Автономная некоммерческая организация

«Профессиональная образовательная организация»

«ФИНАНСОВО-ЭКОНОМИЧЕСКИЙ КОЛЛЕДЖ»

Отчет

по ПРОИЗВОДСТВЕННОЙ практике

На\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(наименование базы практики

Студента (ки)\_\_\_\_\_\_4\_курса\_\_\_414-ф\_\_группы

Аверин Павел Михайлович

(ФИО)

Руководитель предприятия

(должность, Ф.И.О., подпись)

Руководитель практики

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(должность, Ф.И.О., подпись)

Симферополь, 2021 г.

Календарный график прохождения учебной практики

|  |  |  |
| --- | --- | --- |
| Номер | Наименование этапа | Календарные сроки (даты выполнения) |
| 1 | Ознакомление с календарным планом программой производственной практики, её целями и задачами. Составление плана прохождения практики |  |
| 2 | Сбор и систематизация теоретического материала |  |
| 3 | Анализ и обобщение материалов, Тестирование |  |
| 4 | Создание базового прототипа с минимальным функционалом |  |
| 5 | Оформление отчета |  |
| 6 | Сдача отчета |  |

Руководитель практики: \_\_\_\_\_\_\_\_\_

Оглавление

[Введение 3](#_Toc91222428)

[Подготовка сцены 3](#_Toc91222429)

[Создание персонажа 5](#_Toc91222430)

[Анимация 8](#_Toc91222431)

[Стойка 9](#_Toc91222432)

[Бег 11](#_Toc91222433)

[Программирование 16](#_Toc91222434)

[Физика 17](#_Toc91222435)

[Анимации 22](#_Toc91222436)

[Заключение 24](#_Toc91222437)

[Список используемой литературы 24](#_Toc91222438)

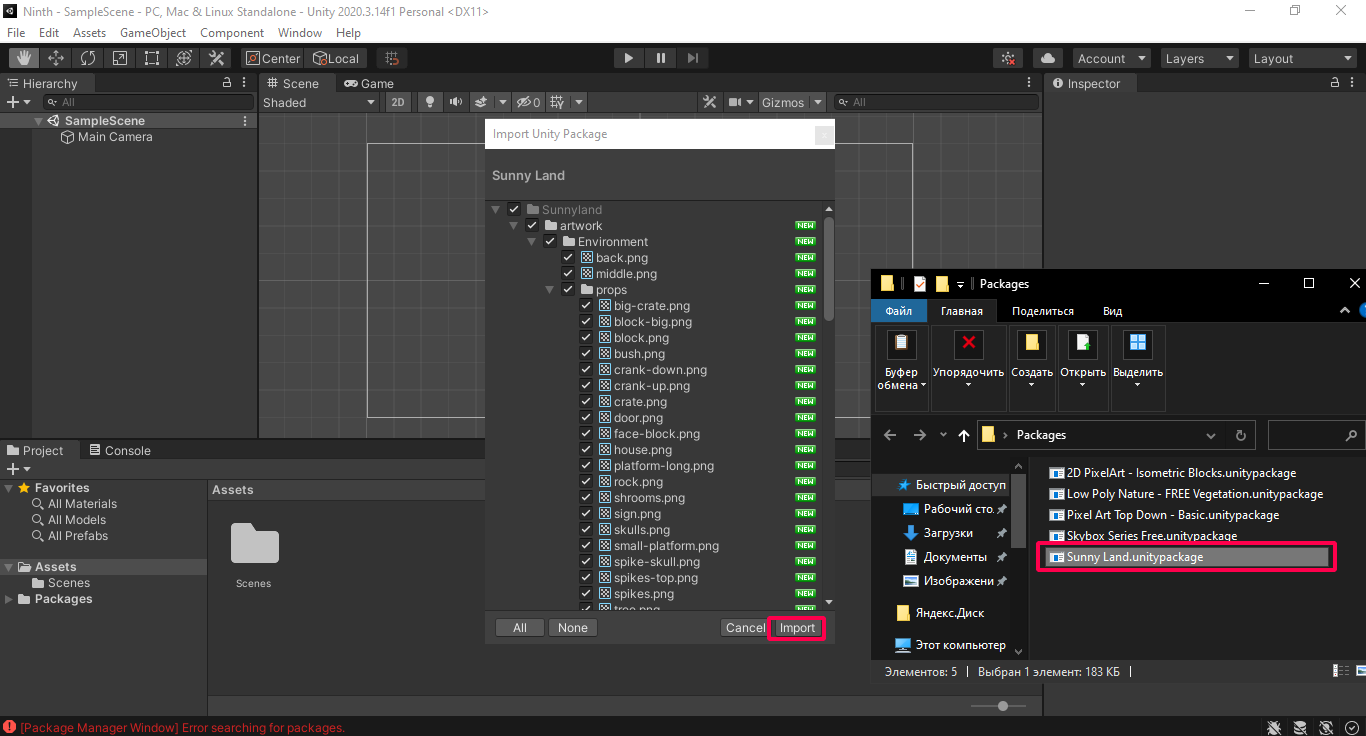
Введение

Цель практической работы – изучить принципы создания и программирования анимации, а также практическое применение полученных знаний.

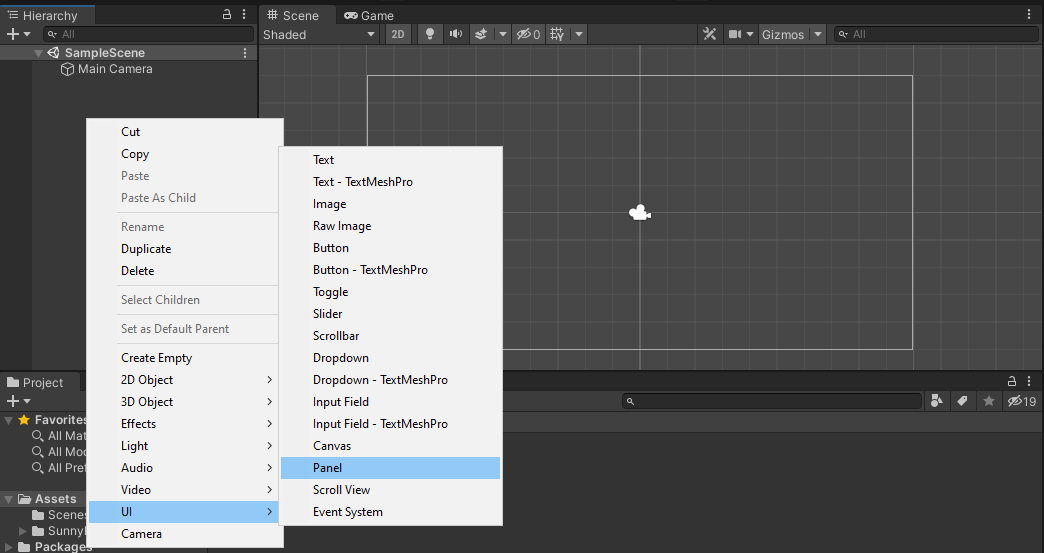
По сути, анимация — это набор изображений, которые сменяют друг друга с одинаковой скоростью. Если Вы когда-нибудь делали мультик в тетради, то догадываетесь как это работает. В Unity же, анимация — это набор параметров компонентов, которые сменяют друг друга. Помните каменного кота? На самом деле анимация этого кота содержит не картинки, а параметры масштаба, который постоянно меняется.

# Подготовка сцены

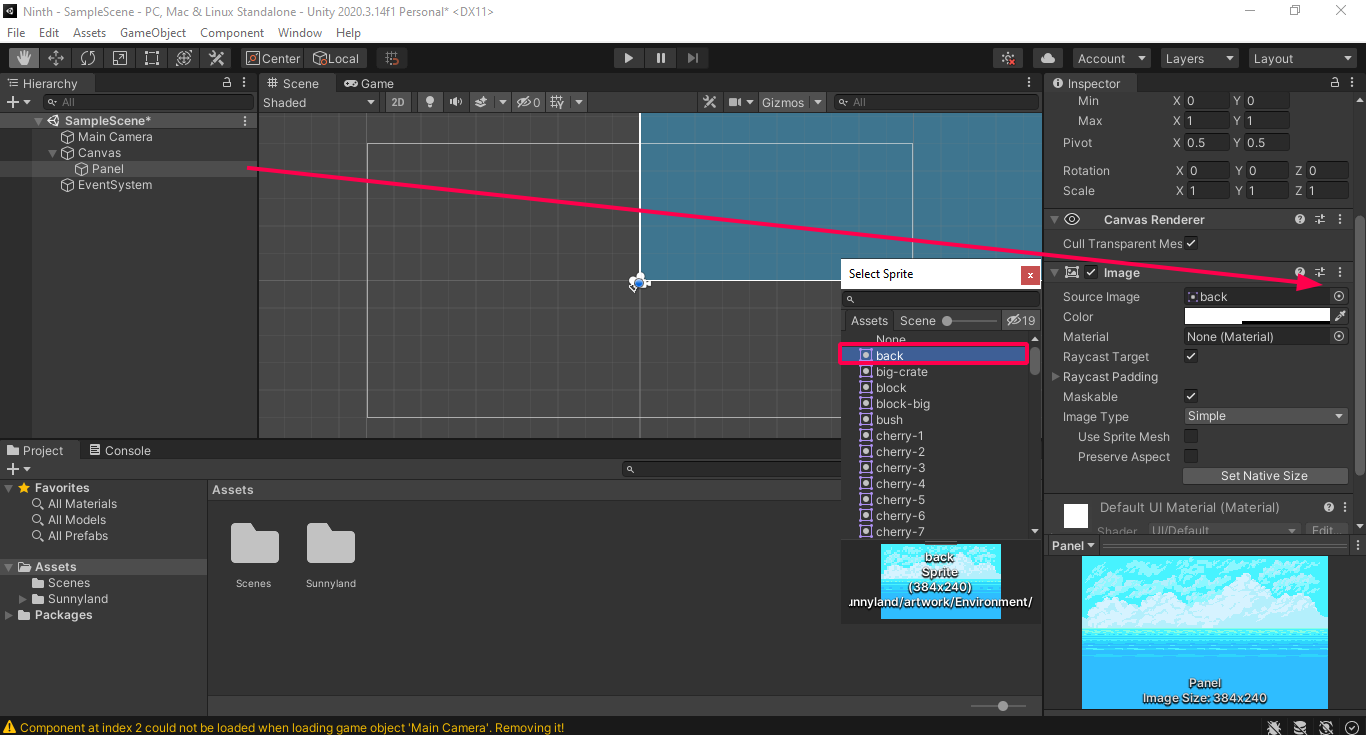
И так, я создал 2D проект в Unity и импортируйте пакет «Sunny Land.unitypackage»:



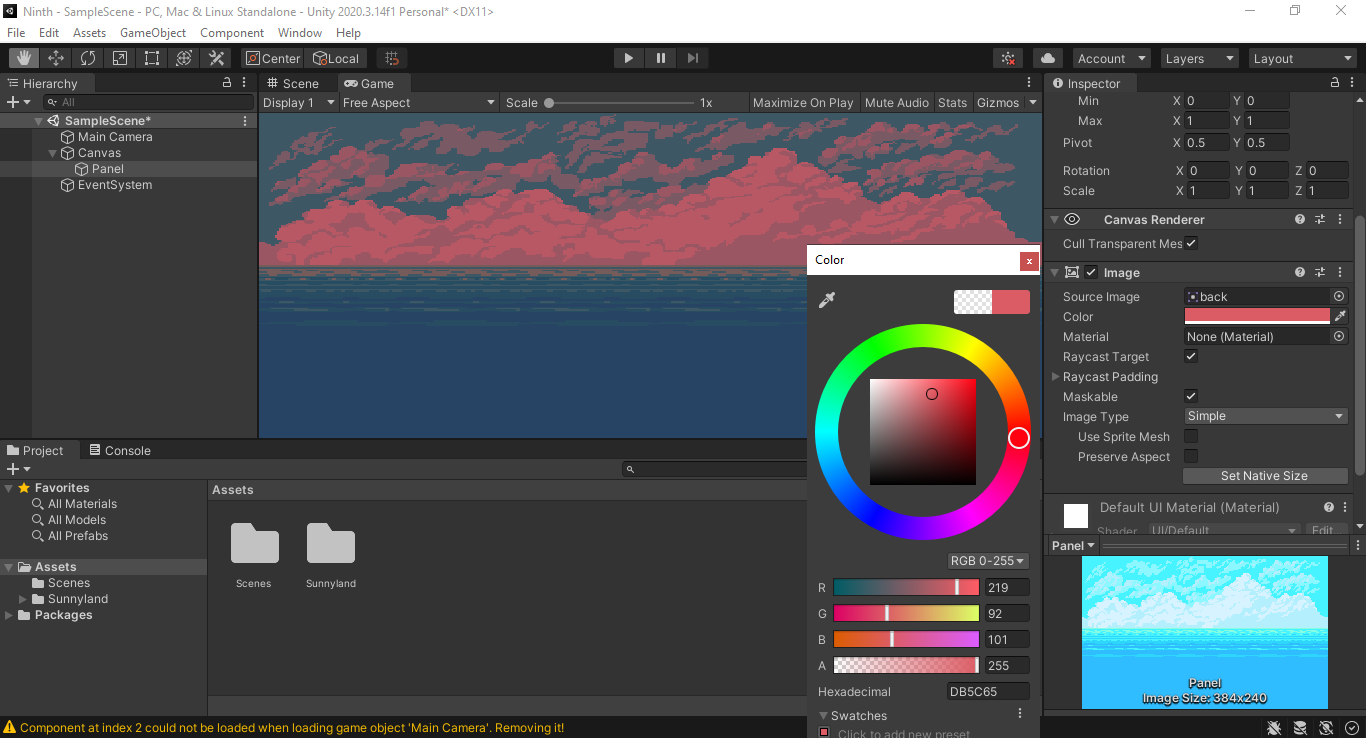
Чтобы задний фон пришелся пользователю по вкусу его стоит сменить с полностью синего на картинку. Для этого создадим панель в окне иерархий. Панель — это элемент пользовательского интерфейса UI:



