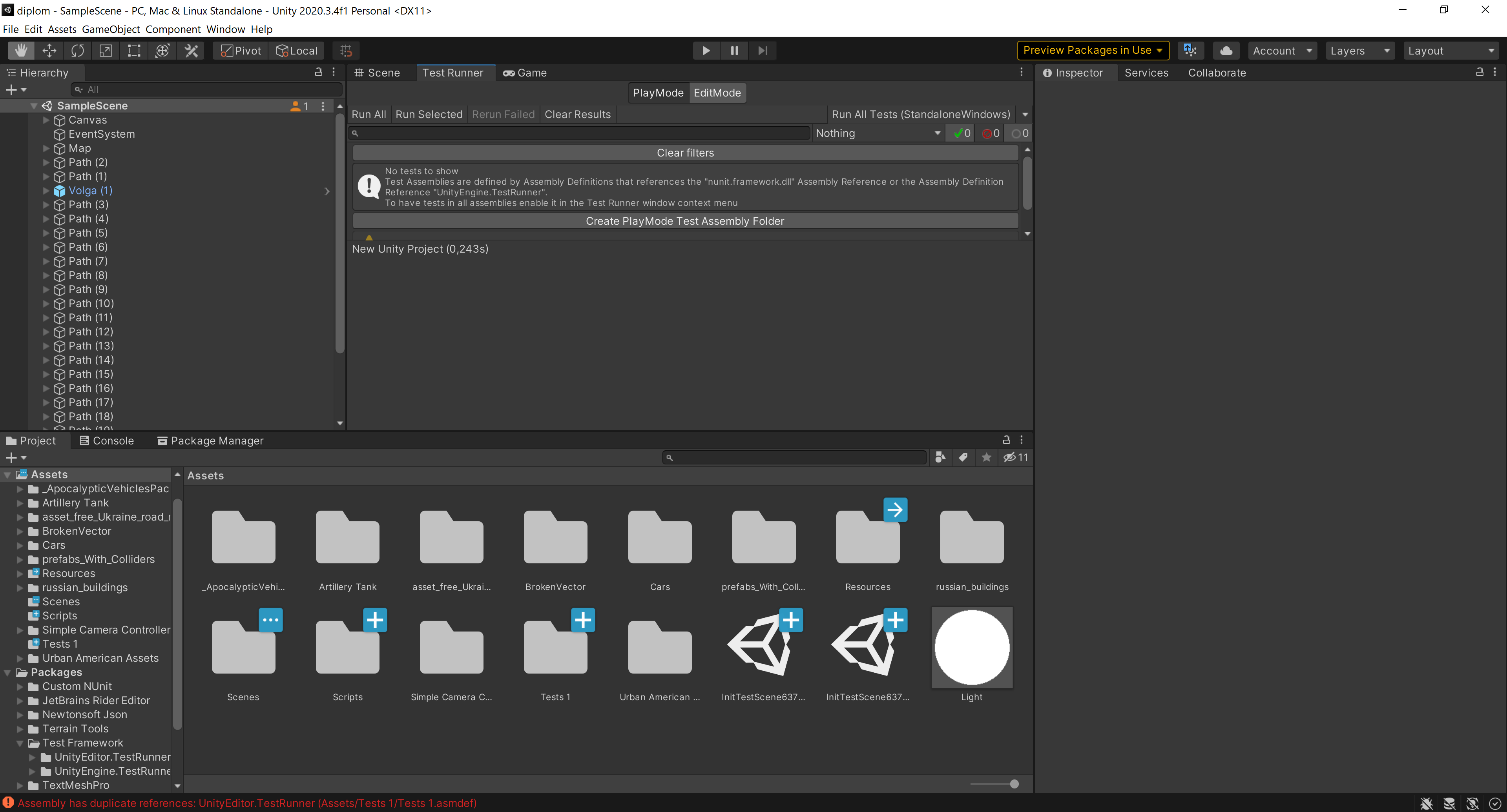


Рисунок 4 ­– Интерфейс создания теста

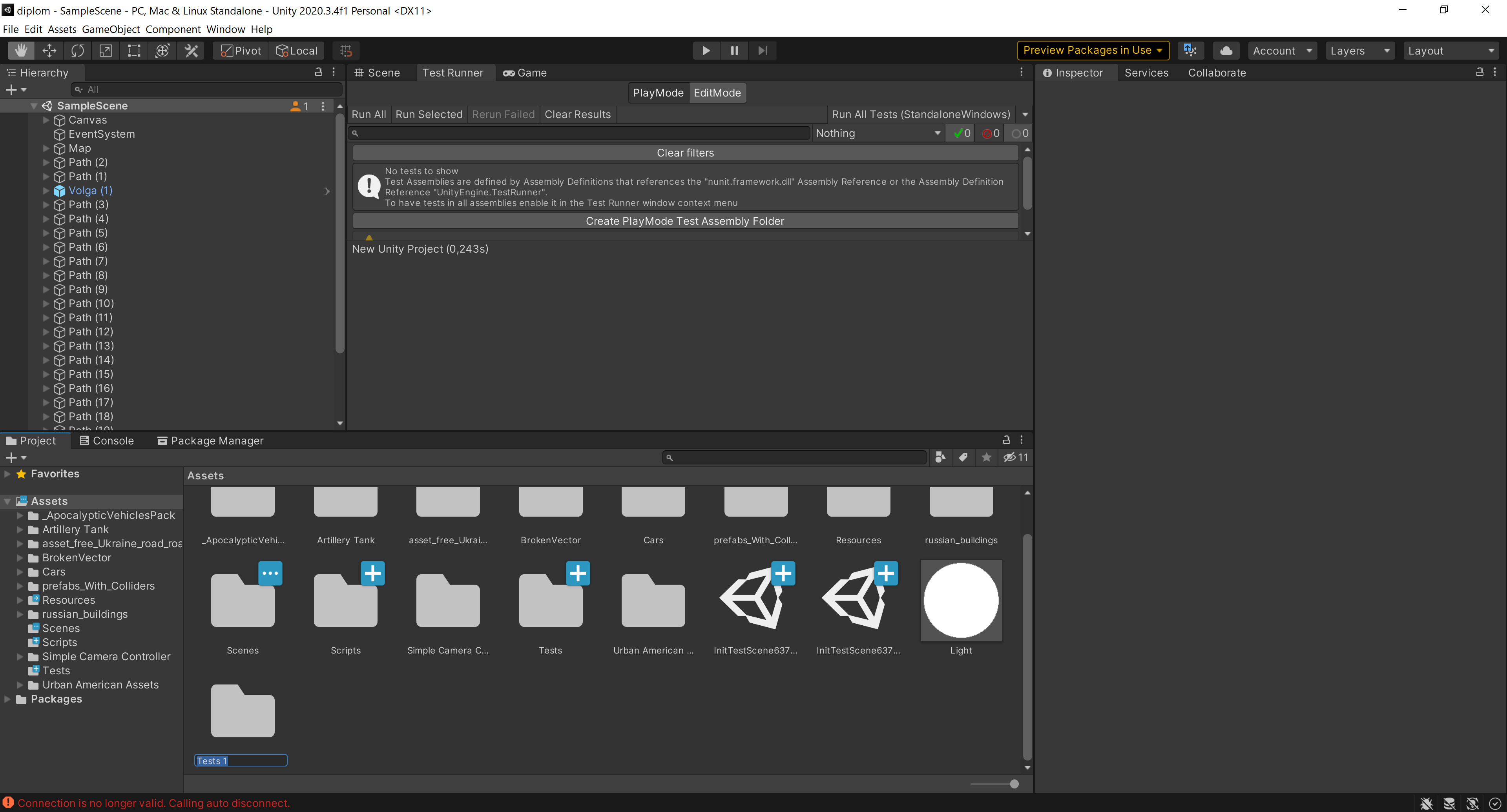


Рисунок 5 ­– Папка Test 1

В данной папке создался файл Tests1.asmdef. Это файл определения сборки (assembly definition file), который используется для того, чтобы показать Unity, где находятся зависимости файла теста. Это нужно, потому что код готового приложения содержится отдельно от тестового кода.

Изображение выглядит как текст, снимок экрана, монитор

Автоматически созданное описание

Рисунок 6 ­– Файл Test1.asmdef

Создадим сам файл тестов. Для этого ПКМ->Create->C# Script, как показано на рис. 7. Созданный файл на рис. 8.

