

МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования

**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**

Кафедра математической кибернетики и компьютерных наук

ПРАВИЛА ОФОРМЛЕНИЯ КУРСОВЫХ И ДИПЛОМНЫХ РАБОТ
КУРСОВАЯ РАБОТА

студента 2 курса 251 группы
направления 09.03.04 — Программная инженерия
факультета КНиИТ
Яшина Егора Александровича

Научный руководитель
доцент, к. ф.-м. н.

А. С. Иванова

Заведующий кафедрой
к. ф.-м. н.

С. В. Миронов

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	3
1 Обзор возможностей игровых движков	4
1.1 Construct 2	4
1.2 Unity	9
2 Сравнение движков	14
2.1 Сложность	14
2.2 Занимаемая память и скорость запуска	14
2.3 Поддерживаемые платформы	15
2.4 Поддержка разработчика	15
3 Разработанная игра	16
3.1 Реализация на Construct 2	16
3.2 Реализация на Unity	24
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	37

ВВЕДЕНИЕ

В современном мире немалую часть рынка развлечений занимают игры, каждая из них создана на своем игровом движке. Игровых движков в мире также имеется достаточное количество, и при выборе нужного разработчик может подобрать под свои нужды и желания. Но как же определить нужный движок? В данной курсовой работе будут сравнены движки Construct 2 и Unity, и на разработанных примерах будет определена предпочитаемая область разработки на этих движках.

1 Обзор возможностей игровых движков

1.1 Construct 2

Construct 2 является движком, основанным на HTML5 и позволяющим создавать двухмерные игры. Отличительной особенностью движка Construct является полное отсутствие кода, вместо него используются события. Событие — это логическое ветвление, состоящее из условия, которое может быть создано с помощью конструкций «И», «Или», и действия, которое может включать в себя как набор последовательных команд так и одиночную команду.

Проект в Construct разделяется на макеты и листы событий. Макет представляет собой рабочую область в которой располагаются игровые объекты. Макет имеет собственные параметры такие как название, размер видимой рабочей области, лист событий который работает в данном макете. [1] Макет пустого проекта представлен на рисунке 1.

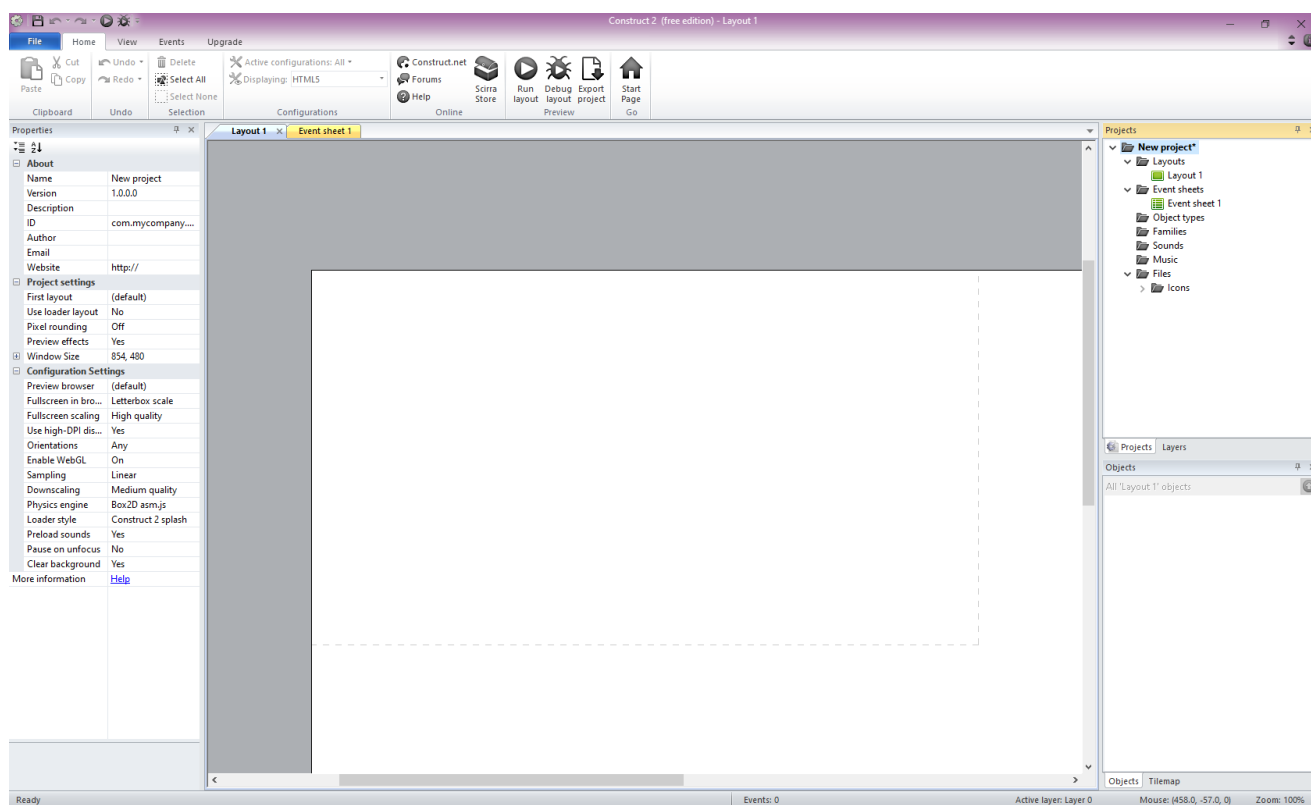


Рисунок 1 – Макет пустого проекта

При создании нового объекта, выбирается тип объекта, которым он является, и в зависимости от типа будут определены параметры и события доступные данному элементу. Все возможные типы объектов приведены на рисунке 2. Для включения определенных средств ввода нужно добавление объекта с типом требуемого средства ввода. [1]