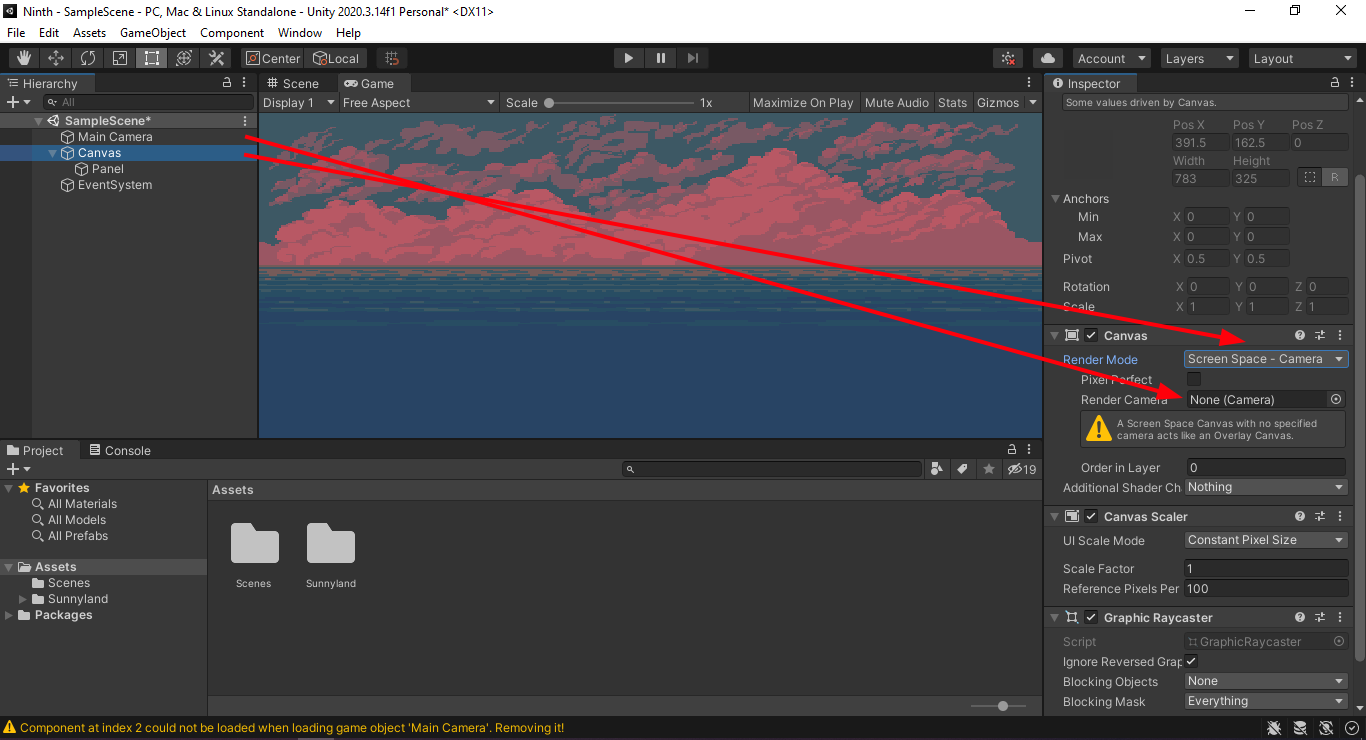
Теперь свойству «Source Image», которое находится в компоненте «Image» я задал значение «back». Это добавит красивый фон на панель:



Но если мы перейдем в окно игры, то заметим, что картинка отличается, от той, что мы выбрали. Она более тусклая. Дело в том, что в параметре цвет свойство Alpha (А) установлен на 100, что добавляет полупрозрачности изображению. Измените этот параметр на 255 и поиграйтесь с цветом:



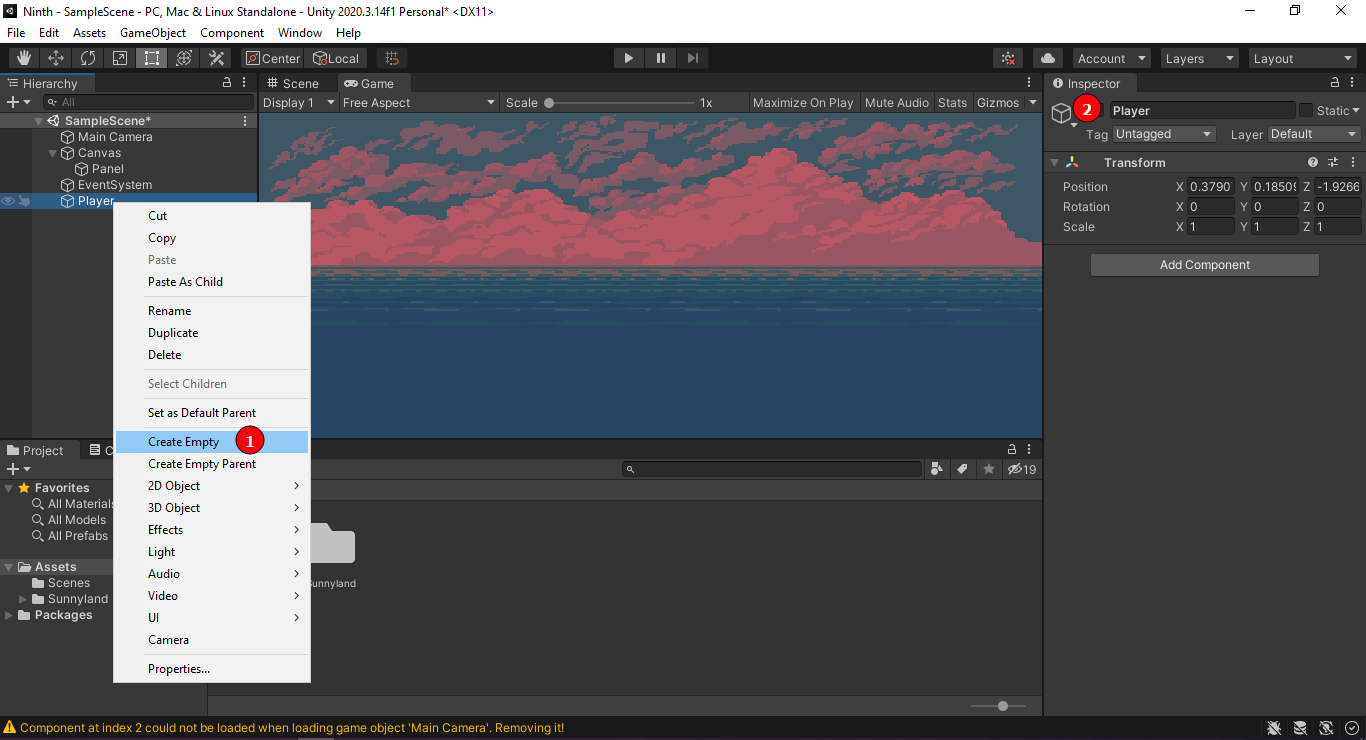
Теперь нам необходимо сделать так, чтобы задний фон был виден только на камере. Для этого в объекте Canvas я изменю свойство «Render Mode» на «Sceen Space Camera», после чего перемещу объект «MainCamera» в появившееся свойство «Render Camera»:



# Создание персонажа

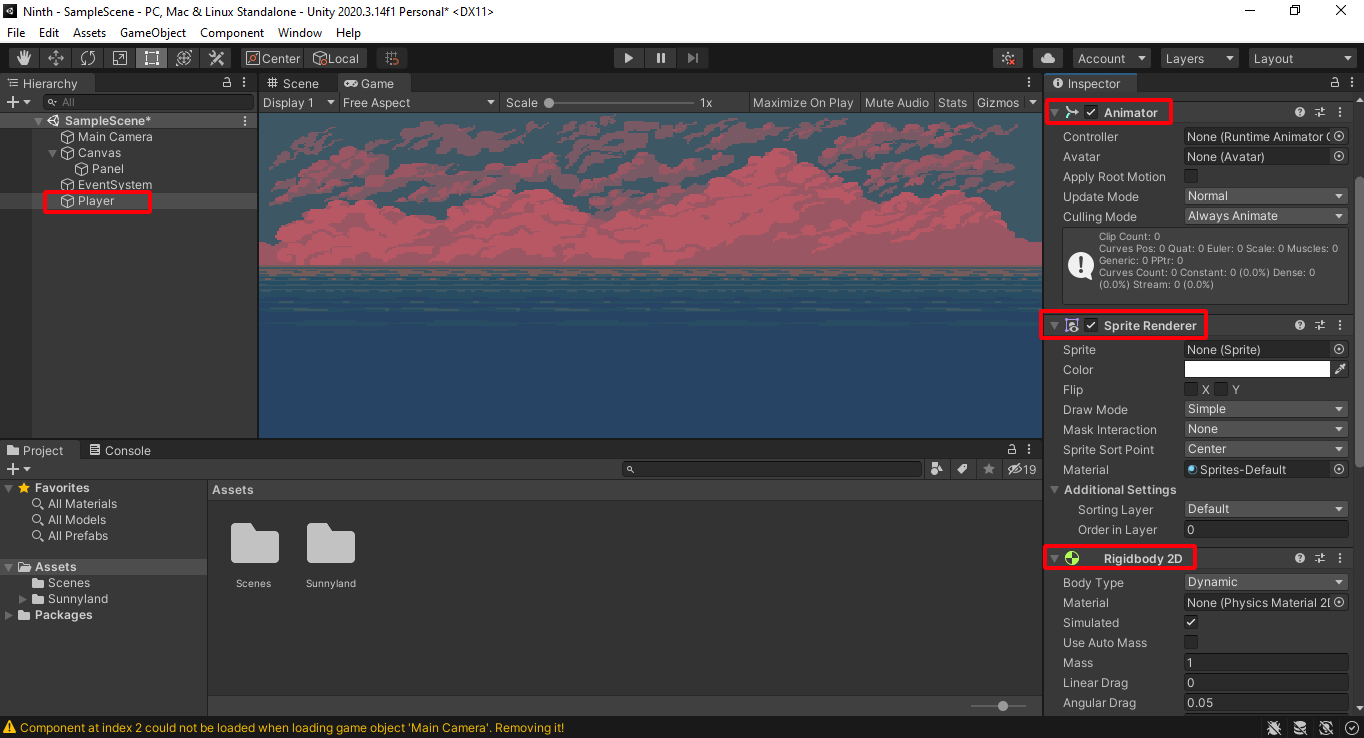
Теперь я пристулю к созданию игрока. По сути, персонаж является игровым объектом, который содержит разные компоненты для отображения спрайтов, проигрывания музыки, физики и т.д. Это означает, что нужно будет создать объект, а затем наполнить его всеми компонентами.

Я создам пустой объект в иерархии и назовите его «Player»:

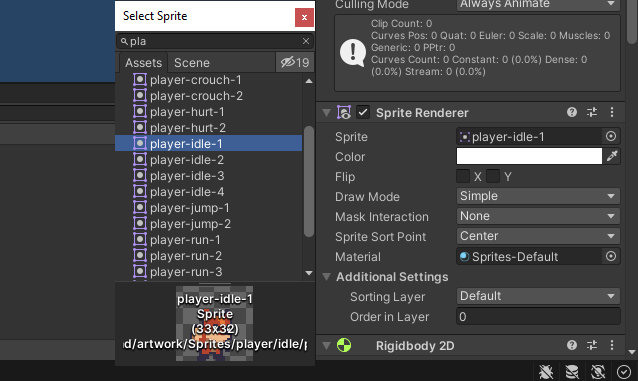


Теперь я буду добавлять компоненты с помощью кнопки «Add Component» в объекте Player:

1. Sprite Renderer. Необходим для отображения спрайта персонажа
2. Animator. Необходим для настройки анимации.
3. Rigidbody2D. Физика для персонажа

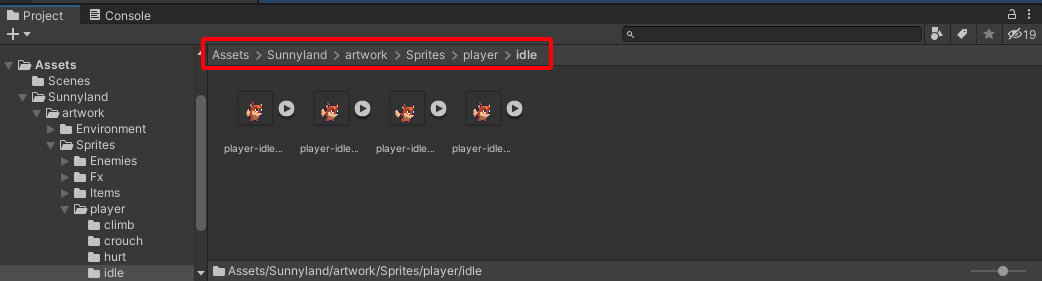


Теперь я изменю спрайт в Sprite Renderer на «player-idle-1».