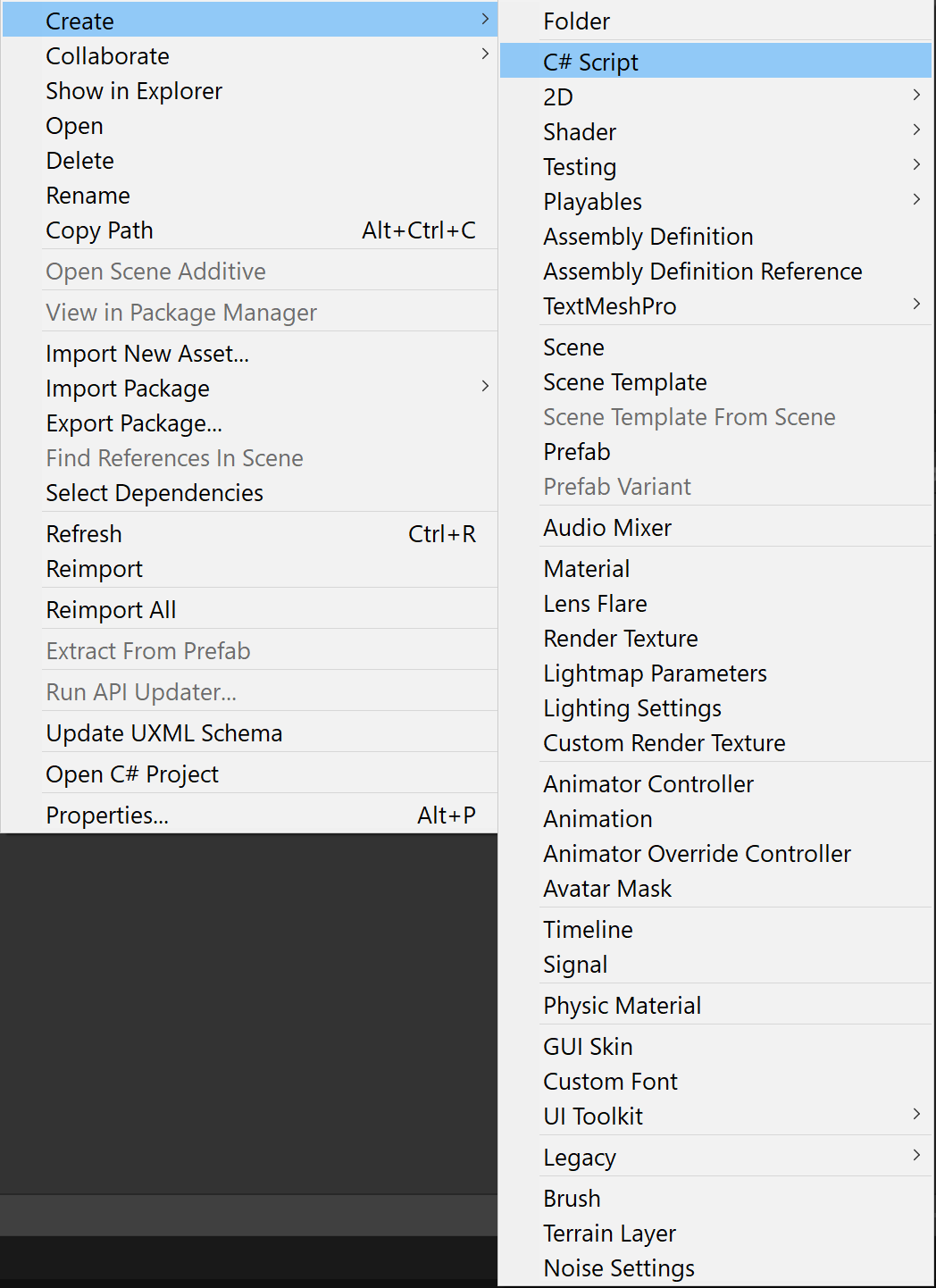


Рисунок 7 ­– Создание C# файлов

Изображение выглядит как текст, монитор, снимок экрана, внутренний

Автоматически созданное описание

Рисунок 8 ­– Файл NewBehaviourScript.cs

Чтобы код тестов имел доступ к классам игры, мы создадим сборку кода классов и зададим ссылку в сборке Tests1. Нажмите на папку Scripts, чтобы выбрать её. Нажмите правой клавишей на эту папку и выберите Create->Assembly Definition (рис. 9). Созданный файл представлен на рис. 10.

