УДК 004

Программирование базовых механик игры в Unity 3D при помощи Unity Bolt

Ульянов Егор Андреевич
Приамурский государственный университет имени Шолом-Алейхема
Студент

Аннотация
В данной статье рассматривается и описывается реализация базовых механик простой игры-платформера, при помощи визуального программирования, то есть использую графы. Механики игры программируются при помощи визуального программирования в Unity Bolt. Практическим результатом являются работающие механики игры.
Ключевые слова: Unity 3D, Unity Bolt, визуальное программирование

Programming basic game mechanics in Unity 3D using Unity Bolt

Ulianov Egor Andreevich
Sholom-Aleichem Priamursky State University
Student

Abstract
This article discusses and describes the implementation of the basic mechanics of a simple platformer game, using visual programming that is, using graphs. The game mechanics are programmed using visual programming in Unity Bolt. The practical result is working game mechanics.
Keywords: Unity 3D, Unity Bolt, visual programming

Визуальное программирование — способ создания программы для персональных компьютеров путём манипулирования графическими объектами вместо написания её текста. Некоторые авторы представляют визуальное программирование как следующий этап развития языков программирования (следующее поколение). В настоящее время визуальному программированию стали уделять больше внимания, в том числе в связи с развитием мобильных сенсорных устройств (КПК, планшеты). Визуальное программирование может применяться для создания различных типов программного обеспечения, в том числе и для разработки игр. Система визуального программирования в Unity позволяет создавать игровую механику или логику взаимодействия с помощью визуальной графовой системы вместо работы с текстовым представлением кода.

Цель данной статьи рассмотреть возможности игрового движка Unity 3D и модуля Unity Bolt в создании базовых механик платформенных игр.

С.С. Лунин в своей статье описал использование Unity Bolt при разработке модуля, реализующего систему управления игровыми объектами для интегрированной среды разработки и движка Unity3D[1]. С. А. Суродин в своей статье представил сценарий углубленного изучения одного из лучших движков, существующих на данный момент, для создания красивых 2D и 3D игр[2]. В своей работе Р. Ф. Гайнуллин, В. А. Захаров, Е. А. Аксенова изучили инструмент для разработки двух- и трёхмерных игр – Unity 3D[3].

Начинаем создание игры с создаем нового 2D проекта и называем его произвольным именем, а далее импортируем необходимые модули с официального магазина Unity Asset store см. рисунок 1-2.

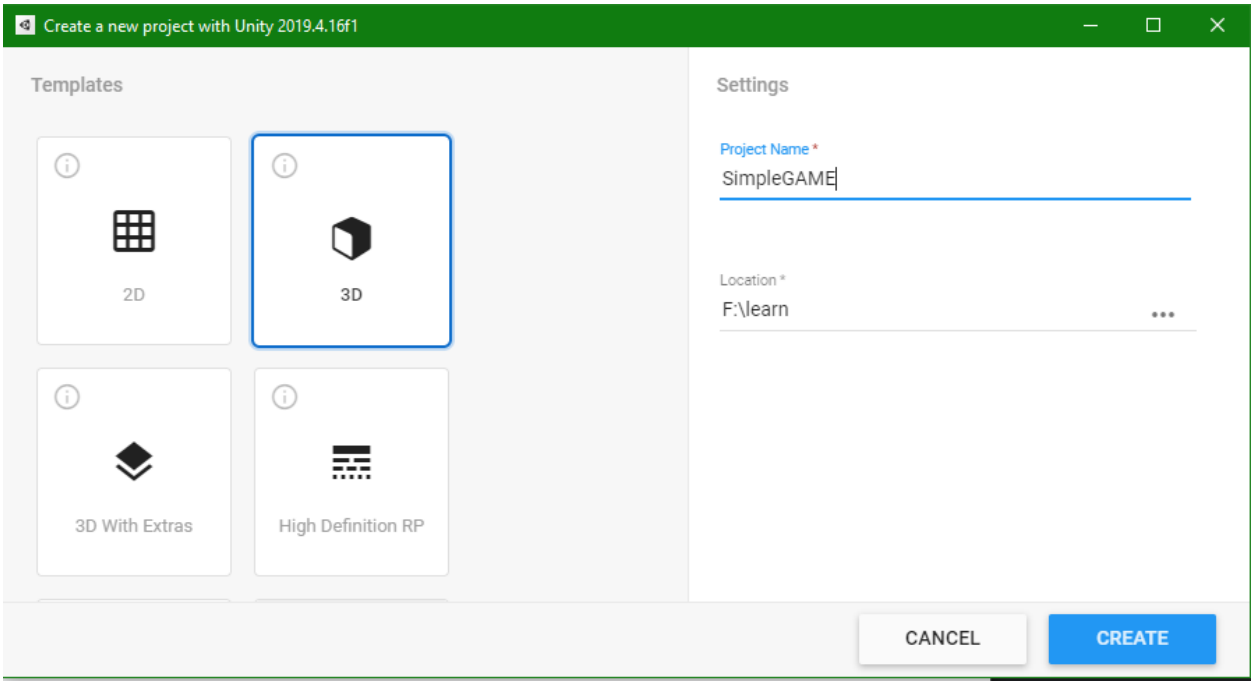

Рис. 1. Создание проекта

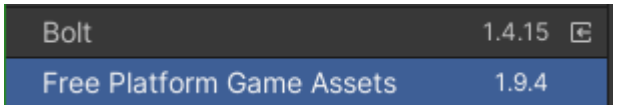

Рис. 2. Импорт модулей в проект

Далее в папке «Scenes» открываем любой понравившийся уровень, в данном случае «Level1», но запуская уровень, персонаж не контролируем, то есть нажатие клавиш не имеет никакого отклика см. рисунок 3-4.