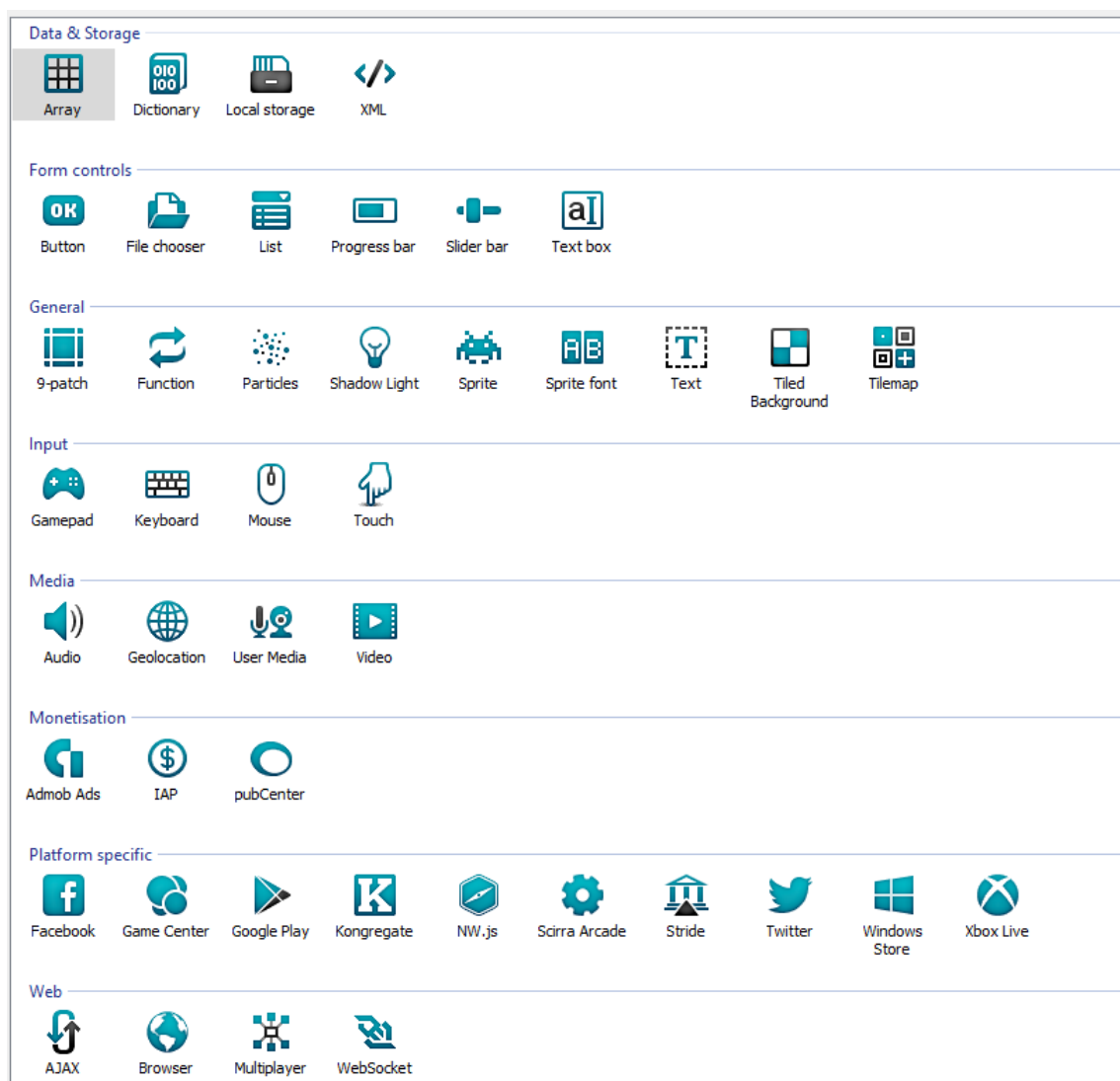


Рисунок 2 – Все типы объектов

Для большинства типов объектов можно добавить новые параметры и возможности с помощью поведения объекта. Все возможные поведения приведены на рисунке 3. Поведение объекта является еще одним важным элементом Construct, поведение задает шаблоны действий объекта в зависимости от параметров поведения. К примеру поведение платформ представляет собой готовый объект игрока для игрового жанра платформер, в данном жанре основными действиями является перемещение и прыжок, поведение платформ предоставляет возможность настроить скорость перемещения и прыжка, также устанавливается базовое управление на клавиши влево и вправо движение в соответствующие стороны и на клавишу вверх устанавливается прыжок. [2]

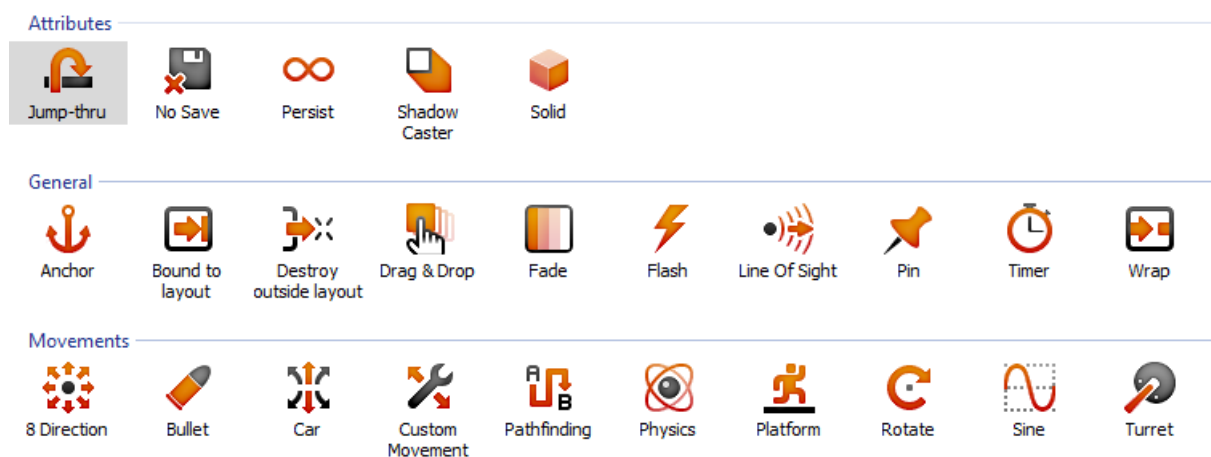


Рисунок 3 – Все поведения объекта

Construct предоставляет инструменты для создания новых объектов, требующих графическое изображение, внутри движка. Так при создании нового объекта типа спрайта открывается графический редактор который, позволяет нарисовать новое изображение в движке либо загрузить уже существующее, позволяет настроить границы соприкосновения объекта и исходную точку, также внутри редактора можно настроить анимации объекта и их количество. Пример редактора приведён на рисунке 4.