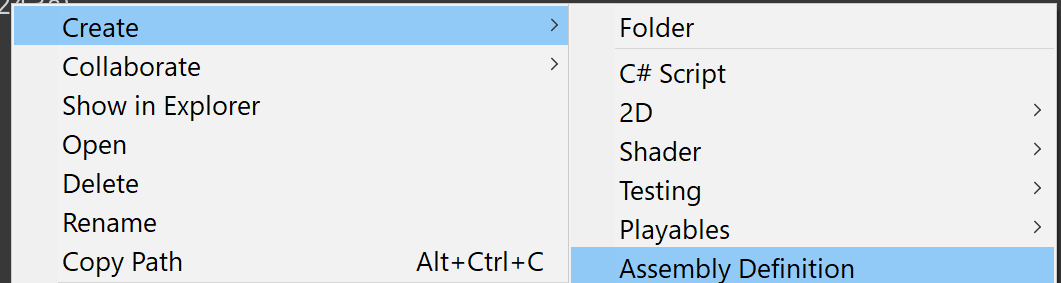


Рисунок 9 ­– Создание сборки класса

Изображение выглядит как текст, снимок экрана, монитор

Автоматически созданное описание

Рисунок 10 ­– Файл сборки кода

Добавим в сборку к файлу определения сборки. Для этого нажмем на файл Test1.asmdef. Найдем пункт меню «Assembly Definition References», нажмем на «+» и добавим файл сборки кода, как показано на рис. 11.

Изображение выглядит как текст, снимок экрана, монитор, черный

Автоматически созданное описание

Рисунок 11 ­– Добавление сборки кода к определению сборок

На этом подготовка тестирования завершена.

Для тестирования модуля «Моделирование» был разработан следующий скрипт тестирования, результат тестирования представлен на рис. 12, 13.

public class NewTestScript

{

    [UnityTest]

    // класс теста

    public IEnumerator createModel()

    {

        yield return null;

    }

}

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 12 ­– Результат 1 для теста модуля «Моделирование»

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 13 ­– Результат 2 для теста модуля «Моделирование»

Для тестирования модуля «Создание препятствий» был разработан следующий скрипт тестирования, результат тестирования представлен на рис. 14, 15.

public class Test

{

    [UnityTest]

    // класс теста

    public IEnumerator CarSpawnerTest()

    {

        // Создаем пустой объукт

        GameObject go = new GameObject();

        // Делаем его объектом класса CarSpawner

        CarSpawner cs = go.AddComponent<CarSpawner>();

        // Вызываем необходимую для теста функцию

        cs.spawnCar();

        yield return null;

    }

}

public class CarSpawner : MonoBehaviour

{

    // проверяемая функция

    public void spawnCar() {

        // Загружаем объект "авария"

        GameObject carPrefabs = (GameObject)Resources.Load("Prefabs/Crash 1", typeof(GameObject));

        //ставим на позицию

        carPrefabs.transform.position = GameObject.Find("Spawner").transform.position;

        //Создаем экземпляр

        Instantiate(carPrefabs);