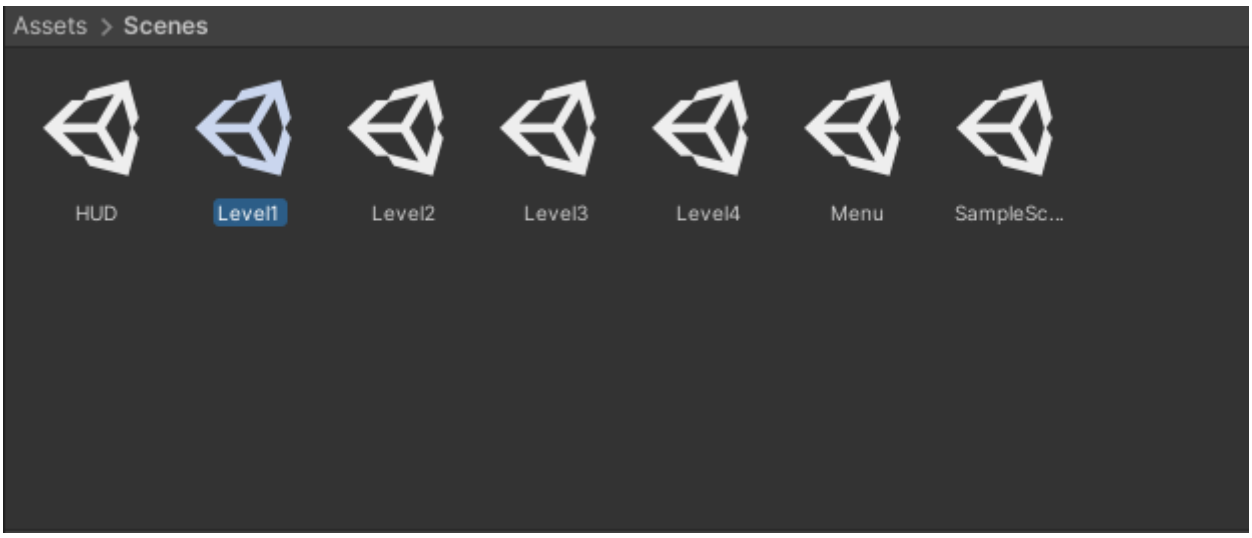

Рис. 3. Открытие уровня

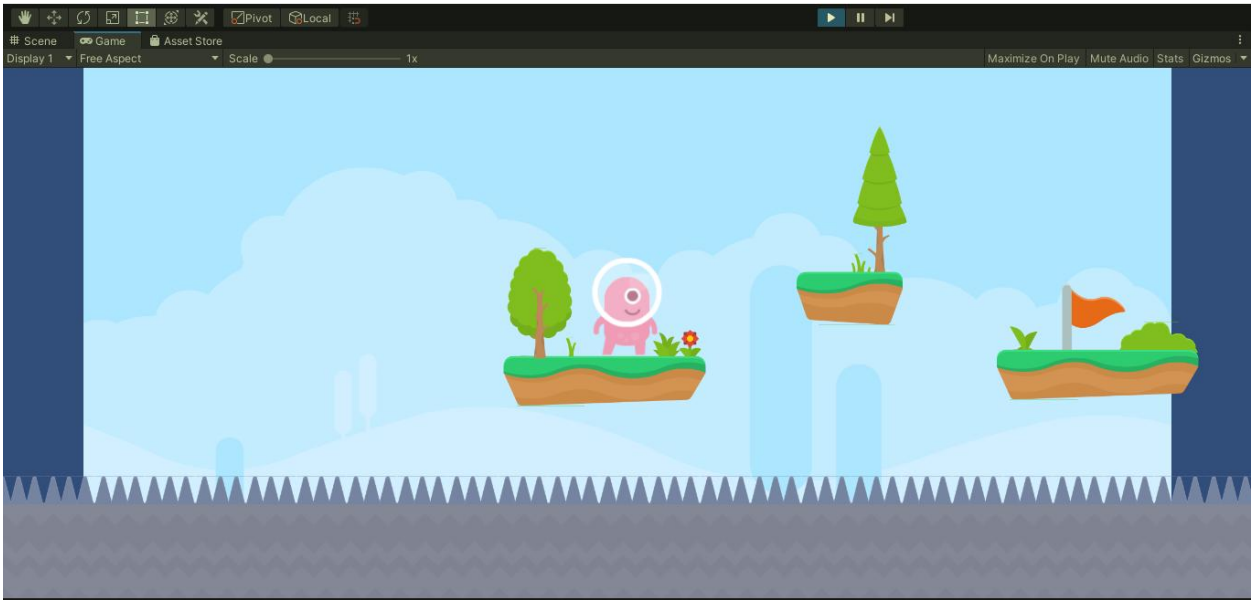

Рис. 4. Отсутствие контроля персонажа

Приступаем к оживлению персонажа, для этого необходимо в окне «Hierarchy» выбрать персонажа «Player» и добавить к нему компонент «Flow Machine» см. рисунок 4.