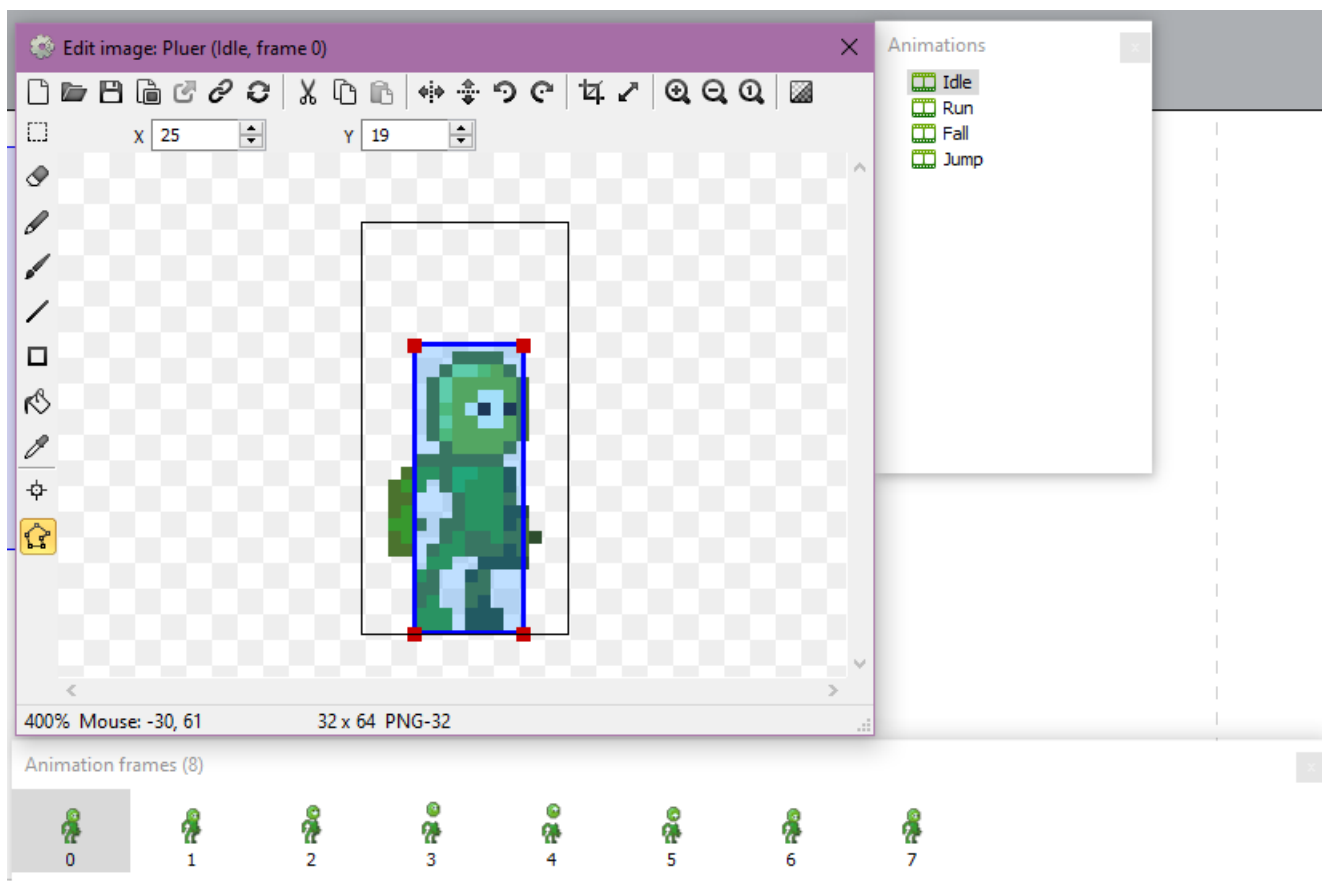


Рисунок 4 – Все поведения объекта

Лист событий представляет собой последовательное расположение всех событий проекта. Пример листа событий приведен на рисунке 5. Для добавления нового события на листе событий выбирается объект и условие допустимое движком, после этого выбирается другой или уже выбранный объект и допустимое движком действие с этим объектом. Примером может являться перемещение персонажа влево при нажатии клавиши A, для этого проект должен включать объект типа клавиатуры для считывания нажатой клавиши и объект типа платформер для создания перемещения. Как логическое условие берется нажатие клавиши A, действием будет являться симуляция передвижения влево. Вид этого события приведён на рисунке 5.