    }

}

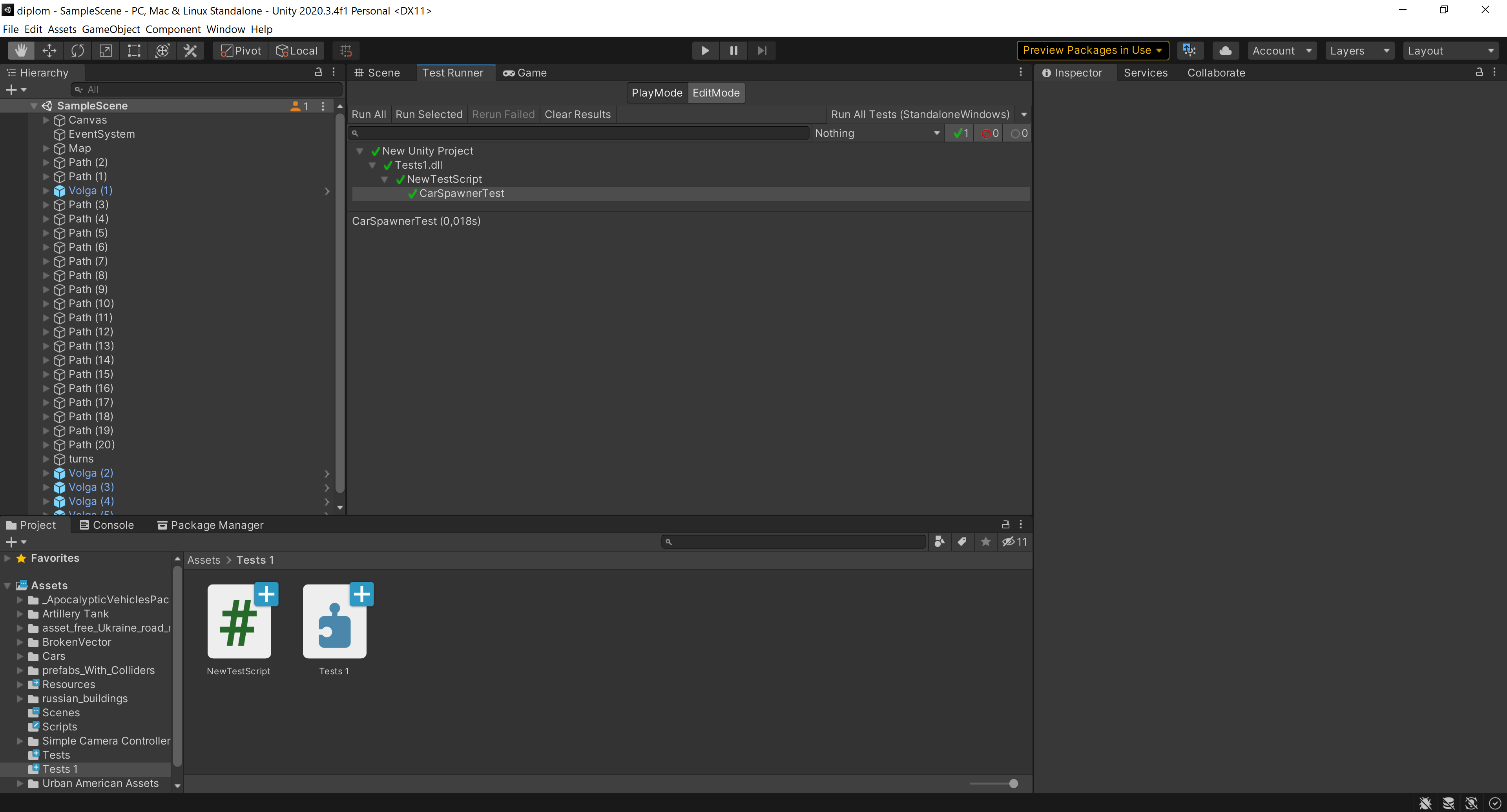


Рисунок 14 ­– Результат 1 для теста модуля «Создание препятствий»