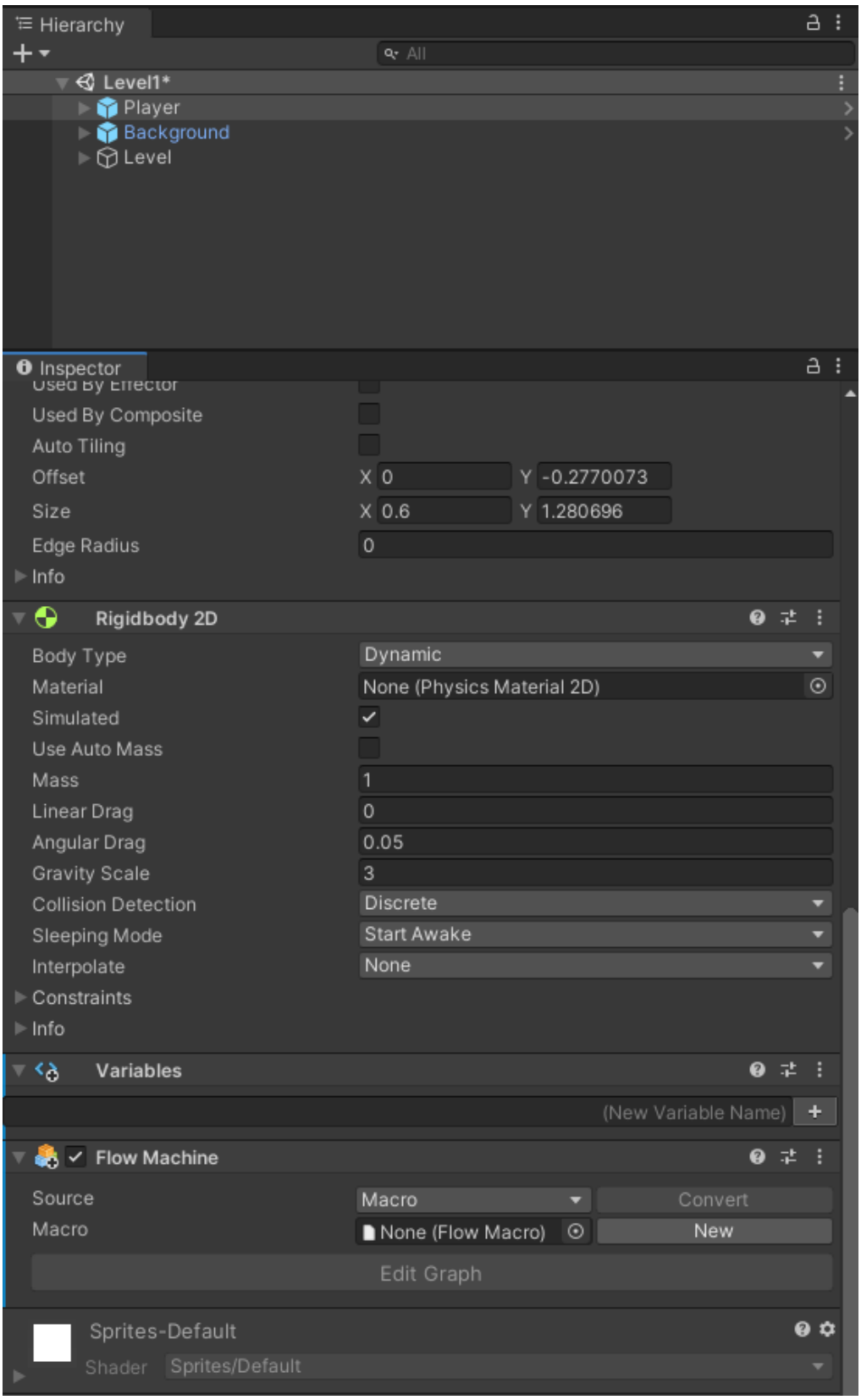

Рис. 4. Добавление компонента

Далее необходимо создать макрос (это многоразовый график, на который можно ссылаться несколькими разными «Flow Machine»), для этого рядом нажимаем рядом с полем «Macro» на «New», создаем новую папку «Macros» и называем файл «PlayerController» см. рисунок 5-6.

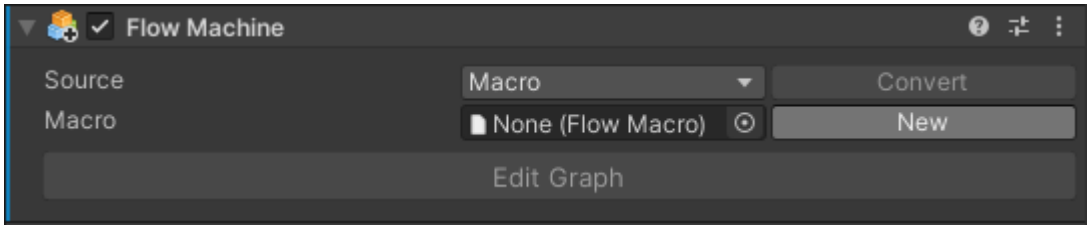

Рис. 5. Создание макроса

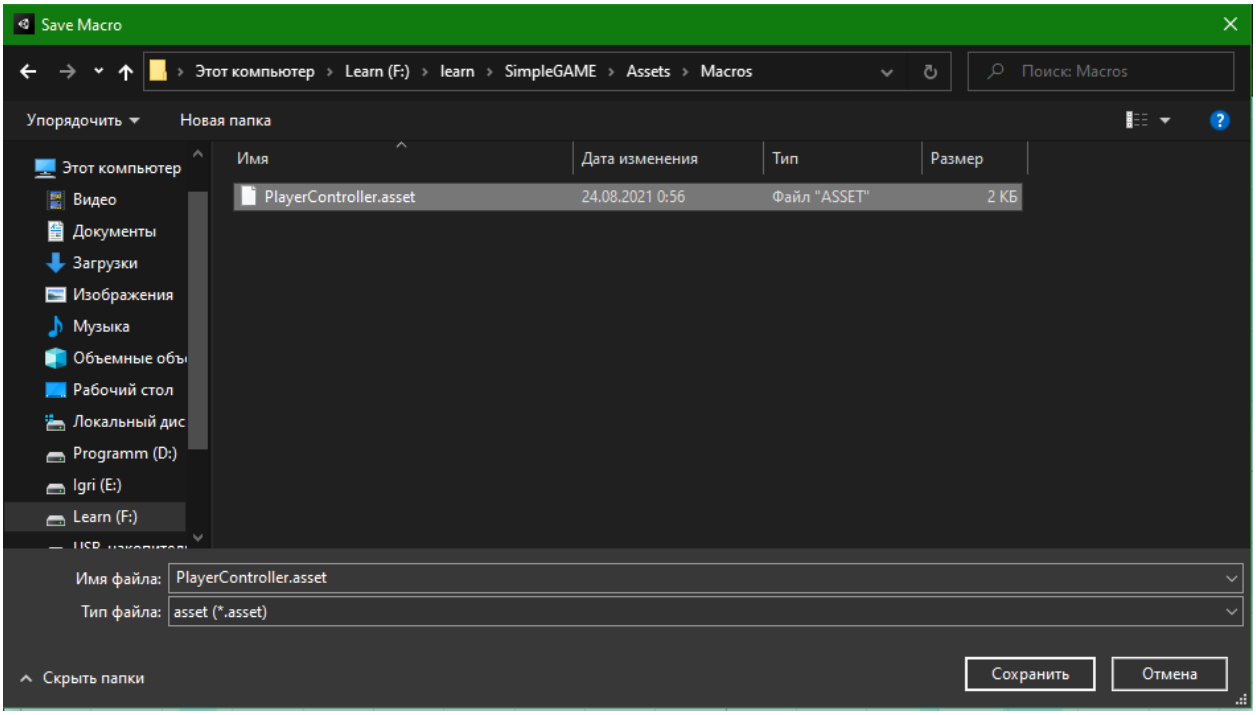

Рис. 6. Создание макроса

Приступаем к визуальному программированию, для этого нажимаем кнопку «Edit Graph». В открывшимся окне для удобства необходимо добавить два окна Variables и Graph Inspector см. рисунок 7-8.