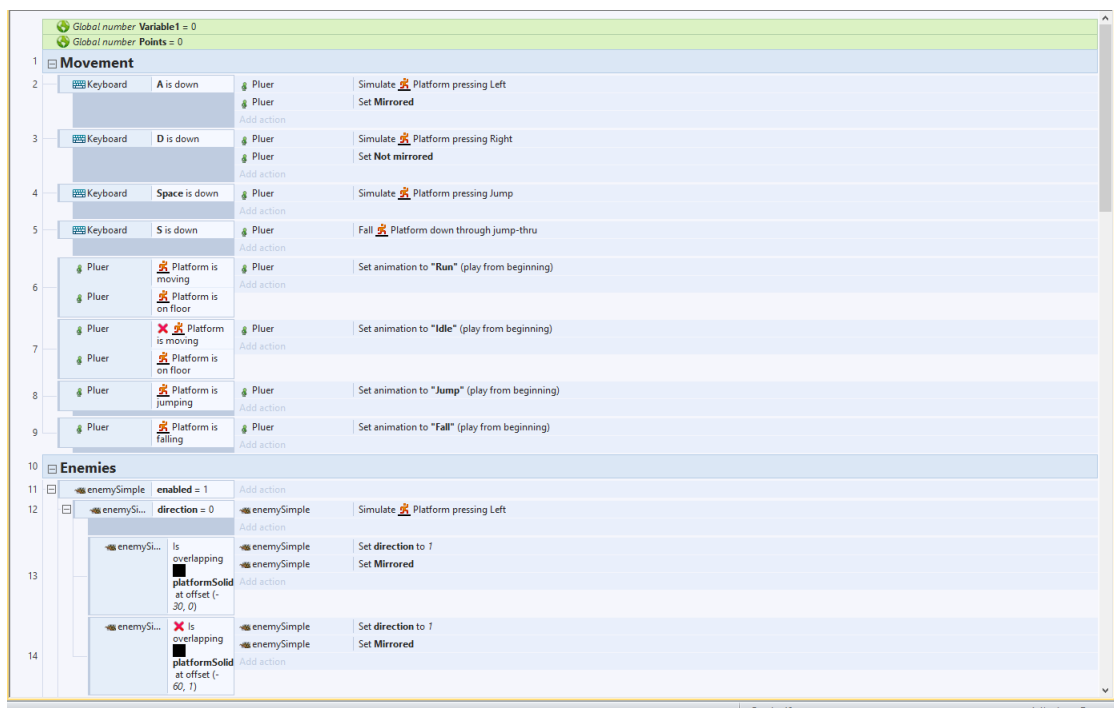


Рисунок 5 – Лист событий

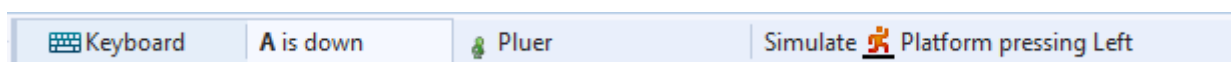


Рисунок 6 – Пример события

Construct является проприетарным движком. Он имеет бесплатную версию с ограничениями, то есть отсутствуют функции и возможность коммерческого использования, также имеется лимит по количеству событий. Существует две платные версии: одна для бизнеса другая для персонального использования. Все отличия бесплатной версии от платных представлены в таблице 1. Различия в функционале этих версий отсутствуют в них присутствуют все возможные функции без ограничений, но существует условие что персональную версию нельзя использовать после заработанных 5000\$ с помощью этого движка. Соответственно платные версии отличаются в цене 200\$ за персональную версию и 500\$ за бизнес. [3]

Таблица 1 – Отличия в лицензиях Construct 2

Характеристика	Бесплатная лицензия	Платные лицензии
Вкладка просмотра отладчика	Отсутствует	Присутствует
Профайлер	Отсутствует	Присутствует
Точки останковки для event	Отсутствует	Присутствует
Создание iOS приложений	Отсутствует	Присутствует
Создание Android приложений	Отсутствует	Присутствует
Создание Windows приложений	Отсутствует	Присутствует
Создание Mac приложений	Отсутствует	Присутствует
Создание Linux приложений	Отсутствует	Присутствует
Создание Amazon Store приложений	Отсутствует	Присутствует
Создание игр для Wii U	Отсутствует	Присутствует
Внутриигровые покупки	Отсутствует	Присутствует
Максимальное количество событий	100	Неограниченно
Максимальное количество сцен	4	Неограниченно
Максимальное количество особых эффектов	2	Неограниченно
Подпапки проекта	Отсутствует	Присутствует
Поиск событий	Отсутствует	Присутствует
Панель расположения z-порядка	Отсутствует	Присутствует
Семейства объектов	Отсутствует	Присутствует
Мгновенный просмотр	Отсутствует	Присутствует

1.2 Unity

Unity является игровым движком который позволяет создавать как двухмерные так и трехмерные игры. Особенностью Unity является то что все объекты являются сущностями. Сущность — это представление любого объекта, который может быть представлен в игре. Сущностью могут являться медиа файлы принадлежащие проекту, файлы созданные Unity, скрипты.

Проект в Unity представляет безграничную рабочую область, в которой расставляются все игровые объекты задействованные в данной сцене. Рабочая область использует собственные значения для определения размера объекта внутри себя называемые юнитами, в следствии этого при загрузке нового графического файла требуется указать сколько пикселей будет внутри 1 юнита. В рабочей области можно располагать объекты как в трехмерном так и в двухмерном пространстве. [4] Unity позволяет производить переключение между пространствами в любой момент разработки проекта, это позволяет определить расположение объектов относительно друг друга в разработке двухмерных игр, и позволяет производить разработку игр со смешанным пространством. [5] Пример расположения двухмерных объектов в трехмерном пространстве приведен на рисунке 7.