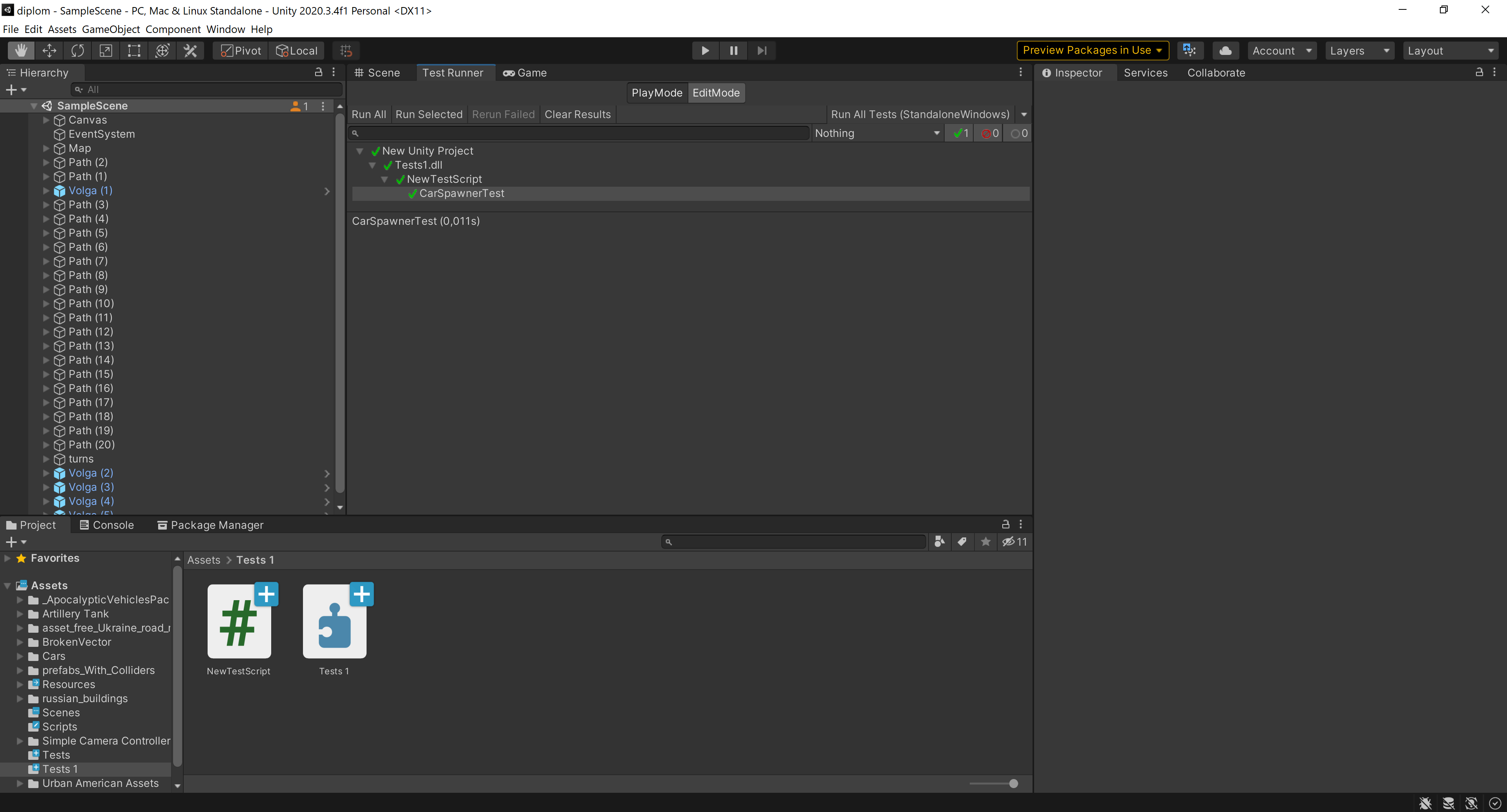


Рисунок 15 ­– Результат 2 для теста модуля «Создание препятствий»

Для тестирования модуля «Сохранение объектов» был разработан следующий скрипт тестирования, результат тестирования представлен на рис.16, 17.

public class NewTestScript

{

    [UnityTest]

    // класс теста

    public IEnumerator saveTest()

    {

        GameObject go = new GameObject();

        // Делаем его объектом класса AutoSave

        AutoSave cs = go.AddComponent<AutoSave>();

        // Вызываем необходимую для теста функцию

        cs.Save();

        yield return null;

    }

}

public class AutoSave : MonoBehaviour

{

    // Создаем массив объектов

    private GameObject[] cars;

    public void Save() {

    // Находим все объекты типа "Car"

        cars = GameObject.FindGameObjectsWithTag ("Car");

    // Инициализируем список для вывода в Json типа "Data"

        List<Data> datas = new List<Data>();

        foreach(GameObject car in cars){

    // Инициализируем объект типа "Data" для хранения информации об одном объекте

            Data data = new Data(car.name, car.transform.position.x, car.transform.position.y, car.transform.position.z, car.transform.rotation.x, car.transform.rotation.y, car.transform.rotation.z);

// Добавляем объект в список

            datas.Add(data);

        }

// Создаем строку JSON из списка типа "Data"

        string json = JsonUtility.ToJson(datas);

// Создаем выходной файл

        System.IO.File.WriteAllText(Application.persistentDataPath + "/data.json", json);

    }

}

[System.Serializable]