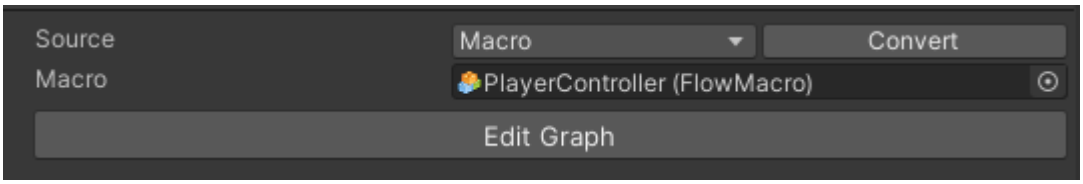

Рис. 7. Редактор графов

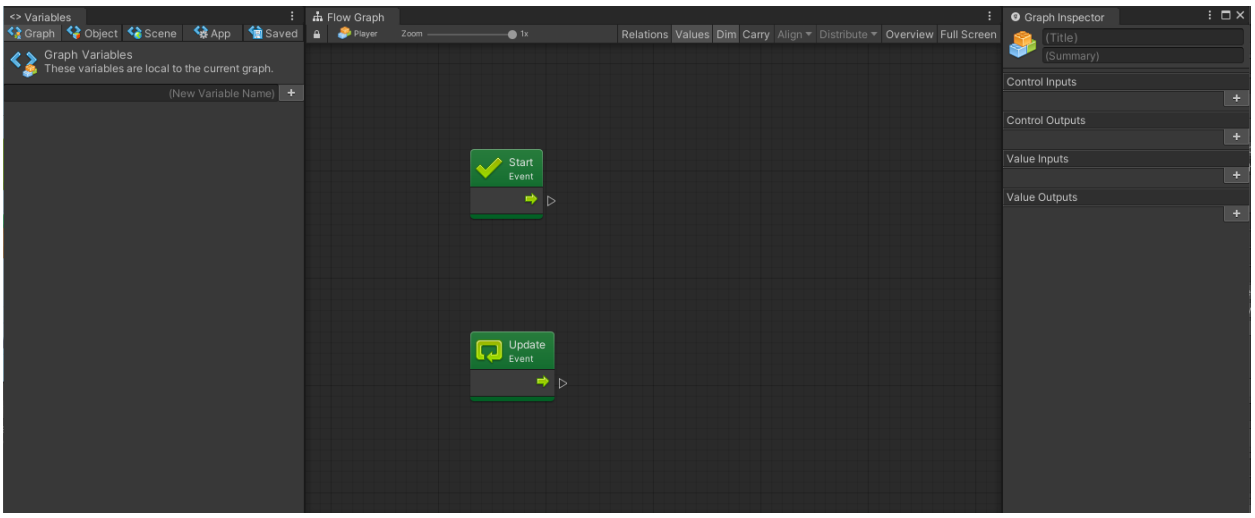

Рис. 8. Добавление необходимых окон

Далее нужно создать необходимые переменные, для этого на вкладке «Object» создаем переменную типа «Float», с название «Speed» (значение скорости персонажа) и присваиваем ей значение 5, а на вкладке «Graph» создаем переменную того же типа, но уже с названием «Movement» (основа для перемещения персонажа по осям) см. рисунок 8-9.

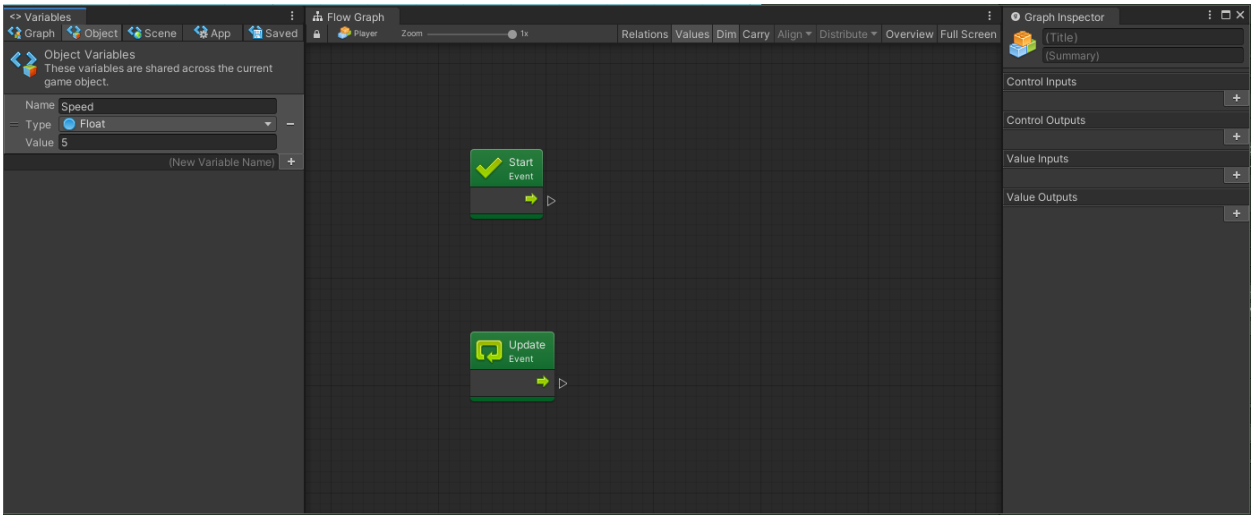

Рис. 8. Добавление переменной «Speed»

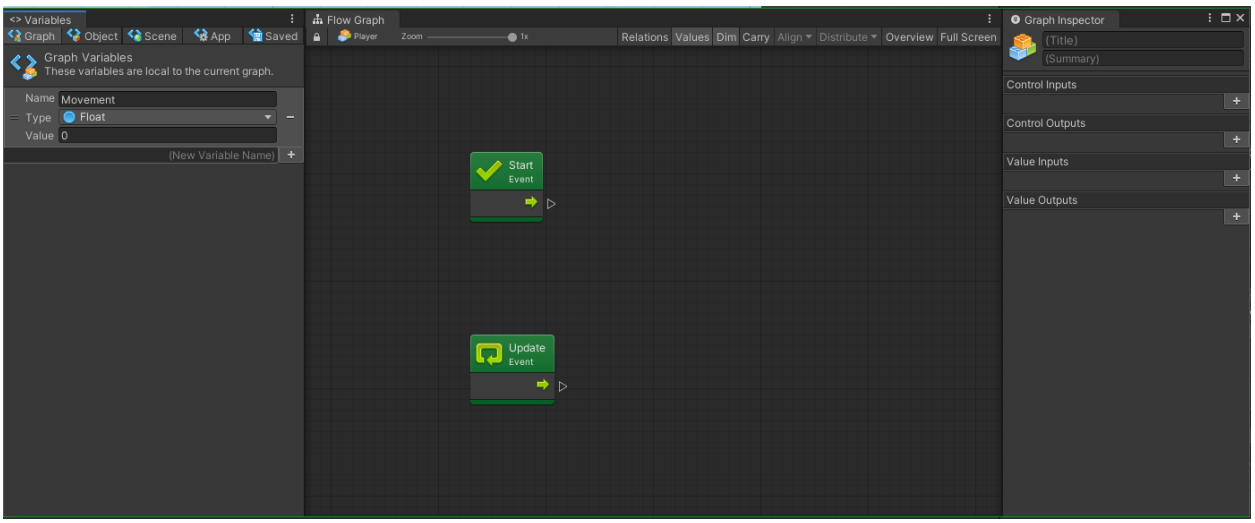

Рис. 9. Добавление переменной «Movement»