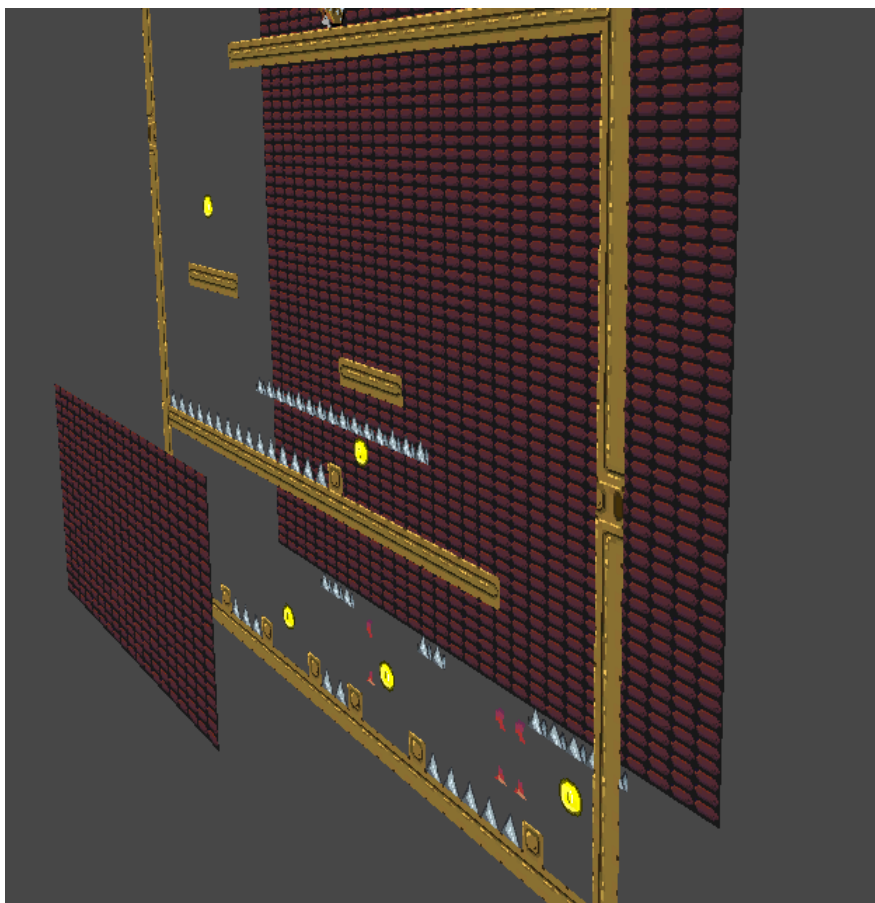


Рисунок 7 – Расположение двухмерных объектов в трехмерном пространстве

Любой игровой объект представленный в рабочей области обладает компонентом `transform`, то есть компонентом которое определяет положение объекта внутри сцены. Для отображения и взаимодействия объекта с другими объектами требуется добавить новые компоненты. К примеру для отображения используется компонент `sprite renderer` которое производит рендер изображения с параметрами свойства `transform`, физические свойства объекта определяются самим движком и инициализируются компонентами `rigidbody2d` или `rigidbody` в зависимости от требуемого пространства. Пример объекта с несколькими компонентами представлен на рисунке 8.

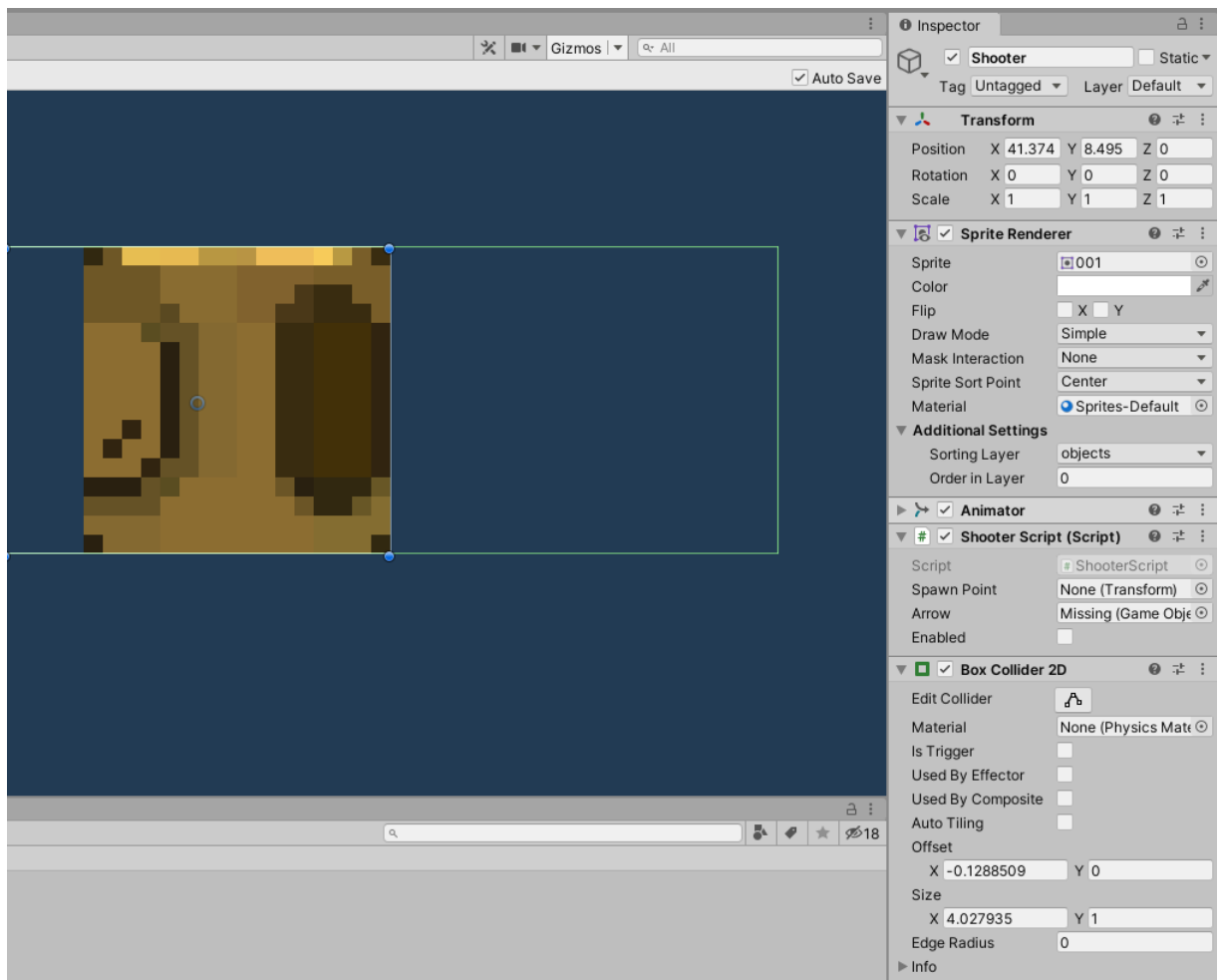


Рисунок 8 – Объект с несколькими компонентами

Animator является свойством требуемым для отображения анимации, Unity имеет инструменты для создания новых анимаций и создание условий перехода между разными видами анимаций. Для создания новой анимации достаточно создать пустую анимацию и добавить её к объекту, после этого для добавления новых кадров нужно перенести графические файлы в поле анимации, также анимация может иметь вызов функции по достижению определённого времени внутри себя. [4] Пример анимации на рисунке 9.