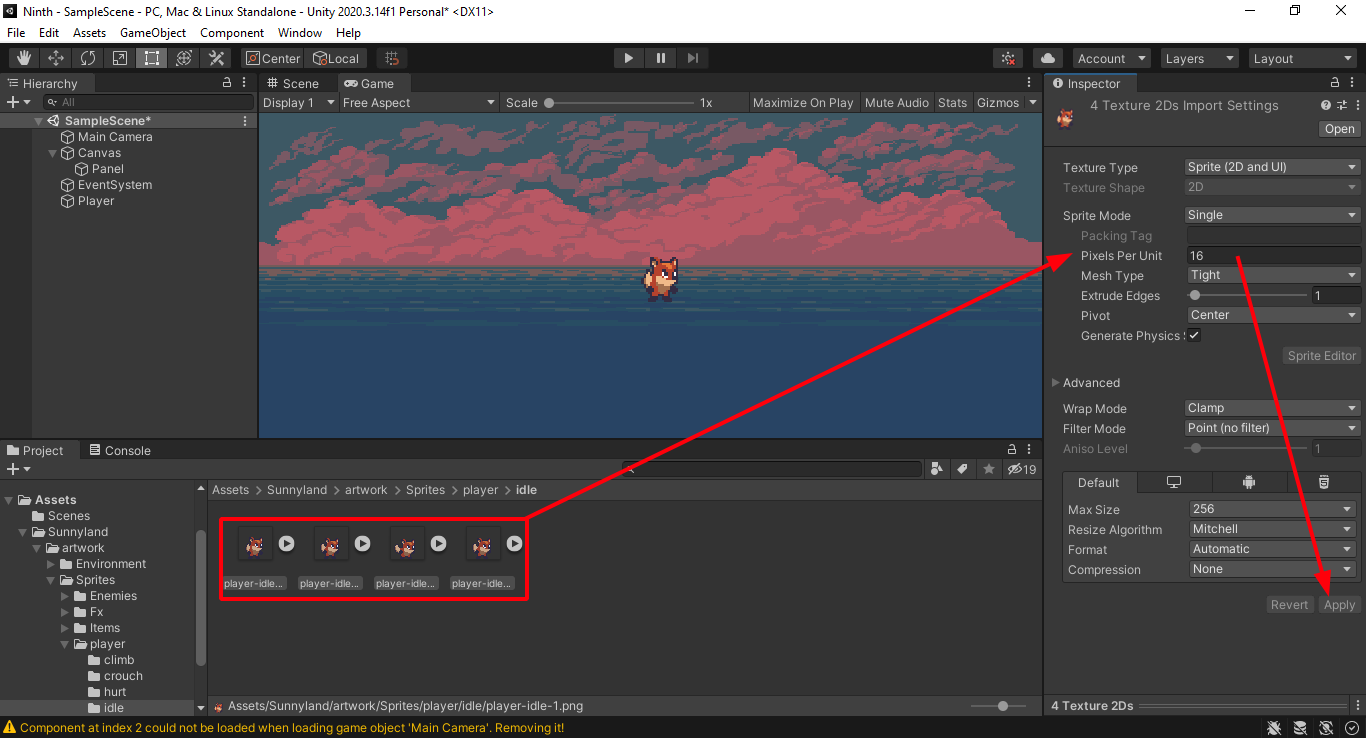


Сейчас Вы можете наблюдать, что картинка, которая была добавлена, чрезвычайно маленькая. Это произошло из-за настроек PPU (Pixel per unit). Дело в том, что в Unity все пространство поделено на юниты – клетки, а количество пикселей на ней слишком велико. Данная настройка отвечает то, сколько пикселей будет содержатся на одной такой клетке. Соответственно, чем меньше это значение, тем больше будет картинка и наоборот.

В директории «Assets/Sunnyland/artwork/Sprites/player/idle» содержатся все картинки для анимации под названием «idle». Она будет проигрываться, когда пользователь будет стоять на месте:

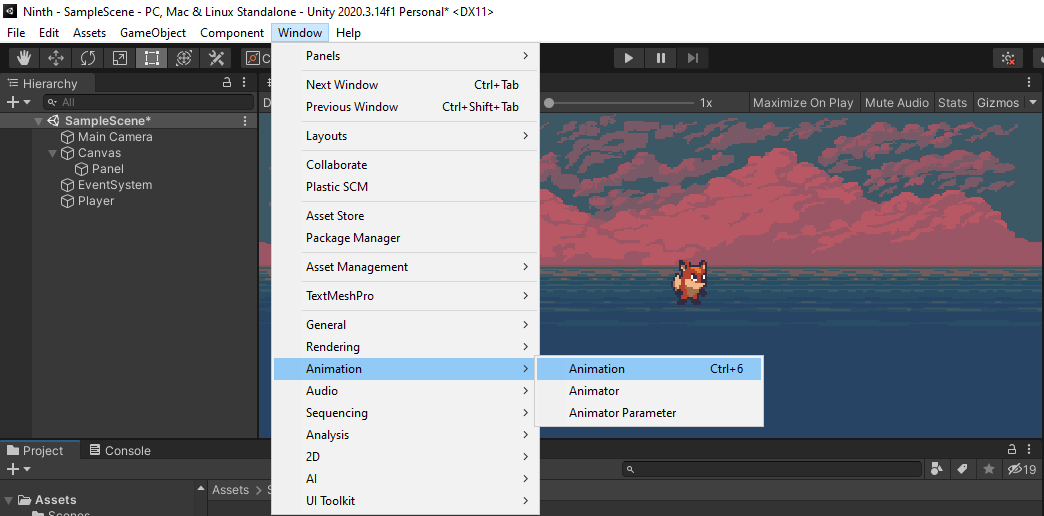


Я выделю все спрайты. Для этого нажму сначала на первый, а затем, зажав кнопку SHIFT, на последний. Все эти картинки имеют разрешение 32x32 пикселя. Это значит, что необходимо задать значение Pixels per unit на 32, а затем нажать на кнопку применить «Apply»:



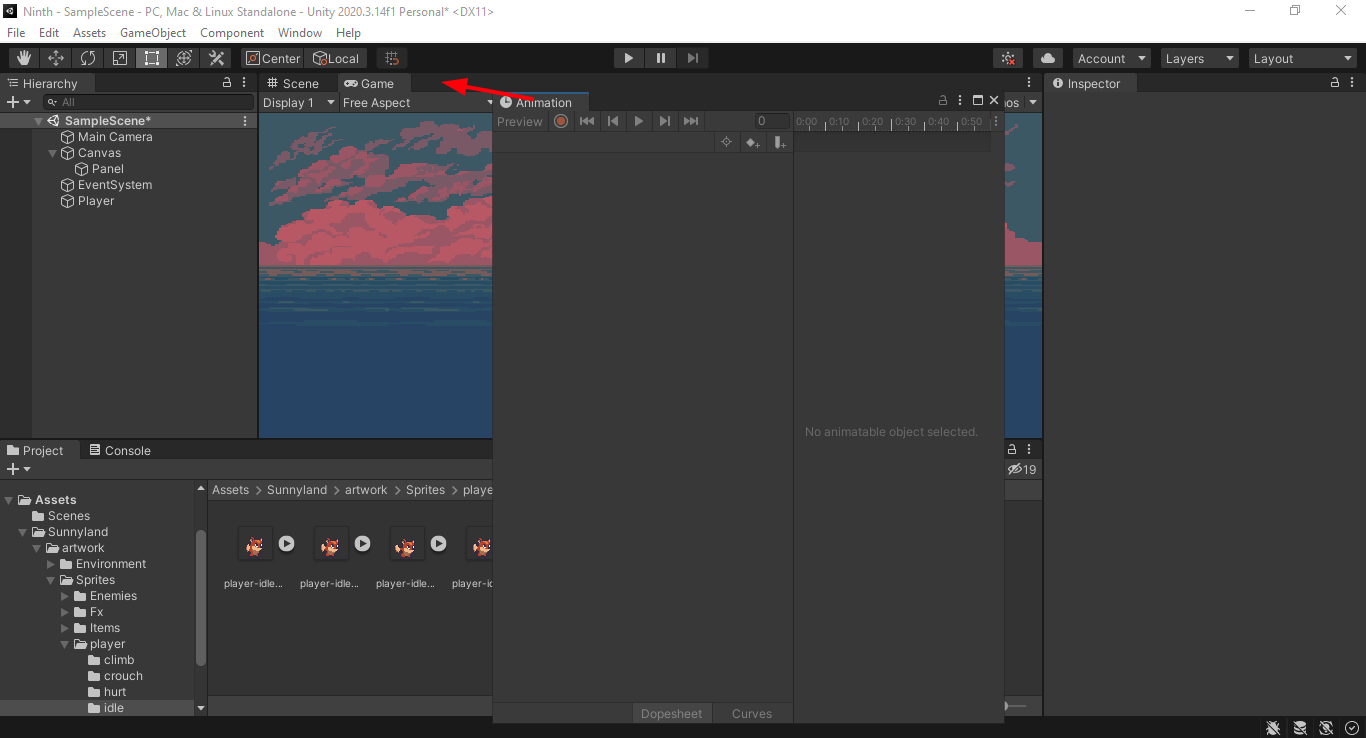
# Анимация

Окно Animation находится в верхнем контекстном меню Window, пункте Animation

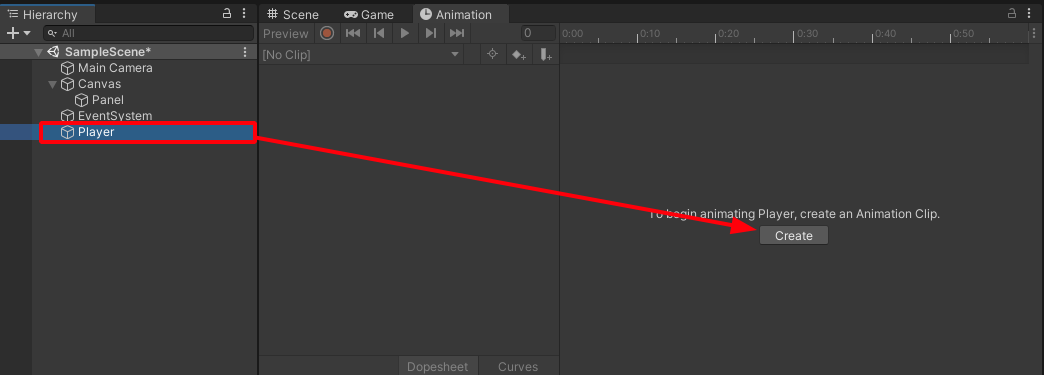


После нажатия на этот элемент меню, появится окно Animation, которое будет находится поверх других окон. Вы можете перетаскивать его за темно серую область.

Необходимо перетащить это окно рядом с окнами «Scene» и «Game». Для этого нужно перетаскивать окно Animation за название.