Создадим первую блок схему, щелкаем правой кнопкой мыши по свободному полю, и находим «Get Axis» (блок, который будет выдавать значение от -1 до 1, что соответствует направлению движения персонажа влево или вправо, при нажитии соответствующих клавиш). Теперь переносим переменную «Speed» в блок-схему посредством «Drag-and-drop». Далее необходимо значение переменной «Speed» умножить на «Get Axis». Для этого из порта «Get Axis» перетащим соединение, в котором выбираем компонент «Multiply» соединяем, и для вывод значения перемножения создаем новый блок «Set Graph Variable». Соединяем блок «Update» с «Set Graph Variable» чтобы считывать значение горизонтальной оси в каждом кадре см. рисунок 10.

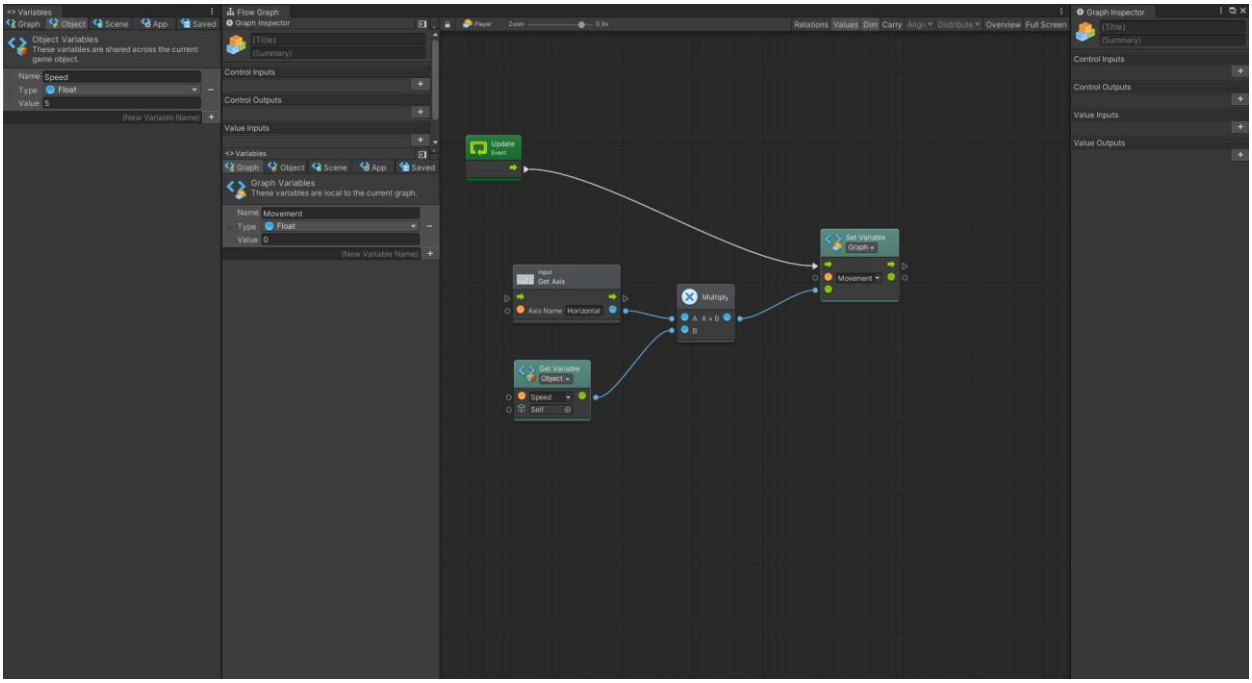

Рис. 10. Метод вычисления направления и скорости персонажа

Далее необходимо добавить физические свойства игроку, для этого в блок-схеме добавим компонент «Get Velocity Rigidbody 2D» и соединим с блоком «Get Y». Затем извлекаем переменную «Movement» создаем блок «Vector 2» и присваиваем полученную скорость компоненту «Rigidbody 2D». Соединяем блоки чтобы получилось, как на рисунке, см. рисунок 11.