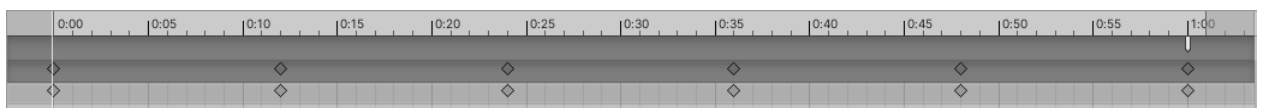


Рисунок 9 – Анимация в Unity

Переход анимаций в Unity может происходить при определённом условии, animator может включать в себя переменные типов bool, int, float, trigger и при достижении нужного состояния переменных для смены анимации произойдёт смена на новую анимацию с первого кадра. Каждый переход будет

отображаться стрелочкой направляющей в анимацию, в которую будет осуществлен переход. Пример Animator с переходами анимаций представлен на рисунке 10.

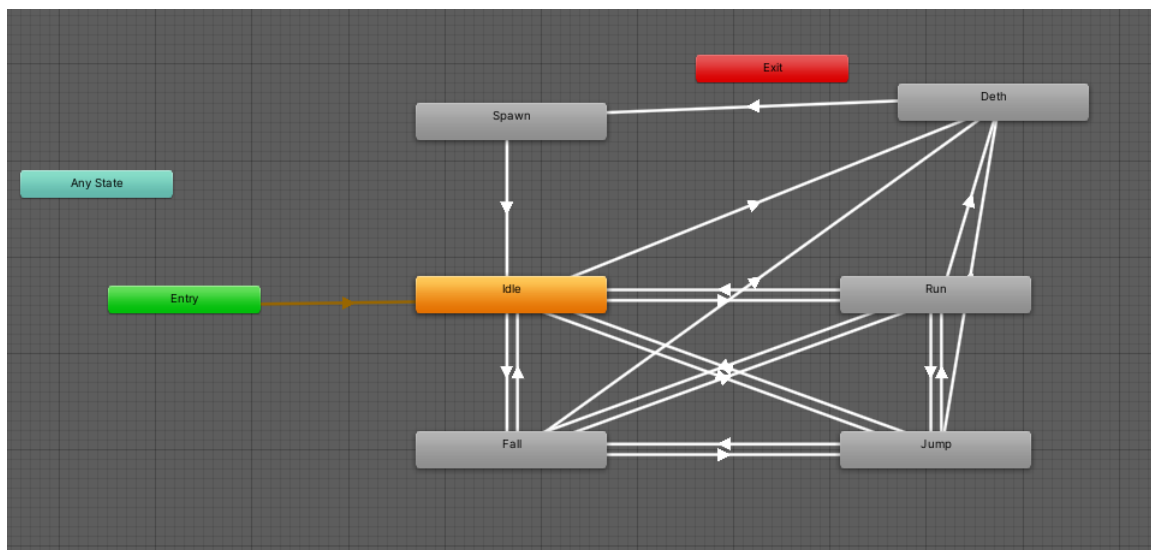


Рисунок 10 – Animator с переходами анимаций

Для изменения параметров свойств объектов программным путём используются скрипты. В Unity скрипты могут быть написаны как на C# — Unity Scripting Api [6] так и на JavaScript — UnityScript. Пустой скрипт включает в себя библиотеки требуемые для написания на языке Unity, и две функции Start и Update. Start перед первым кадром. Функция Update вызывается в проекте каждый кадр. [7] Пустой скрипт на C# выглядит так:

```

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class NewBehaviourScript : MonoBehaviour
6  {
7      // Start is called before the first frame update
8      void Start()
9      {
10
11      }
12
13      // Update is called once per frame
14      void Update()
15      {

```

16
17
18

}
}

Форма распространения Unity является проприетарной. Видов подписок четыре: Personal, Plus, Pro, Enterprise. Все подписки доступны для коммерческого использования но имеется ограничение по годовому заработку пользователя определенной лицензии. Различия подписок еще заключаются в возможностях при разработке приложения. [8] Все отличия подписок представлены в таблице 2.

Таблица 2 – Отличия в подписках Unity

V — в составе, X — не входит в состав, + — доступно за дополнительную плату				
Характеристика	Personal	Plus	Pro	Enterprise
Ограничения по годовому заработку	≤ 100000\$	≤ 200000\$	> 200000\$	> 200000\$
Pro Editor UI	X	V	V	V
Управление заставочным экраном	X	V	V	V
Пакет графических ресурсов высокого разрешения	X	X	V	V
Доступ к исходному коду	X	X	+	+
Специальные отраслевые решения	X	X	X	+
Advanced Cloud Diagnostics	X	V	V	V
Основная аналитика	X	V	V	V
Analytics: 50 ГБ экспорта необработанных данных в месяц	X	X	V	V
Доступ к Unity Learn Premium	+	V	V	V
Приоритетный доступ к консультантам по успеху Unity	X	X	V	V
Приоритетная очередь клиентской поддержки	X	X	V	V
Premium Support	X	X	+	+
Встроенные услуги Success Service	X	X	X	+
Стоимость в месяц	Бесплатно	40\$	150\$	Гибкая цена

Также существует вариант студенческой подписки для этого студент должен иметь возраст, юридически достаточный для согласия на сбор и обработку персональных данных, и быть зарегистрированным в GitHub Student Developer Pack. Эта подписка является бесплатной и предоставляет возможности который в обычной нет. Этими возможностями являются неограниченный доступ к Learn Premium, пять рабочих мест в Unity Teams Advanced, темная тема пользовательского интерфейса и диагностика реального времени в облаке.

2 Сравнение движков

2.1 Сложность

При знакомстве с новым движком пользователь может запутаться в его строении и сделать решение сменить среду разработки.