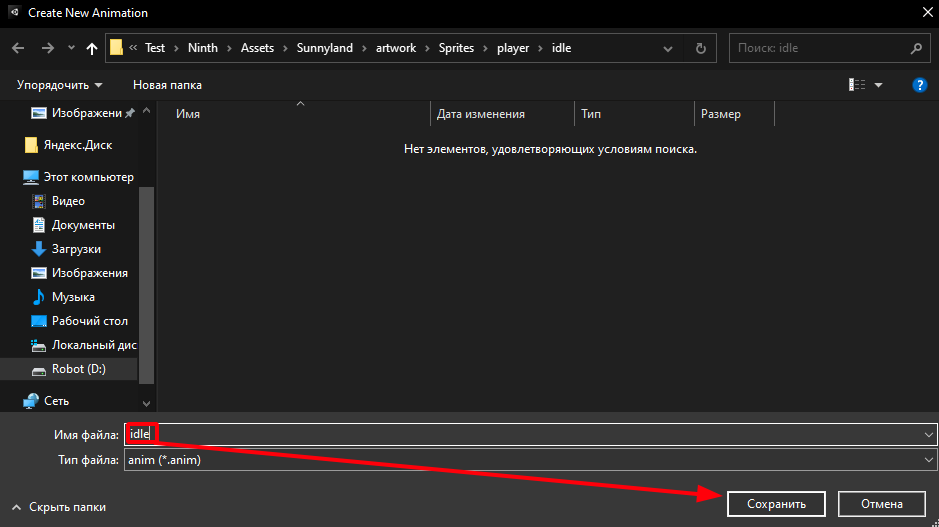


В окне анимации можно настраивать анимацию для разных игровых объектов. Чтобы добавить анимацию игроку необходимо его выбрать в окне иерархии, и после нажать «Create»:



## Стойка

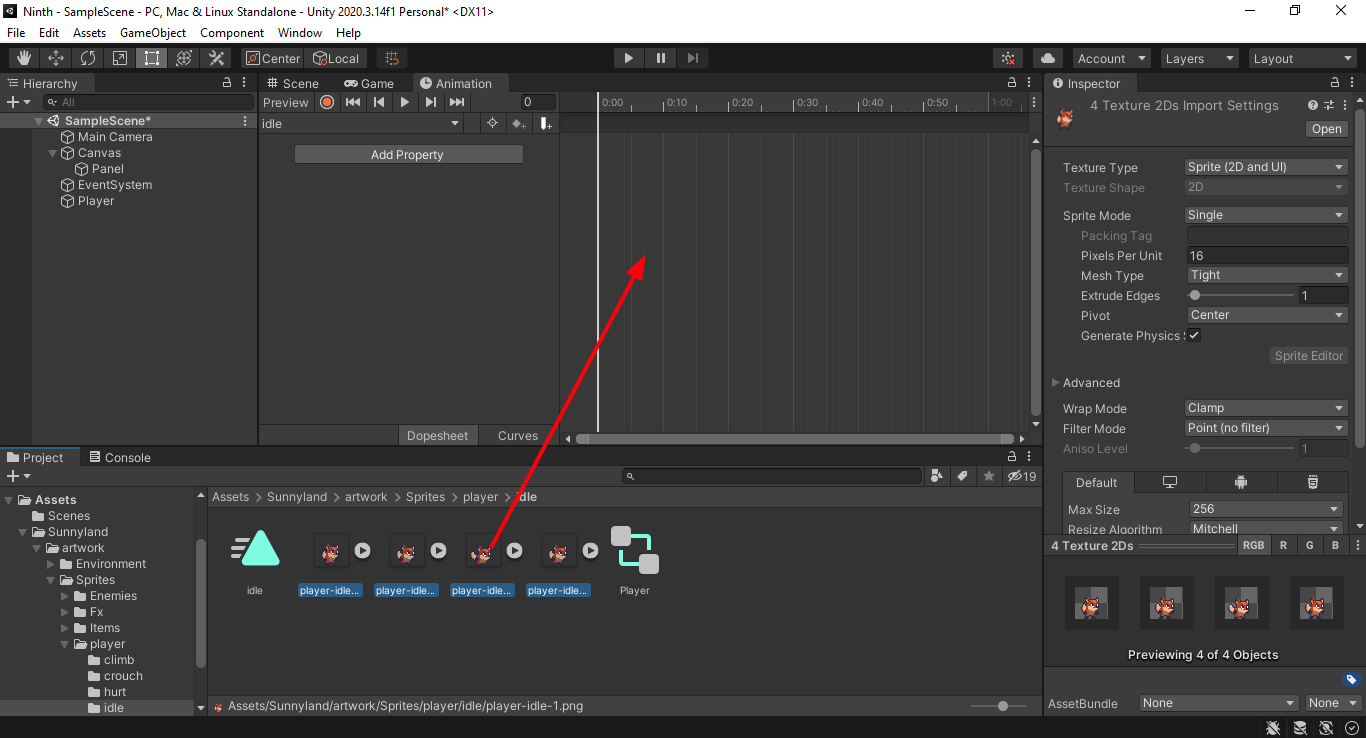
После нажатия на кнопку добавления анимации откроется проводник, в котором необходимо назвать анимацию и нажать на кнопку «Сохранить»



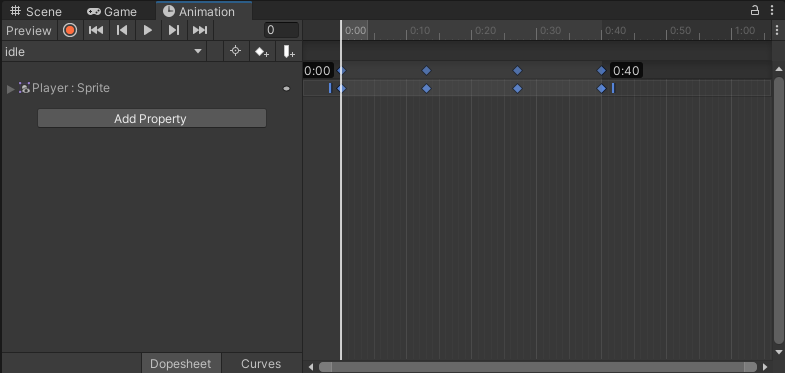
Окно анимации изменится: в нем появится настройка параметров для текущей анимации и временная шкала.

Существует такое понятие как FPS – дословно расшифровывается как Frame per Second или количество кадров в секунду. Чем больше этих кадров, тем плавнее мы видим картинку. На отметке около 24 кадров в секунду человек воспринимает такое слайд шоу как обычное видео. В играх минимальным стандартом принято 30 кадров в секунду, а большинство игр использует стандарт в 60 кадров в секунду.

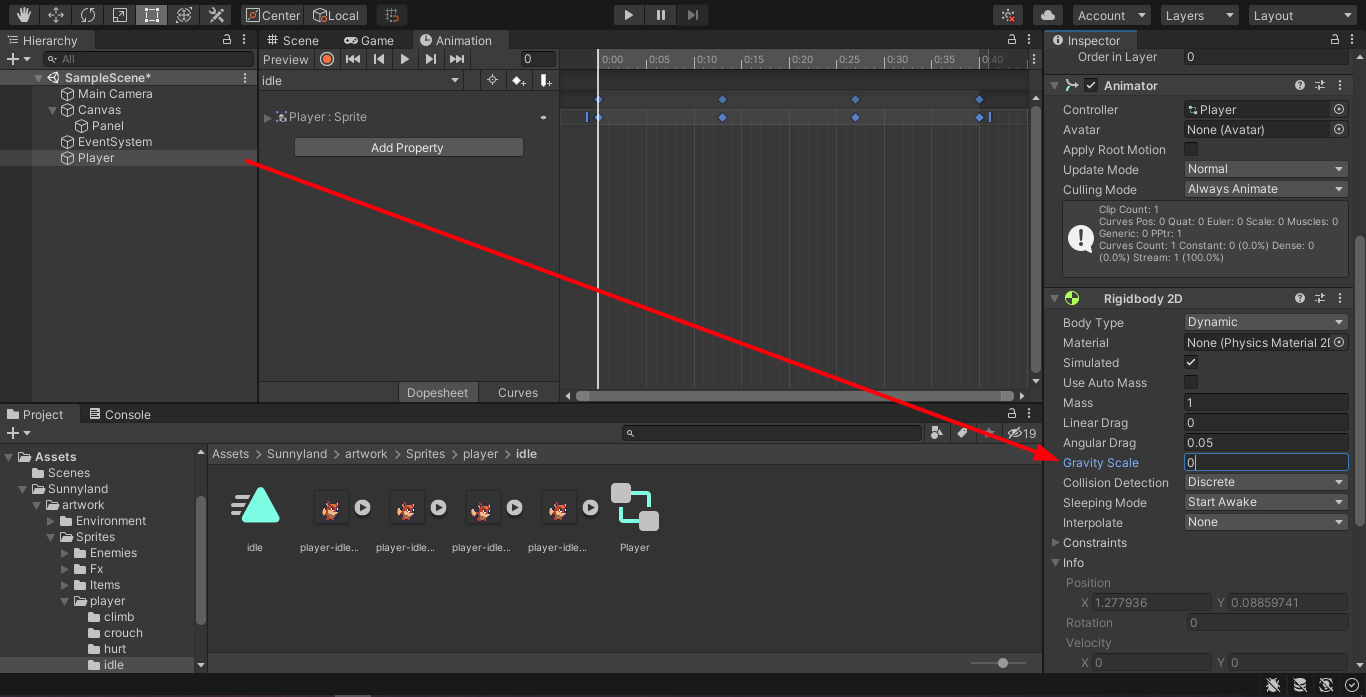
В нашем случае в распоряжении временная шкала. Каждая вертикальная линия тут – это один кадр. 60 кадров на ней равняется 1-й секунде. В каждом кадре мы можем менять какое-либо свойство у объекта, что будет отображено в виде синего ромба. Суть нашей анимации будет заключатся в том, что мы будем быстро менять 4 спрайта из директории «idle».



После добавления спрайтов появится несколько синих ромбиков. Сейчас необходимо растянуть время выполнения анимации. Дело в том, что сейчас анимация проигрывается слишком быстро. Растягивать мы будем по принципу (количество спрайтов \* 10) в нашем случае это 40. 40 кадров будет длится наша анимация. Растянуть анимацию можно за правую, синюю вертикальную черту:

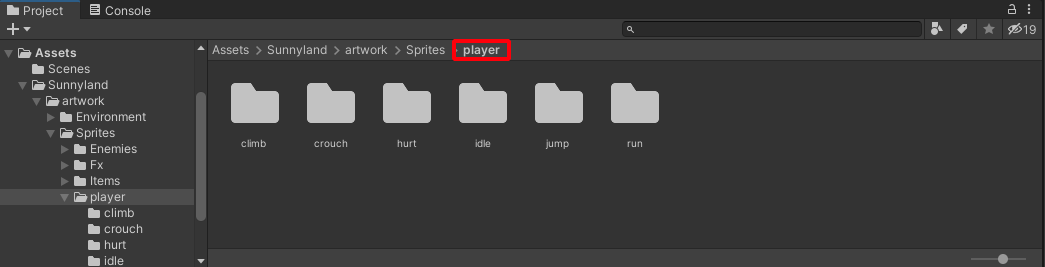


Перед тем, как проверить работу анимации нужно отключить гравитацию для игрока, чтобы он не улетел вниз. Для этого в компоненте Ridgidbody2D, который мы создавали у игрока измените свойство Gravity Scale на 0

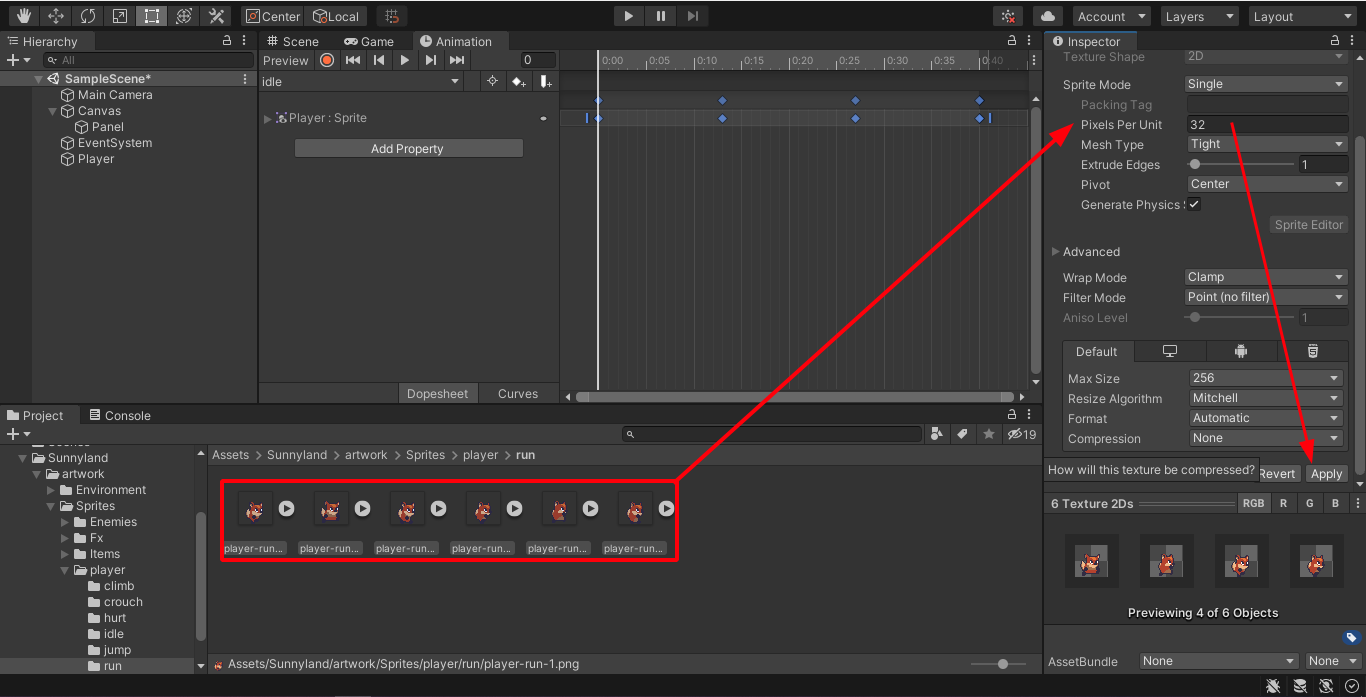


## Бег

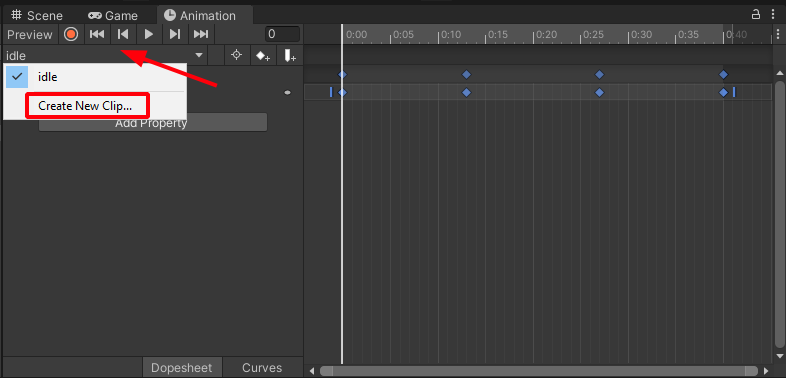
Теперь в директории «Player», где расположены все директории с спрайтами для анимации необходимо перейти в директорию «Run».:



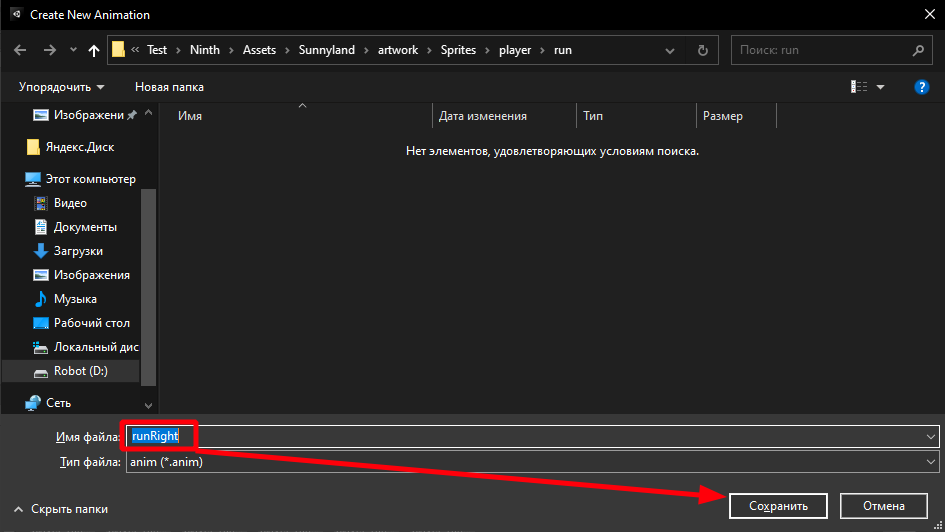
В ней необходимо сделать все то же самое, что мы делали в директории «idle»: выбрать все спрайты и изменить значение PPU на 32, после чего нажать на apply:



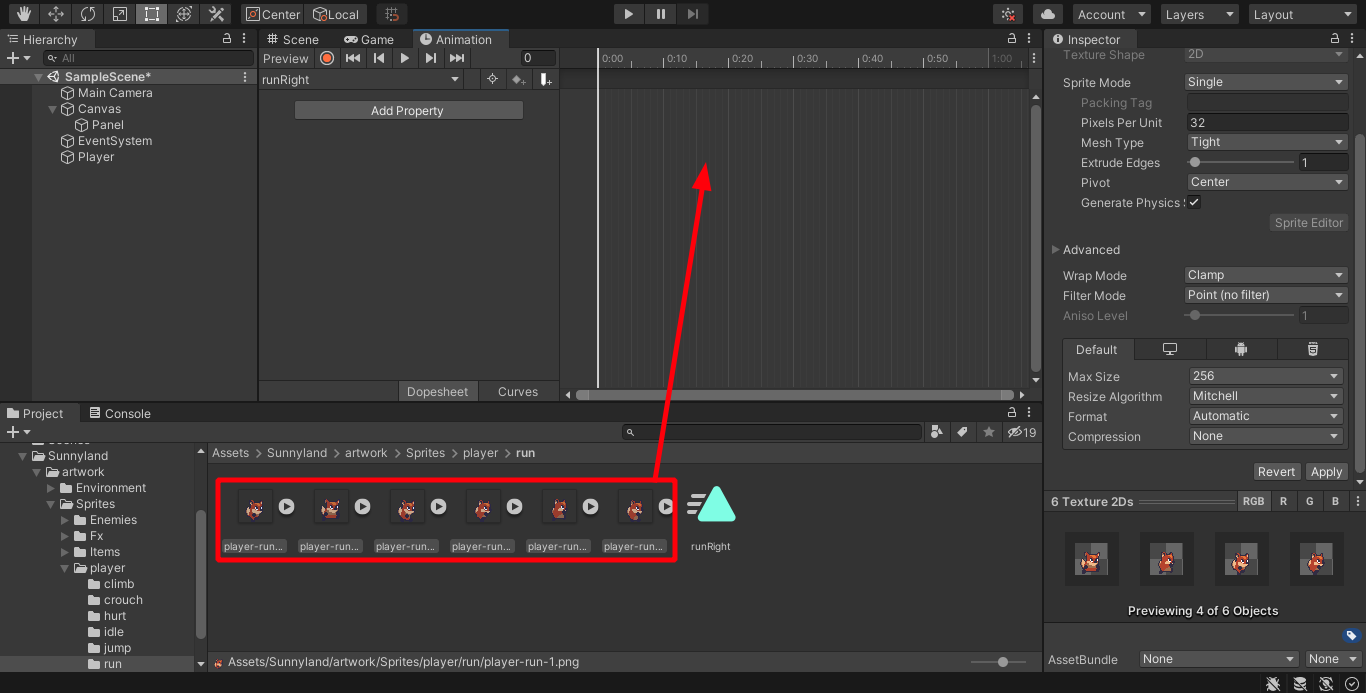
Теперь добавим игроку анимацию бега в право и лево. Для этого в окне анимации нужно нажать на меню с названием анимации и выберите пункт «Create new clip»:

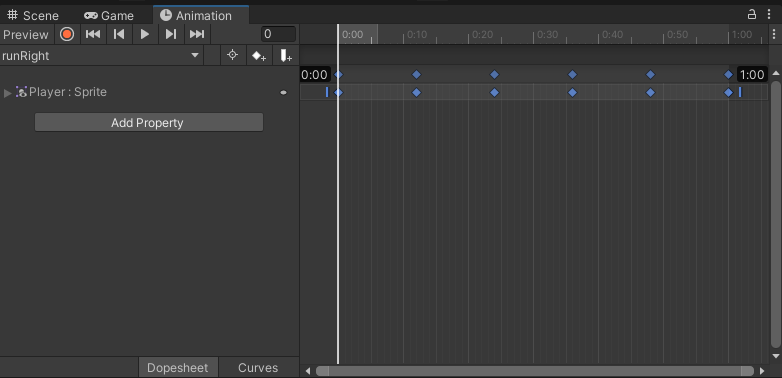


Появится знакомый нам проводник. В нем необходимо назвать анимацию «runRight», что означает бег вправо, и сохранить её.

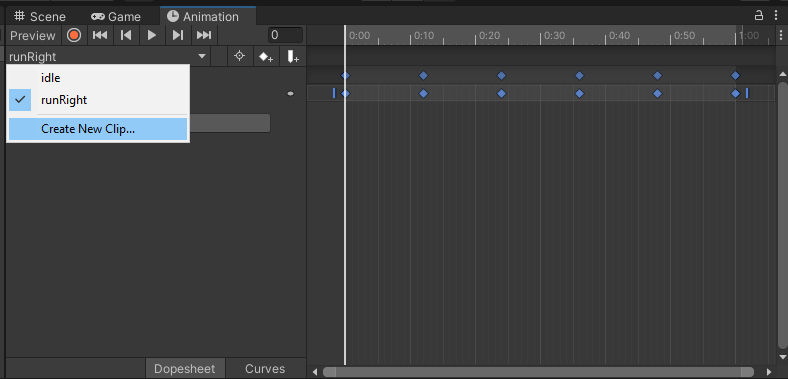


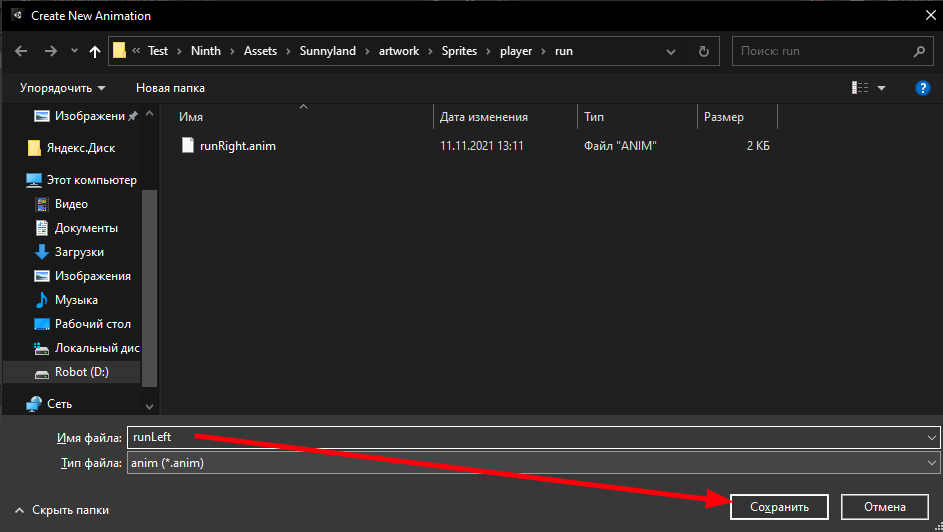
Теперь все спрайты необходимо перетащить на временную шкалу и растянуть анимацию до 1 секунды (60 кадров):

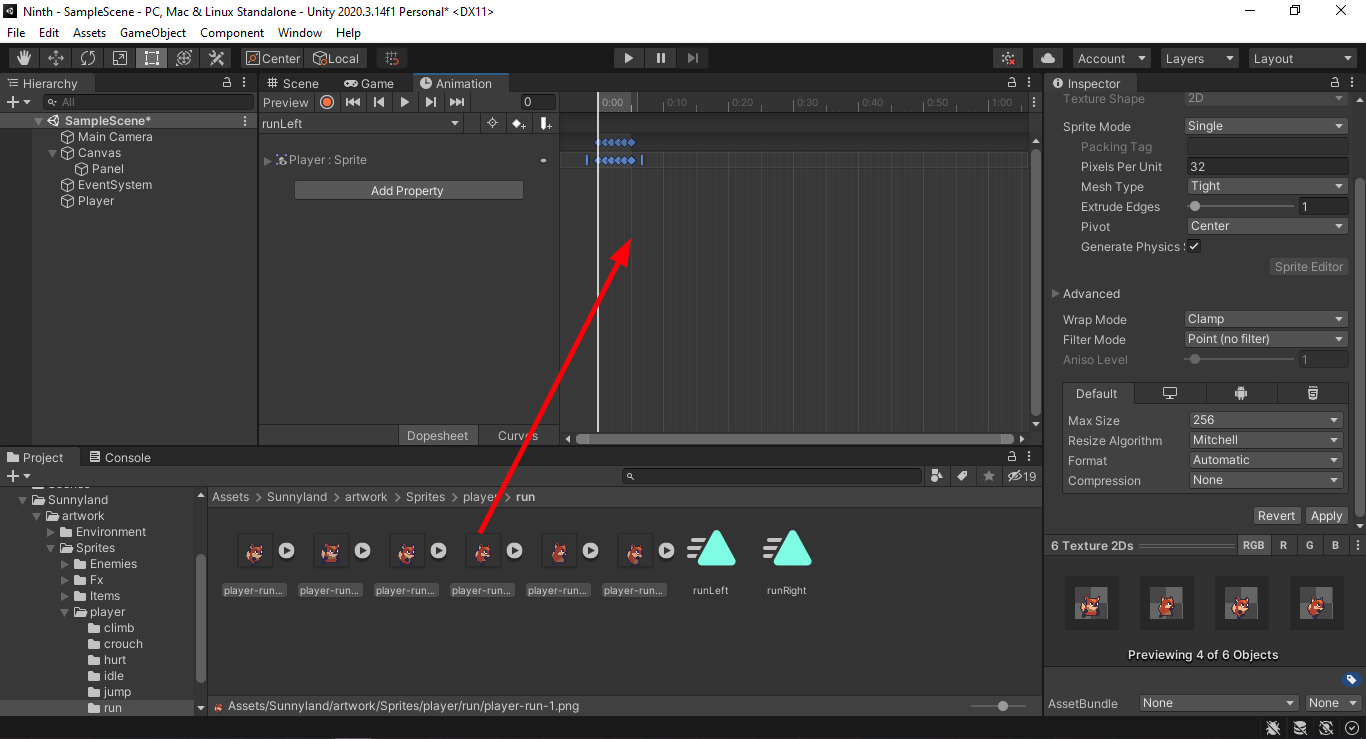


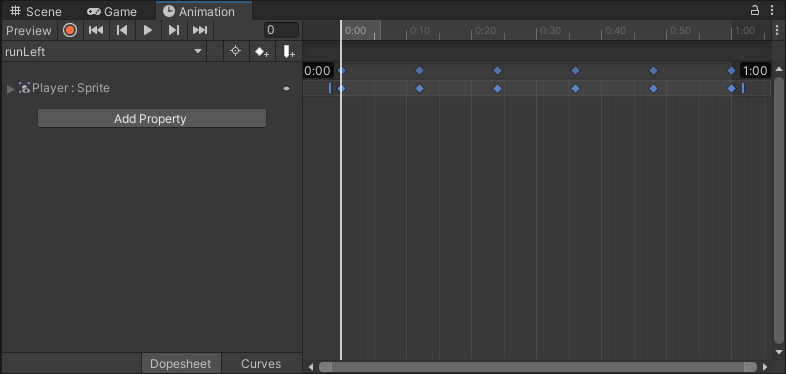


После я создал новую анимацию: «runLeft», что будет означать бег вправо. После чего добавил спрайты и растянул анимацию:



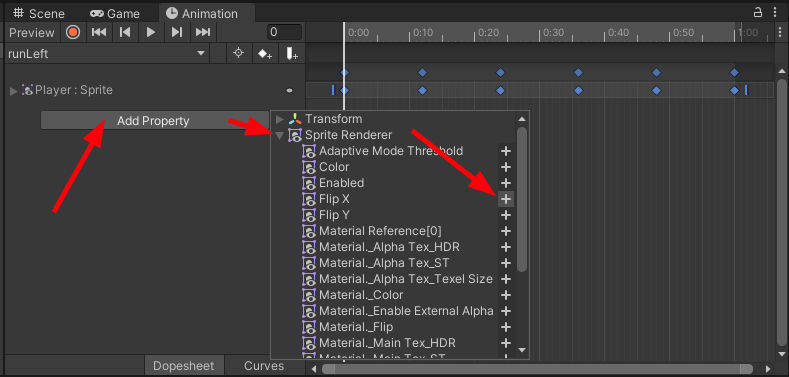




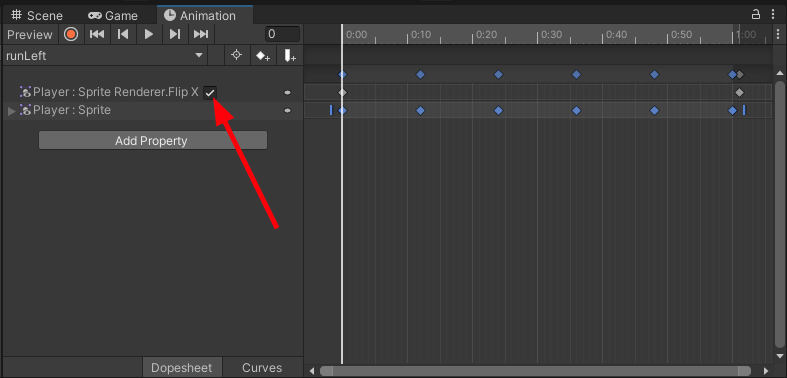


Однако в отличии от предыдущей анимации мы изменим параметр объекта Sprite Renderer «Flip X». Этот параметр позволяет зеркально отразить спрайт справа на лево. Как и было описано выше анимация в Unity это не просто набор спрайтов. В каждом кадре мы можем изменить любой параметр у объекта. Вспомните анимацию каменного кота в прошлых наших играх. Там менялся параметр Scale Y, что заставляло кота уменьшится в размерах, а затем параметр менялся обратно.

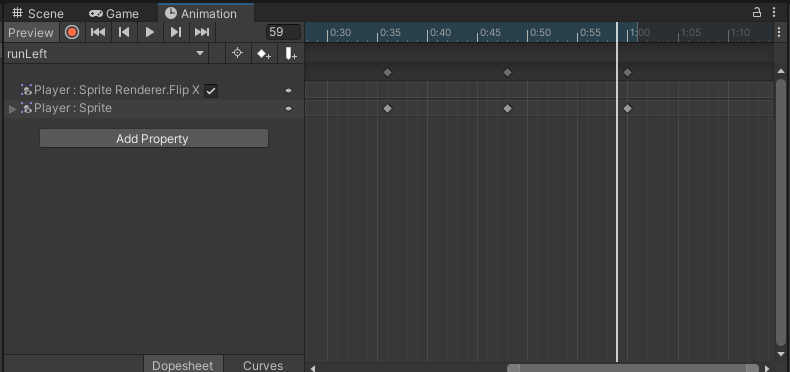
Чтобы добавить изменение параметра нам нужно нажать на кнопку «Add Property», а затем выбрать какое свойство у какого компонента мы хотим изменить.



На временную шкалу добавилось всего 2 ромбика в начало и конец анимации. Только на этих позициях будет меняться параметр Flip X. Теперь, чтобы объект был отзеркален нужно нажать на галочку напротив параметра «Player: SpriteRenderer.Flip X»:

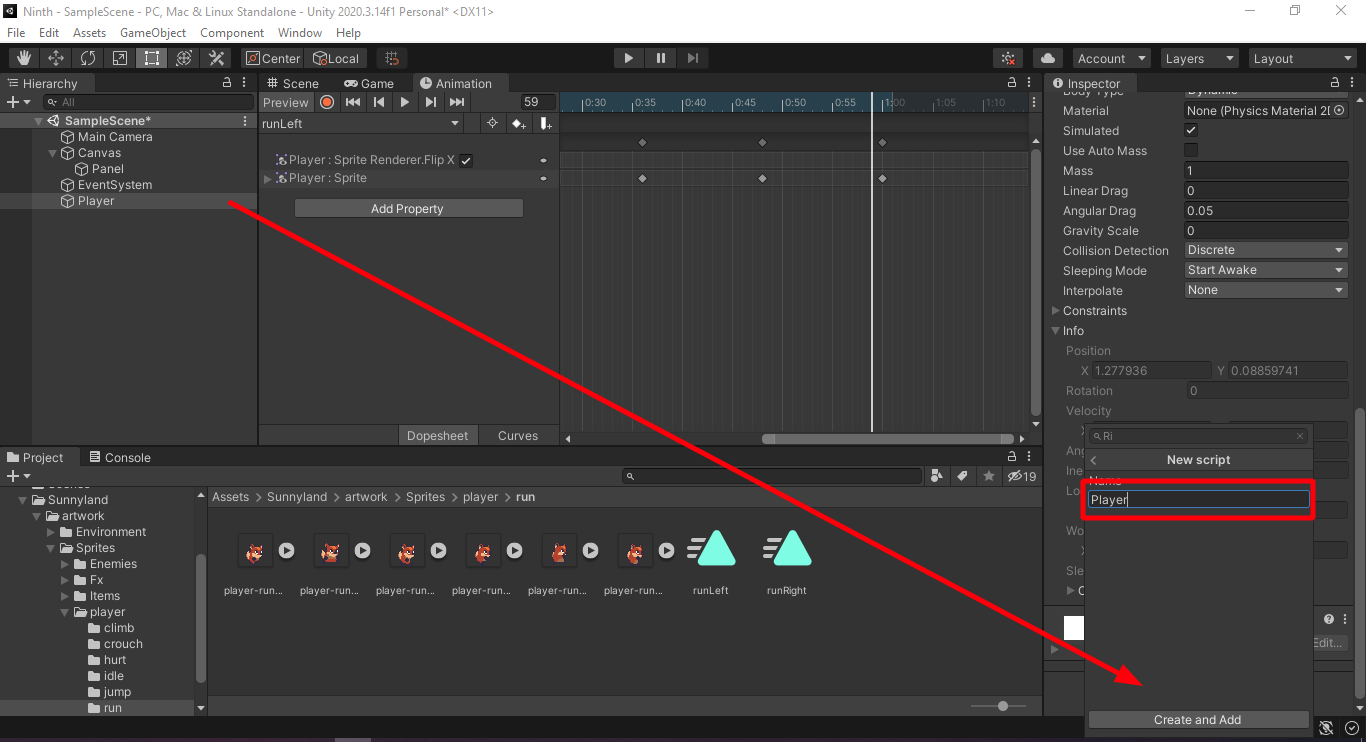


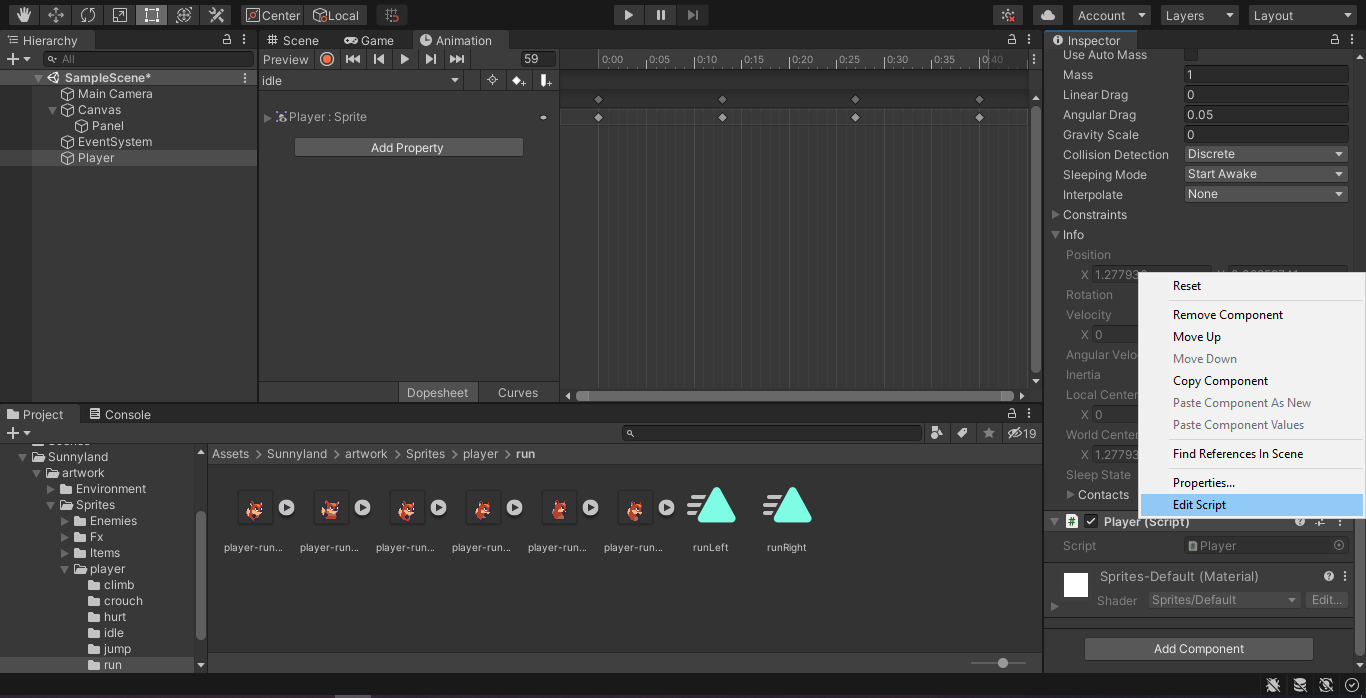
После чего я переместился на на второй ромб и удалил его, нажав на клавишу delete.



# Программирование

Теперь необходимо заставить персонажа двигаться и связать его движения с анимацией. Для этого в игрока необходимо добавить новый скрипт «Player», а затем открыть его:

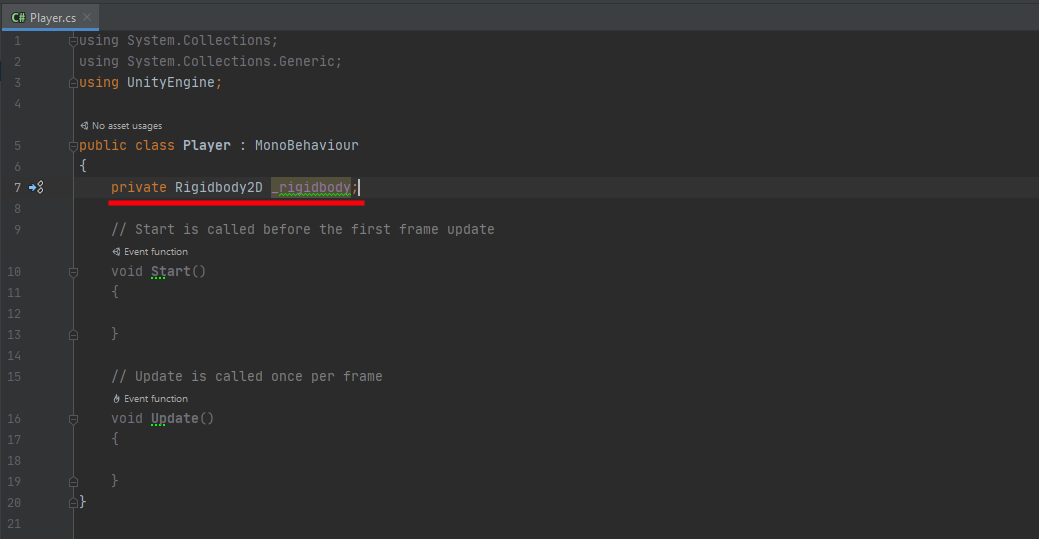




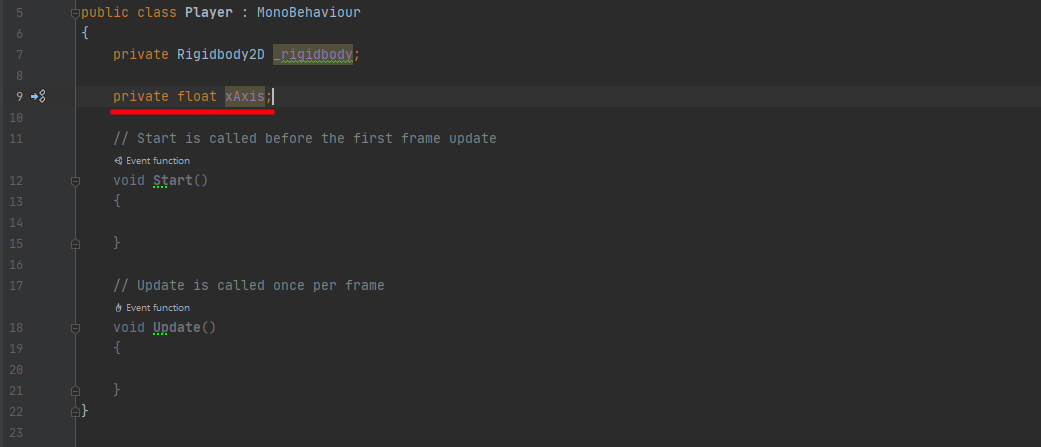
## Физика

Физика персонажа основывается на нажатии на кнопки: a, d. Программа будет задавать скорость персонажу используя компонент Rigidbody2D.