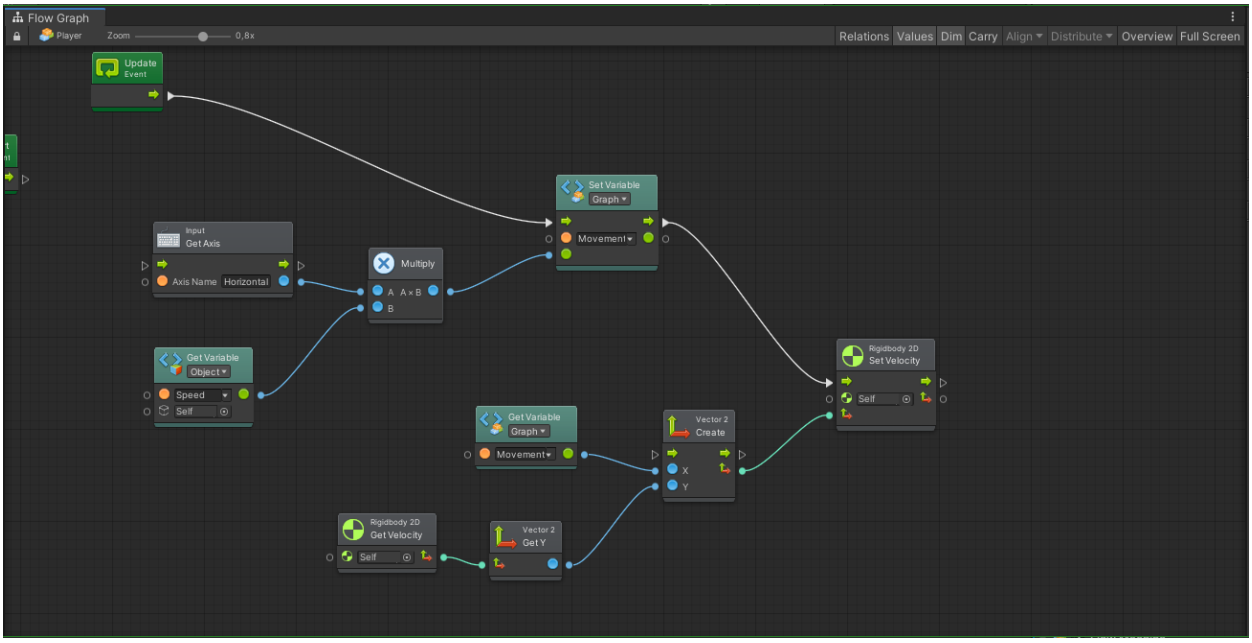

Рис. 11. Присваивание физических свойств игроку

Игру уже можно запустить и убедиться, что логика работает, персонаж двигается влево и вправо в зависимости от нажатий игрока. Персонажу не хватает анимаций, готовых в этом «asset», но не подключенных к логике см. рисунок 12.

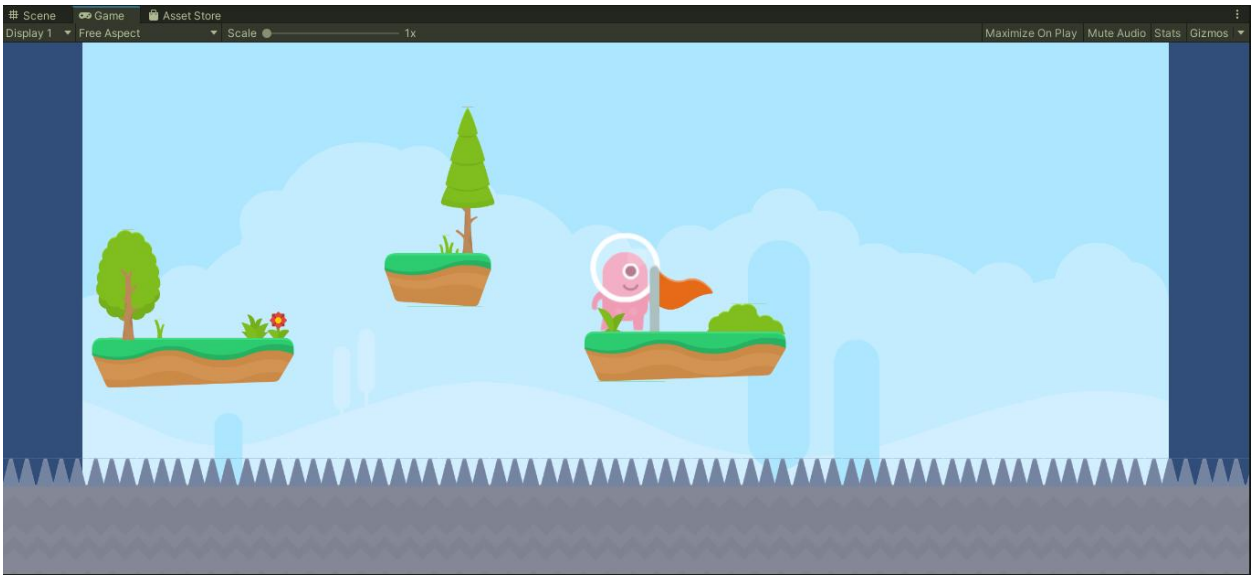

Рис. 12. Движения игрока

В «Animator Controller» видим, что скорость (переменная «Speed») необходимая персонажу для перехода из состояния покоя в состояние ходьбы, была больше 0,01 см. рисунок 13.

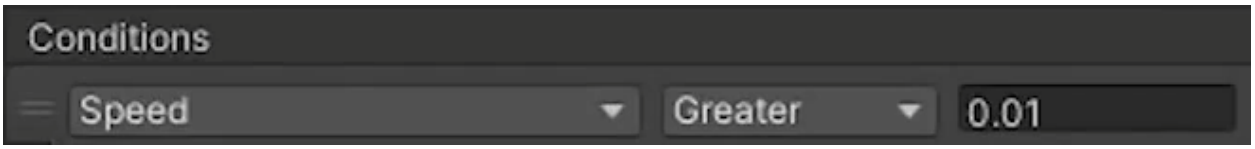

Рис. 13. Скорость для перехода состояний

Для добавления анимаций к персонажу необходимо снова вынести переменную «Movement», и добавить сравнивающий блок «Comparison» (блок поможет определять «сторону взгляда» персонажа), и соединить с «Select» указав значение -1 если условие верно, и 1 если нет. Затем создаем «Vector 3» и соединяем с «Set Local Scale». Подключаем выходной порт условия «A=B» к блоку «Branch» и соединяем с «Set Local Scale». Далее копируем переменную «Movement», соединяем с блоком «Mathf ABS» и соединяем с «Animator Controller», значение параметра «Name» задаем «Speed». Теперь соединяем блоки как на рисунке 14-16.