Хоть в Unity большинство скриптов пишется на C#, который проще чем C++, и сам движок предоставляет возможность создать все, что разработчик может себе представить. Но именно множество возможностей и ошеломляет нового пользователя, который не знает с чего лучше начать изучение. Unity предоставляет бесплатные обучающие уроки, однако в большом количестве информации очень легко запутаться, особенно если материал затрагивает темы не изученные пользователем. С другой стороны за всё время существования Unity было создано глобальное сообщество разработчиков, с его помощью можно найти требуемый материал если не на родном языке то на английском точно. Из-за сложности в начале разработки происходит ограничения для людей, которые не знают основ требуемых для создания нового проекта. [4]

Construct является движком, который значительно проще Unity, это добивается отсутствием кода. Использование поведения объектов и событий сокращает код используемый в скриптах на Unity до одного события. Они как и поведения будут понятны почти каждому, построение нового события может быть переводом простого предложения в логическое высказывания. Эти возможности позволяют разработку на Construct почти каждому человеку.

2.2 Занимаемая память и скорость запуска

При установке движков иногда может возникнуть вопрос доступной памяти для него. И Construct с Unity имеют значительную разницу в весе на диске. Construct не требует дополнительных установок кроме самой версии движка которая занимает около 120МБ. Unity же рекомендует установить программу для управлением версиями и проектами называемую — Unity Hub, которая занимает почти 200МБ, в то время сама версия Unity весит около 6.5 ГБ.

Следовательно из-за разности веса эти движки имеют совсем разное время запуска. В то время как Construct в среднем за несколько секунд и можно сразу же заниматься разработкой, Unity может запускать пустой проект в течении нескольких минут.

2.3 Поддерживаемые платформы

В Construct экспорт проекта происходит с помощью перевода на JavaScript. Тем самым разработчик может создать игру для платформ которые имеют поддержку Web. [?]

При установке Unity разработчик должен выбрать требуемые ему модули сборки проекта. Для движка существуют модули для любой платформы созданной на данный момент.

2.4 Поддержка разработчика

Разработчики движка Construct 2 ещё поддерживают и обновляют движок, также недавно они выпустили новую версию Construct — Construct 3. На данный момент нововведений в новой версии не так много и инструментов последних версий не имеет значительных различий между собой. Однако выпуск нового движка означает, что Construct 2 в скором времени не будет получать новые обновления от разработчиков. [2]

Поддержка Unity продолжается и сам движок получает обновления довольно часто, но при каждой новой версии требуется скачивание и установка нового движка с нуля. Также каждая новая версия иногда не имеет значительных изменений, но в другой версии могут ввести новые возможности для разработчиков, и в месте с ней появляются новые баги, которые не известно сколько будут чинить, ещё новая версия может изменить работу старого функционала тем самым проект может перестать работать, и подстройка под новый функционал может занять много времени и сил. Тем самым постоянная поддержка движка может как положительно повлиять на разработку так и полностью испортить идеально работающую программу. [?]

3 Разработанная игра

3.1 Реализация на Construct 2

Для разработки игры на Construct 2 и Unity был выбран жанр платформер в двухмерном пространстве.

При первом запуске программы был создан стартовый экран с указанием клавиш управления. Вид стартового экрана приведён на рисунке 11. Для реализации экрана был использован новый макет включающий в себя объект типа `sprite` для отображения заднего рисунка и три объекта типа `text` для отображения текста с клавишами управления.



Рисунок 11 – Стартовый экран

Для реализации смены экрана был подключён объект типа `keyboard` для считывания нажатия клавиши на клавиатуре. Также на листе событий было добавлено событие, которое при нажатии на любую клавишу клавиатуры производило переход на новый макет с листом событий. Событие представлено на рисунке 12.

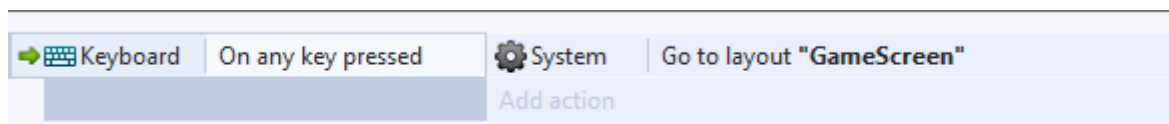


Рисунок 12 – Событие смены макета с листом событий

После смены открываются макет и лист событий, в которых создан игровой уровень. Вид уровня представлен на рисунке 13. В макете используется 13 объектов взаимодействующих с игроком, 2 объекта типа sprite для отображения заднего фона — Back и карты уровня — Tiles, и объект типа keyboard для считывания нажатия клавиши на клавиатуре .