// Класс для хранения информации об одном объекте

public class Data{

    public string name;

    public float posX, posY, posZ, rotX, rotY, rotZ;

    public Data(string name, float posX, float posY, float  posZ, float  rotX, float rotY, float rotZ){

        this.name=name;

        this.posX=posX;

        this.posY=posY;

        this.posZ=posZ;

        this.rotX=rotX;

        this.rotY=rotY;

        this.rotZ=rotZ;

    }

}

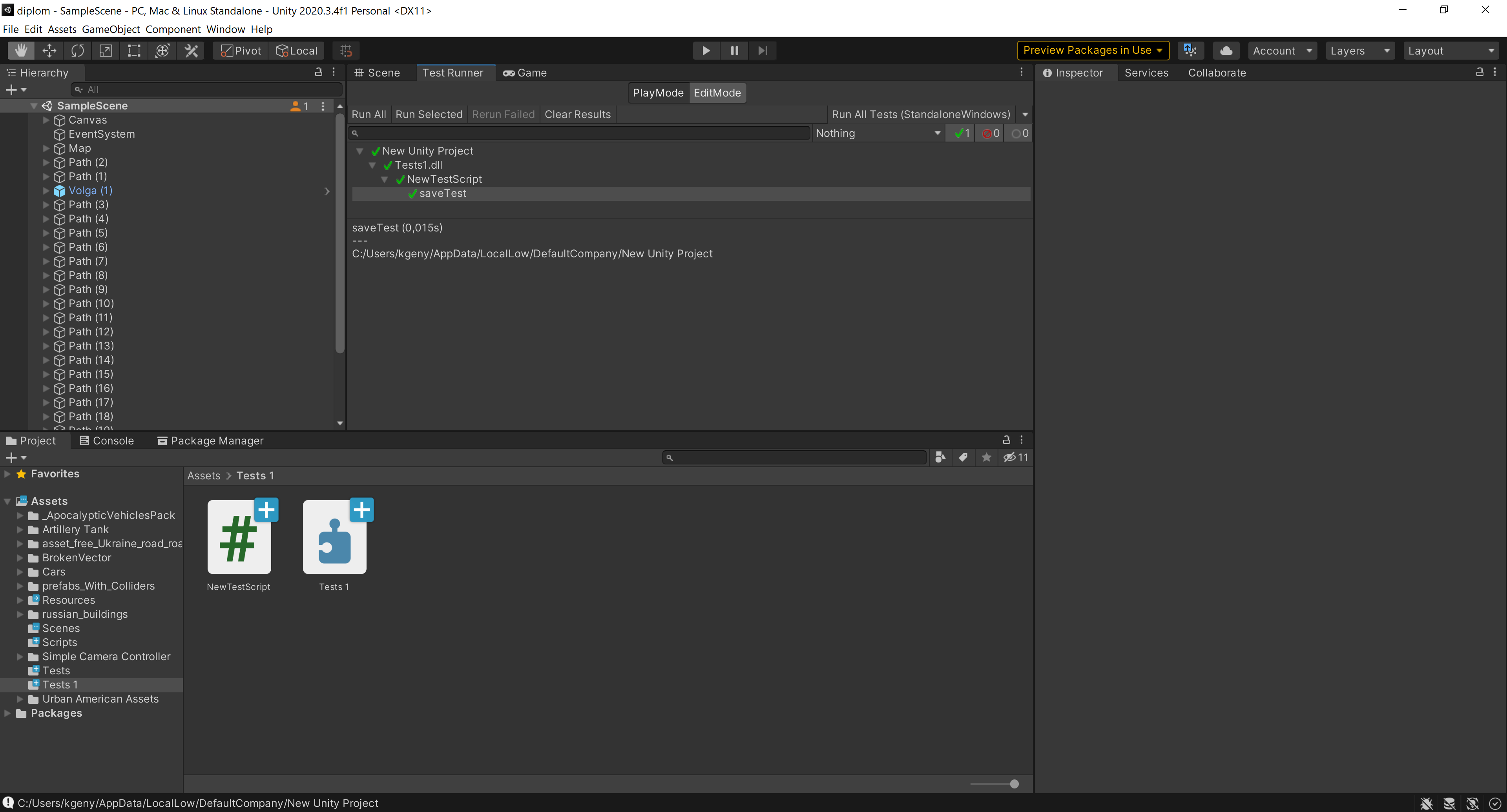


Рисунок 16 ­– Результат 1 для теста модуля «Сохранение объектов»