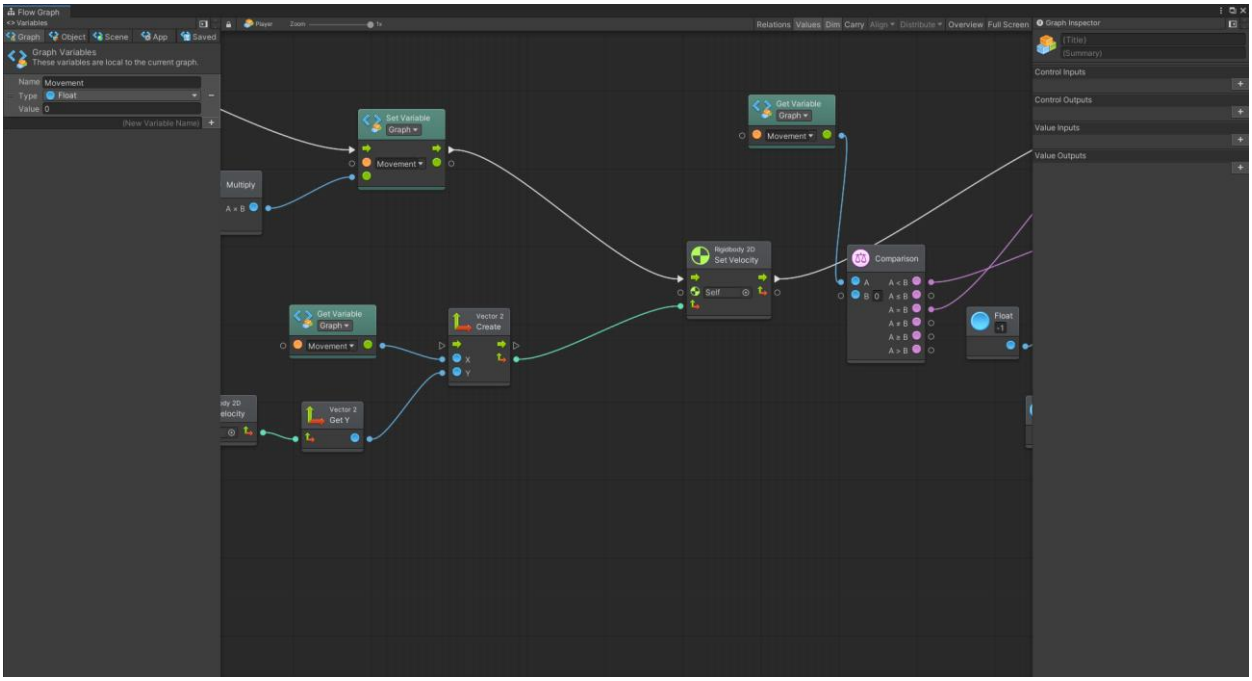

Рис. 14. Готовая блок-схема

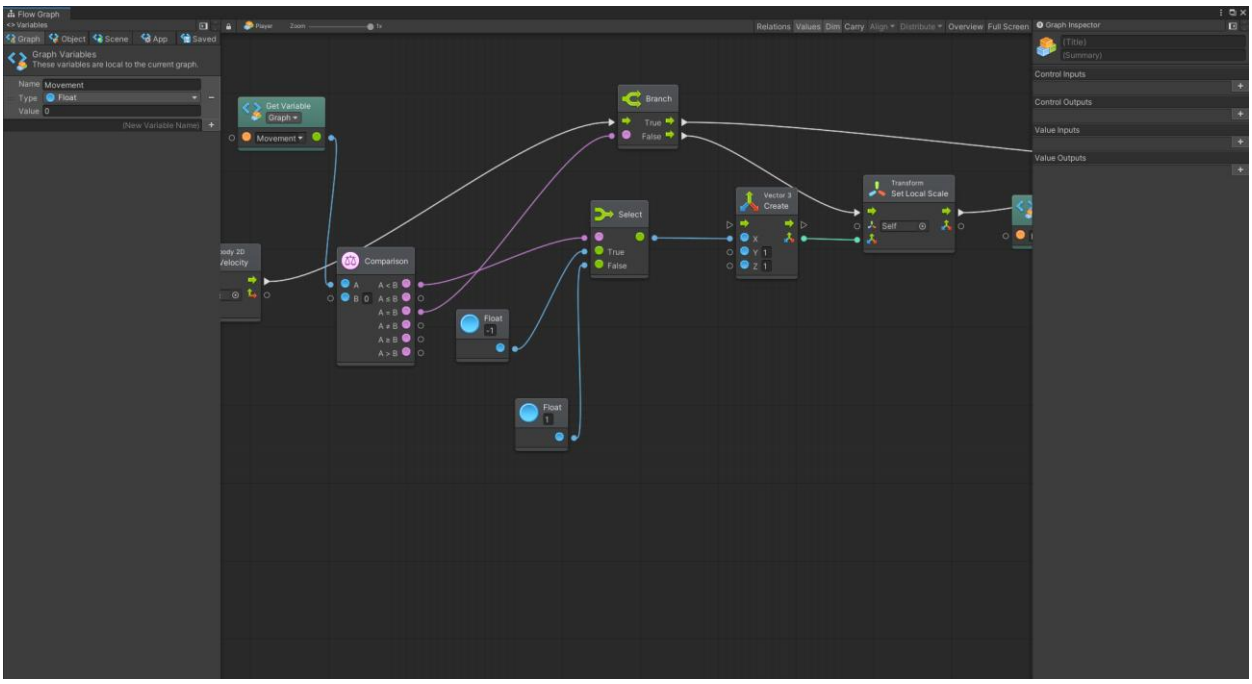

Рис. 15. Продолжение блок-схемы

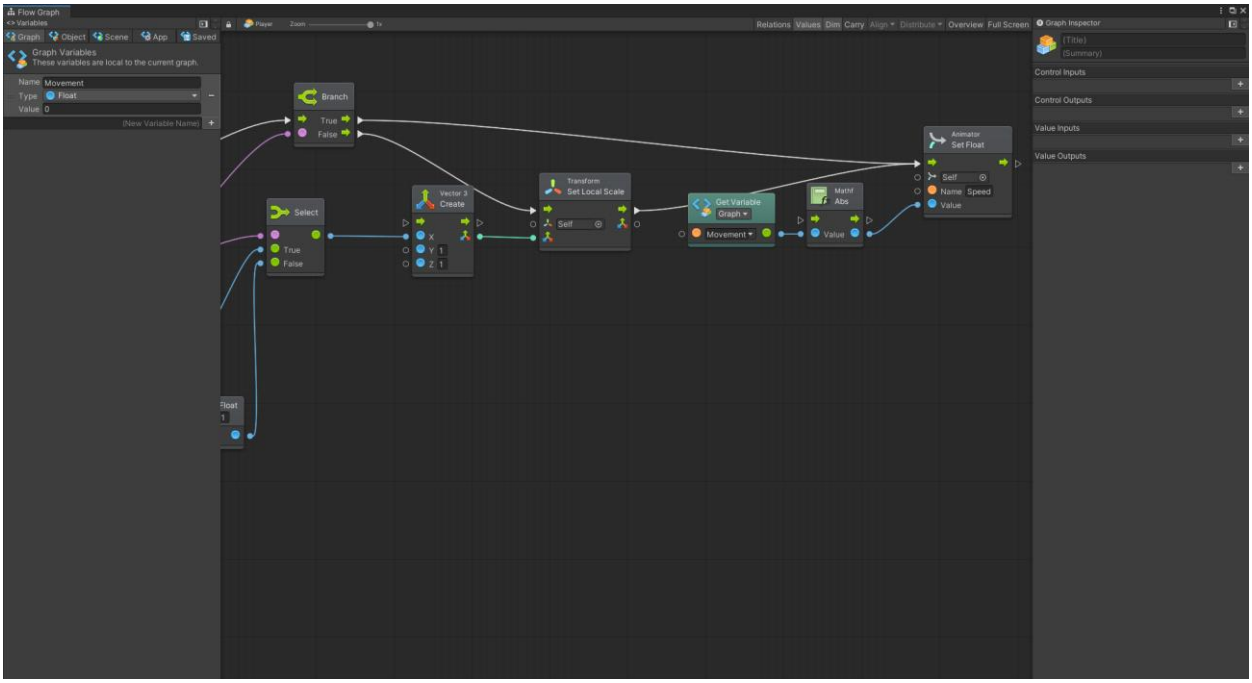

Рис. 16. Продолжение блок-схемы

Далее необходимо протестировать работу логики, то есть готовые механики игры см. рисунок 17-18.