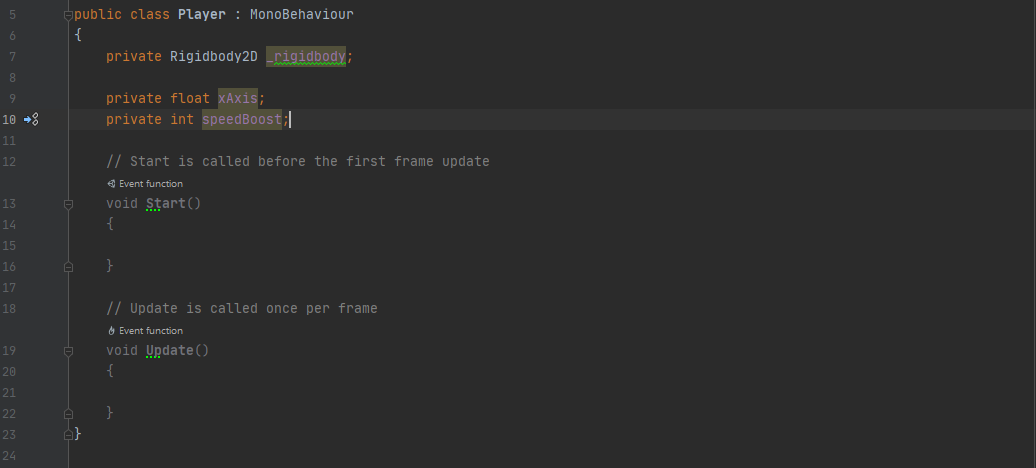
Для начала в класс с персонажем поместим переменную (коробочку), которая будет хранить физику персонажа (Rigidbody) и назвём её \_rigidbody:



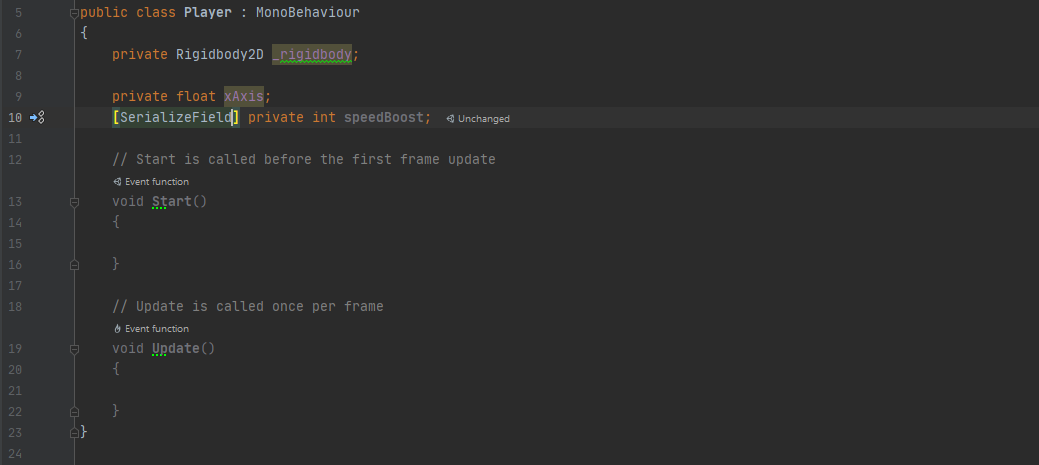
Затем нужно создать еще одну переменную, которая будет хранить направление движения. Она будет хранить значения от -1 до 1. Когда персонаж не движется, значение будет равно нулю. Когда движемся влево от -1 до нуля, а когда движемся вправо – от нуля до 1. Называться переменная будет xAxis, что означает направление по оси X:



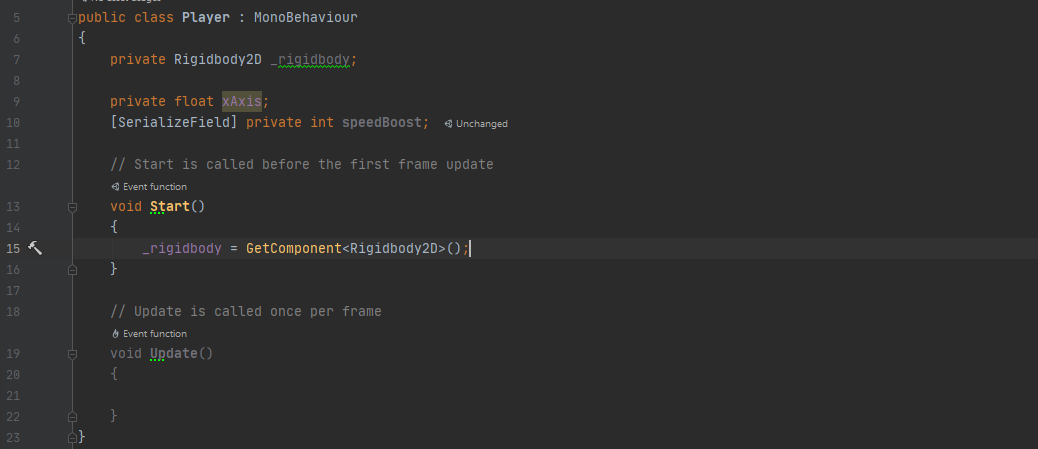
Однако если напрямую задавать скорость персонажу, как направление, то идти он будет очень медленно. Чтобы ускорить его, создадим переменную speedBoost. Это будет целое число, на которое будет умножаться направление движения. Так как эта переменная хранит только целые числа, то и тип переменной (форма коробочки) будет int:



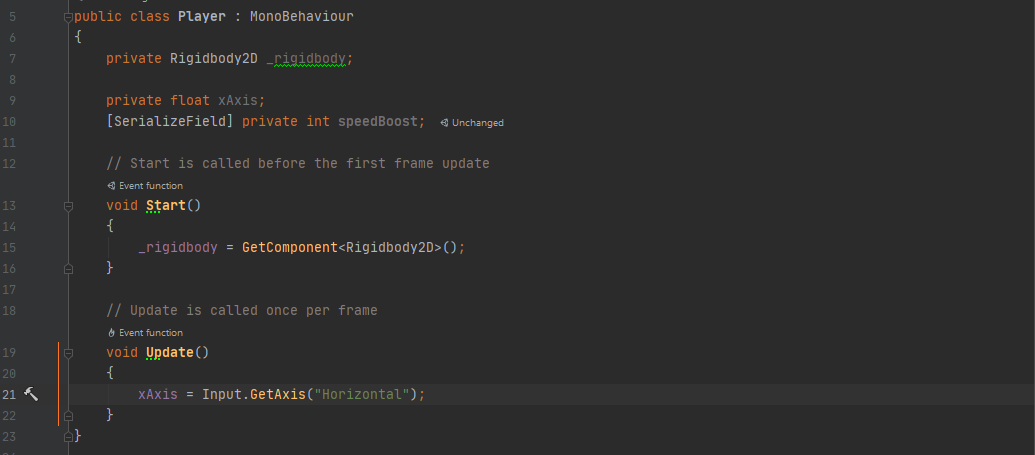
Однако эту скорость хотелось бы настраивать из Unity и для этого необходимо добавить перед её определением «[SerializeField]»



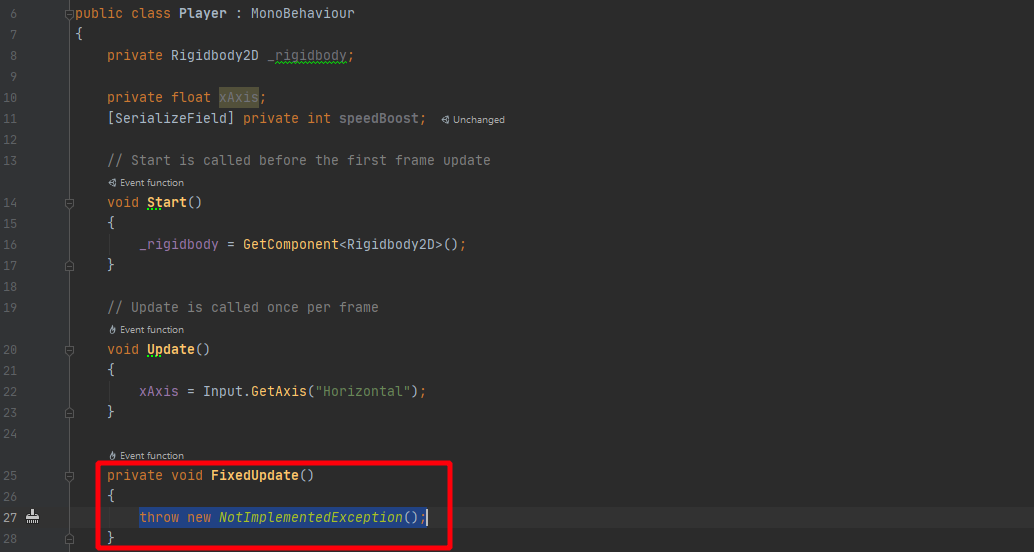
У нашего персонажа есть компонент Rigidbody и нам необходимо записать его в переменную \_rigidbody. Для этого сначала мы обратимся к переменной, а затем используя знак «=» присвоим её значение. Чтобы записать компонент из игрока в переменную нам необходимо воспользоваться функцией «GetComponent» в которой нужно указать какой компонент мы хотим получить:

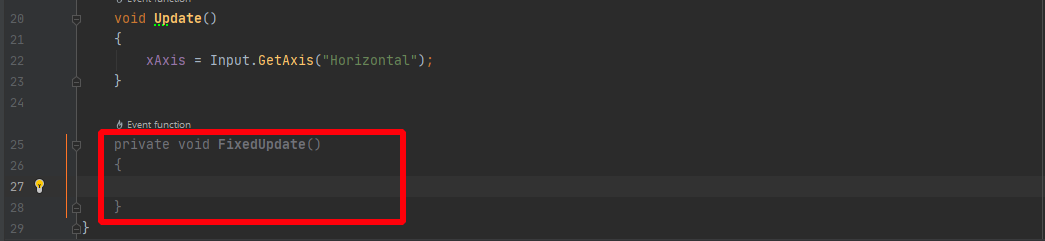


Теперь перейдем в функцию Update. В ней необходимо получить направление движения. Для этого обратитесь к переменной xAxis и запишите в неё значение «Input.GetAxis(«Horizontal»)». Input отвечает за любой ввод пользователя. Функция GetAxis позволяет получить направление опираясь на кнопки, которые нажал пользователь. «Horizontal» это направление, в котором мы хотим получать значения:



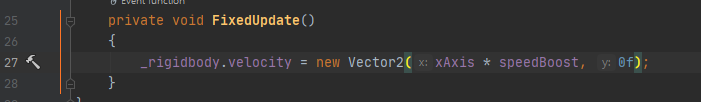
Теперь нам нужно задать скорость персонажу. Однако делать это нужно в функции FizedUpdate, так как она больше подходит для этих задач.