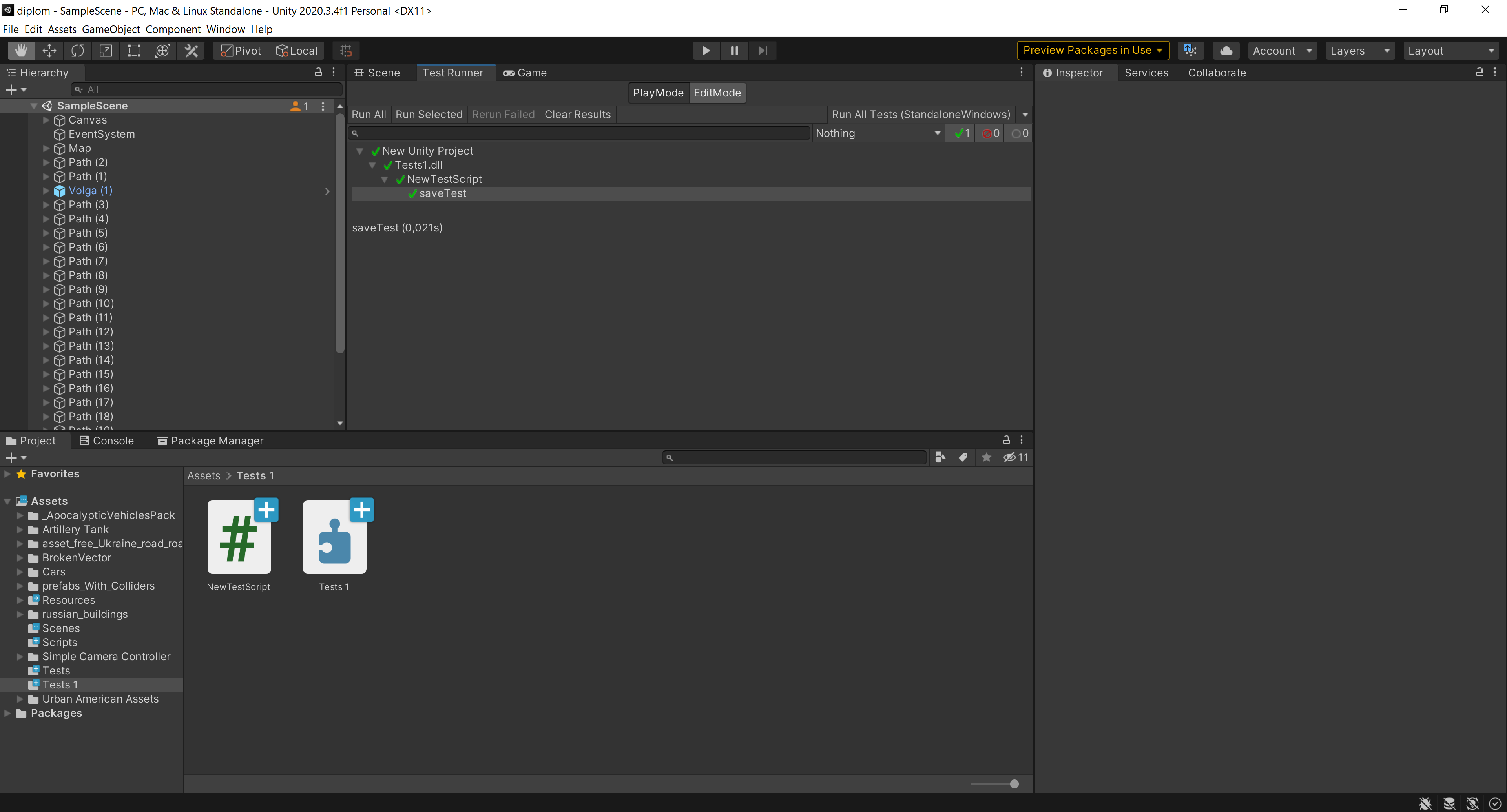


Рисунок 17 ­– Результат 2 для теста модуля «Сохранение объектов»

**Вывод**: в данной лабораторной работе были освоены методики тестирования разработанной программной системы в ручном режиме