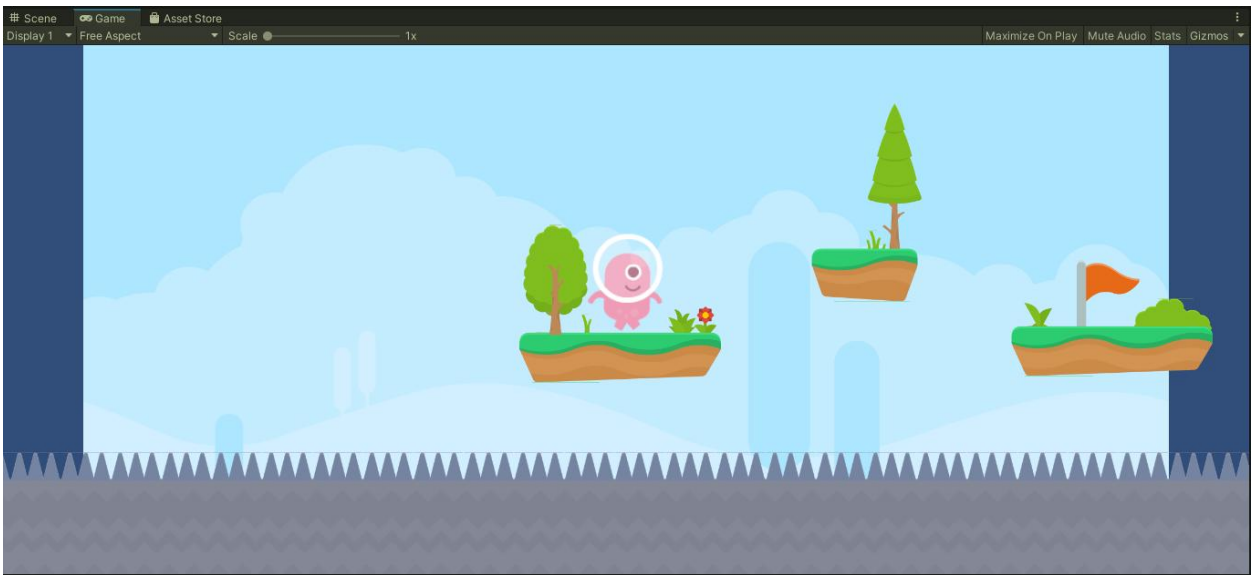

Рис. 17. Работа механик анимации и передвижения вправо

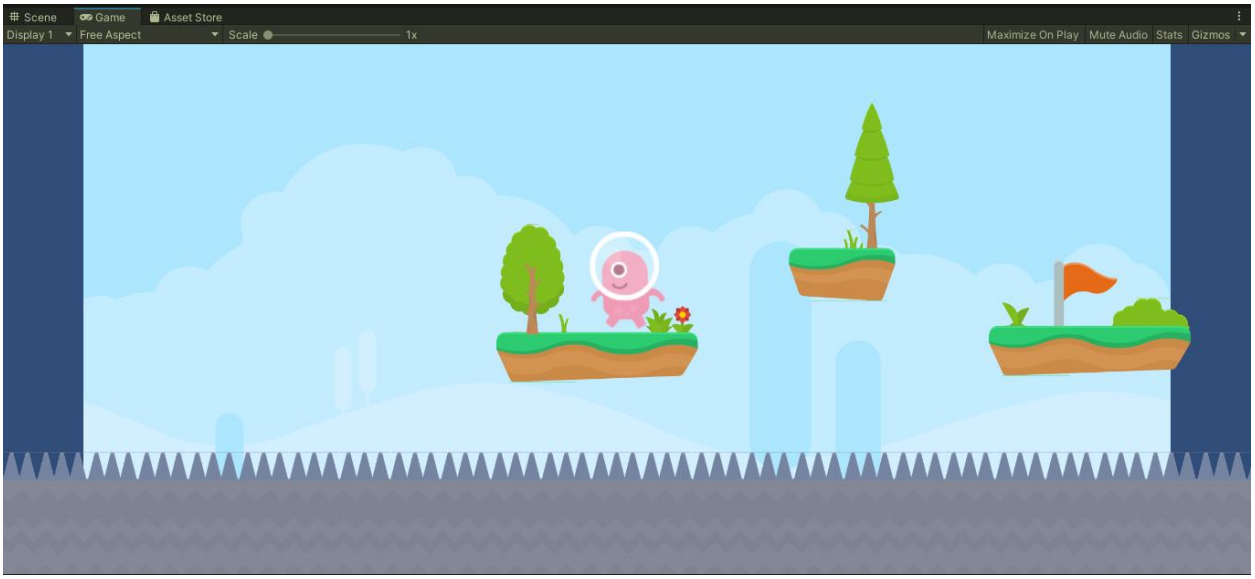

Рис. 18. Работа механик анимации и передвижения влево

В процессе построения логики игры был проведен сравнительный анализ визуального программирования с текстовым аналогом, в ходе которого был принят вывод, что некоторые моменты в визуальном программировании делаются быстрее и понятнее, в следствии чего при должно желании, при помощи такого метода можно разработать игру. В итоге данной работы, были созданы базовые механики для платформенных игр.

Библиографический список

1. Лунин С.С. Разработка подключаемого модуля реализации подсистемы управления игровыми персонажами в среде Unity3D// Информационные технологии, системный анализ и управление (ИТСАУ-2019). 2019. №2. С. 371-376.

2. Суродин С. А. Unity 3D. разработка сценария проектирования в среде Unity 3D// Информатика и вычислительная техника. 2015. №3. С. 504-511.
3. Гайнуллин Р. Ф., Захаров В. А., Аксенова Е. А. Создание 2d игры на Unity 3D 5.4 // Вестник современных исследований. 2018. №4. С. 78-82.
4. https://assetstore.unity.com/packages/tools/visual-scripting/bolt-87491