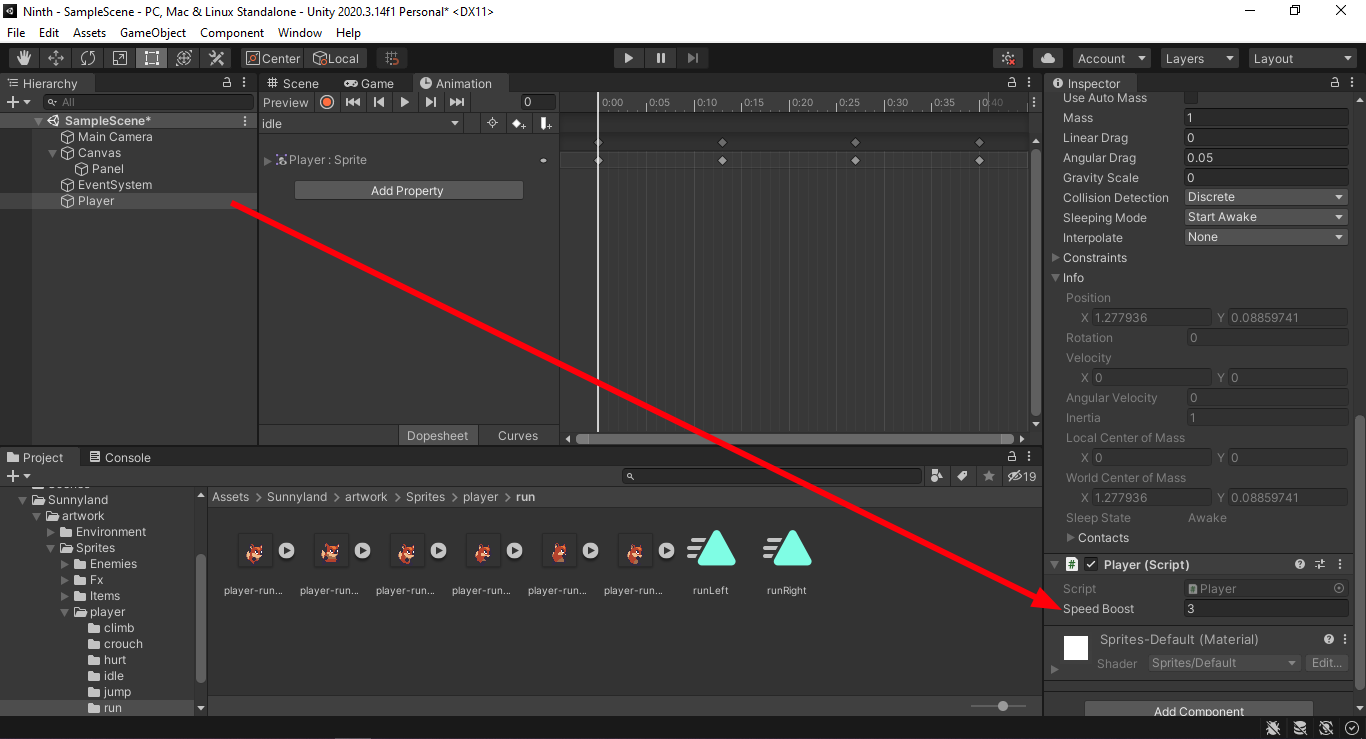




Чтобы задать скорость необходимо изменить свойства переменной velocity, которая находится в объекте \_rigidbody. Velocity хранит объект Vector2, который является перечислением 2-х чисел. В нашем случае это скорость по вертикали и горизонтали (x, y). Необходимо создать такой же Vector2 и скорость по x задать как произведение xAxis на speedBoost, а скорость по y оставить как 0:

Не пишите все, что отмечено серым цветом – это авто подсказки от Rider.

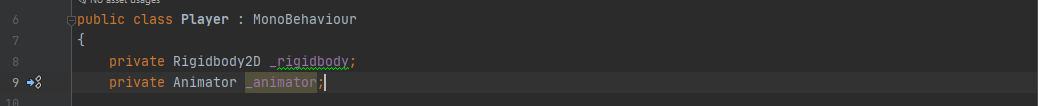




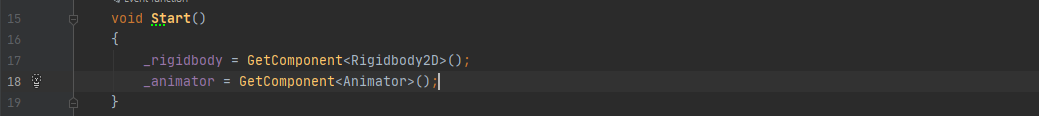
Теперь персонаж должен передвигаться по карте, при запуске игры

## Анимации

Чтобы добавить анимации, необходимо добавить переменную, которая будет хранить компонент «Animator»:

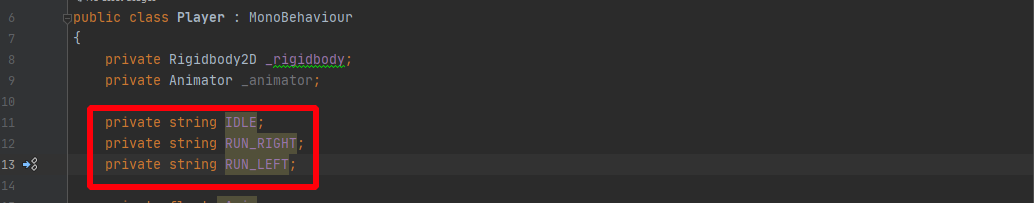


После чего получить компонент Animator в функции Start, так же, как это было сделано с Rigidbody:

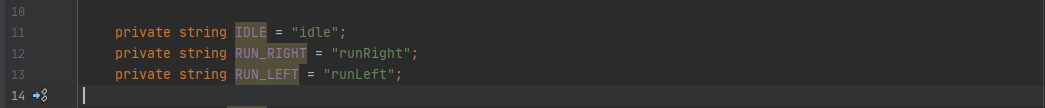


Наша анимация будет основывается на состояниях. У игрока будет 3 состояния: бездействие, бег вправо и бег влево. В процессе игры мы будем проверять направление движения персонажа. Если персонаж будет бежать вправо, то мы изменим состояние анимации на бег вправо и так далее.

Для начала все эти состояния нужно описать. Для этого необходимо создать 3 переменные: IDLE, RUN\_RIGHT и RUN\_LEFT. Они будут хранить названия анимации. Названия это текст. В Unity тип, который хранит текст называется «string», что в переводе – строка:



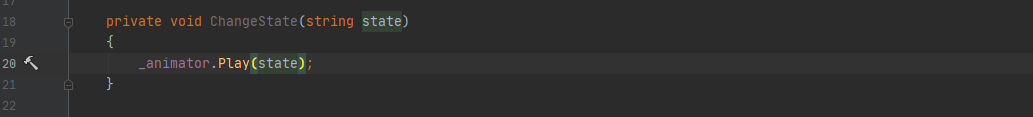
Теперь нужно задать названия анимации. Для IDLE это «idle», для RUN\_RIGHT это «runRight», для RUN\_LEFT это «runLeft»:



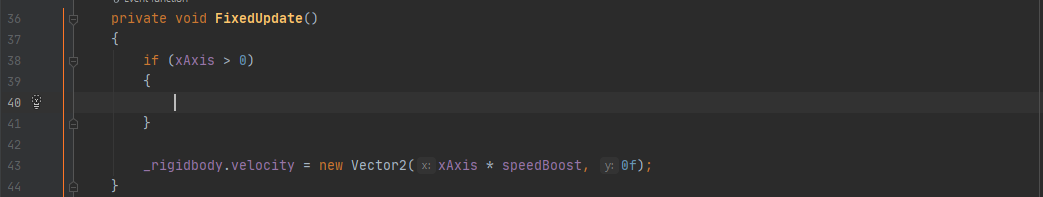
Затем создается функция, которая будет менять состояния анимации для персонажа. На вход она будет принимать состояние и запускать его. Функция нигде кроме игрока использоваться не будет, а значит необходимо добавить «private» в начале. Возвращать она ничего не будет значит её тип «void». А называться она будет ChangeState:



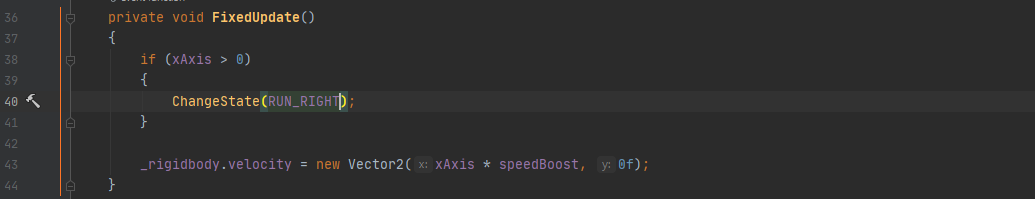
Теперь мы обратимся к Animator и вызовем у него функцию Play, в которую передадим состояние, которое мы получили:



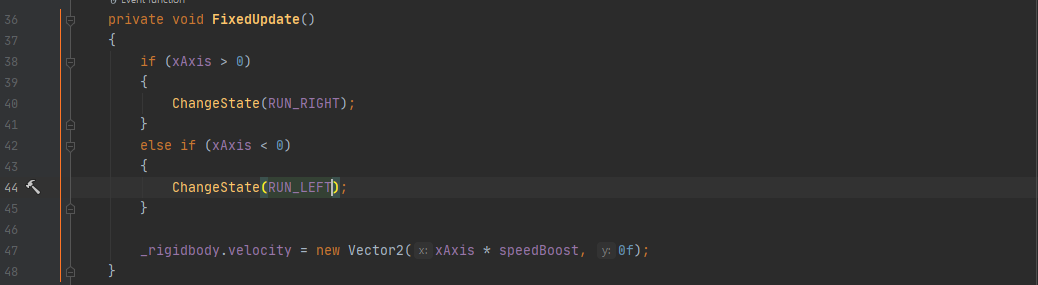
Все эти состояния нам необходимо как-то менять, во время игры. Для этого в функции FixedUpdate мы проверим текущее направление xAxis. В этом нам поможет условие. Чтобы написать условие нужно воспользоваться оператором «if», после чего в скобках добавить условие. Если мы хотим, чтобы при движении вправо запускалась анимация ходьбы вправо, нам необходимо проверить является ли значение xAxis больше 0:



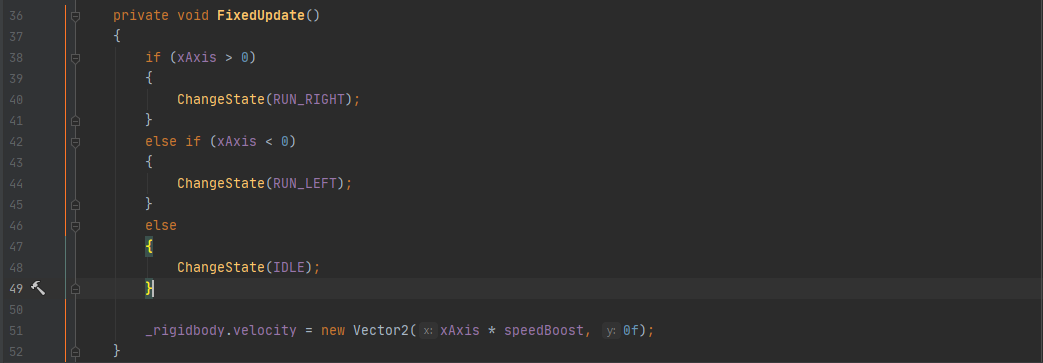
Если это так, то мы должны изменить состояние на «RUN\_RIGHT»:



Теперь нужно указать условие для бега влево. Для этого нужно сказать программе: Иначе, если xAxis < 0, то изменить состояние на бег влево. Создать условие «иначе» можно с помощью оператора «else». Так же после него мы можем дописать второе условие, но уже для бега влево:



Осталось только добавить состояние, когда ни одно из условий не выполнилось. То есть xAxis не больше и не меньше нуля, то есть стоит на месте. Для это мы так же воспользуемся оператором «else», но в этот раз не будем писать никаких условий:



Заключение

В этой практической работе я изучил основные принципы создания и программирования анимации в Unity. Полученные знания были закреплены на практике и реализованы в виде игры.

Основные проблемы заключались в сложности устоновки и использования программных пакетов Unity и JetBrains Rider в связи с введенными на территории США санкциями, которые блокируют доступ к вышеуказанным ресурсам.

Список используемой литературы

1. Deep Learning in Gaming and Animations (Vikas Chaudhary, Moolchand Sharma, Prerna Sharma, Deevyankar Agarwal) 2022
2. Game Animation For Dummies Kelly L. Murdock (2005)
3. The Animator's Survival Kit Ричард Уильямс (2001)
4. Кровь, пот и пиксели. Обратная сторона индустрии видеоигр Jason Schreier (2017)
5. C# 7 и .NET Core. Кросс-платформенная разработка для профессионалов Марк Дж. Прайс
6. Unity и C#. Геймдев от идеи до реализации Бонд Д. Г.
7. Unity in Action: Multiplatform Game Development in C# Джо Хокинг (2015)
8. Developing 2D Games with Unity Джаред Халперн (2018)
9. Introduction to Game Design, Prototyping, and Development Джереми Гибсон Бонд (2014)
10. <https://www.youtube.com/watch?v=z-5djm1pRpU>
11. <https://www.youtube.com/watch?v=LNidsMesxSE>
12. <https://www.youtube.com/watch?v=8_KBjd0iaCU>