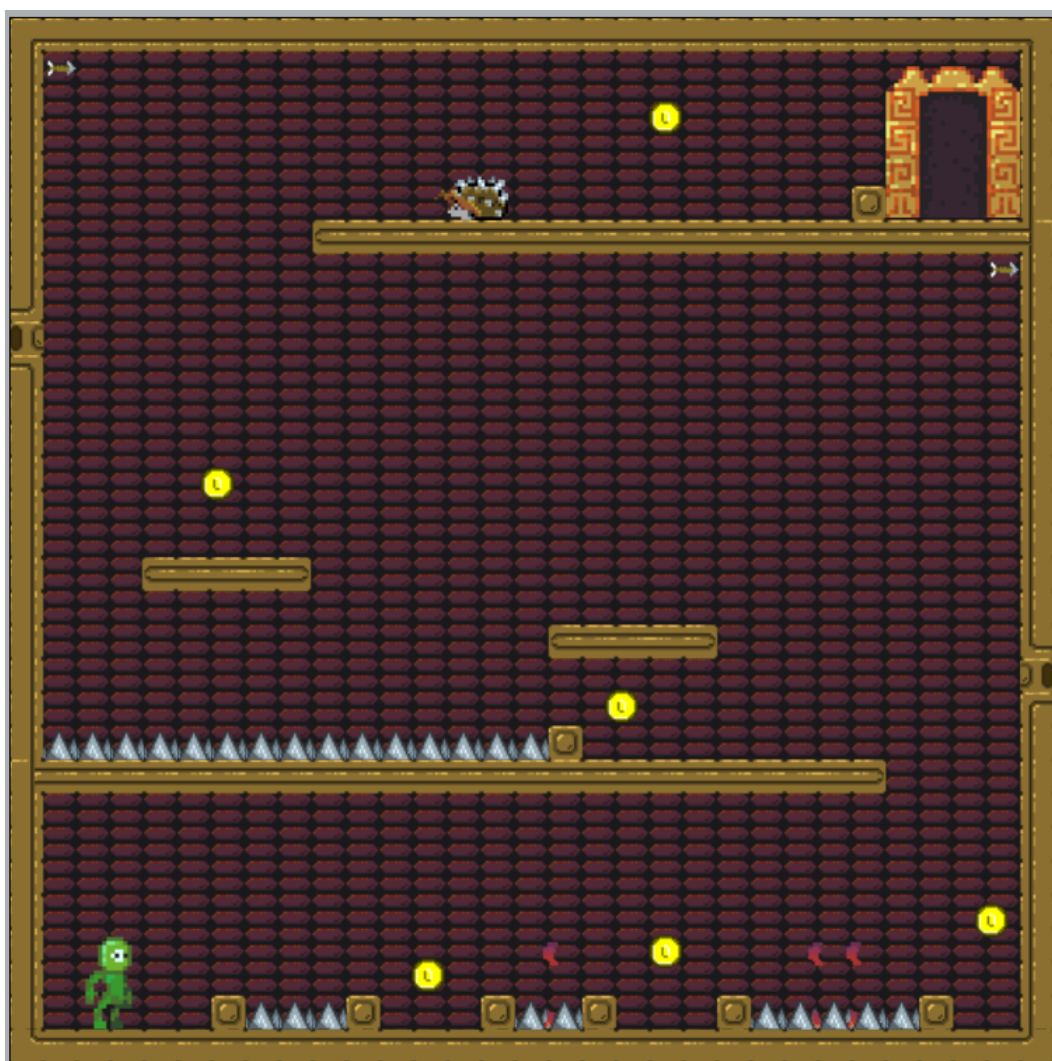


Рисунок 13 – Игровой уровень

Для перемещения игрока используется объект Pluer типа sprite с поведением platform для определения управления игрока. В текущем уровне были изменены параметры поведения platform, изменена максимальная скорость игрока, сила пражка, возможность двойного прыжка и отключены предустанов-

ленные клавиши управления. Все параметры игрока представлены на рисунке 14.

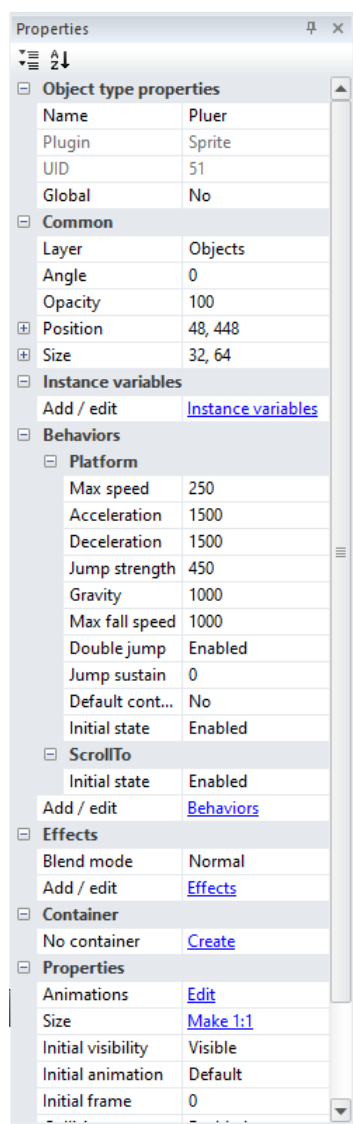


Рисунок 14 – Параметры игрока

Для перемещения игрока влево создано событие, которые при нажатии клавиши A производит симуляцию перемещения влево с помощью возможностей движка и переворот модели игрока для отображения того, что он направлен в левую сторону. Аналогичный процесс для перемещения игрока вправо, только для клавиши D и игрок отображается не перевёрнутым. Для реализации прыжка происходит событие, которое производит симуляцию прыжка при нажатии клавиши Space. События представлено на рисунке 15. Также чтобы игрок не проходил сквозь стены объекту Tiles было добавлено поведение solid.




Keyboard	A is down	Puer	Simulate  Platform pressing Left
		Puer	Set Mirrored
		Add action	
Keyboard	D is down	Puer	Simulate  Platform pressing Right
		Puer	Set Not mirrored
		Add action	
Keyboard	Space is down	Puer	Simulate  Platform pressing Jump
		Add action	

Рисунок 15 – События управления игрока

При перемещении игрок также меняет свою анимации для более приятной игры. Смена анимации происходит с помощью событий сравнивающих текущее состояние игрока и устанавливающие текущую анимацию в зависимости от него. Если игрок перемещается по земле то устанавливается анимация бега, если не двигается на земле то анимация бездействия, если прыгает то прыжка, если падает то падения. События приведены на рисунке 16.







Puer	 Platform is moving	Puer	Set animation to "Run" (play from beginning)
Puer	 Platform is on floor	Add action	
Puer	 Platform is moving	Puer	Set animation to "Idle" (play from beginning)
Puer	 Platform is on floor	Add action	
Puer	 Platform is jumping	Puer	Set animation to "Jump" (play from beginning)
		Add action	
Puer	 Platform is falling	Puer	Set animation to "Fall" (play from beginning)
		Add action	

Рисунок 16 – Смена анимации игрока

Во время игры дополнительной задачей игрока является сбор монет которые являются объектом типа sprite. При взаимодействии с монетой игрок получает одно очко и монета исчезает, это реализовано с помощью события, которое при прохождении игрока через монету уничтожает объект монеты и увеличивает глобальную переменную Points на единицу. Событие приведено на рисунке 17.

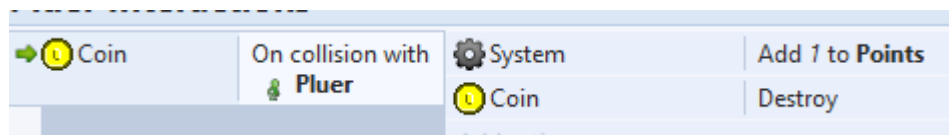


Рисунок 17 – Подбор монеты

Первой опасностью для игрока являются объекты Spikes и Fire, которые имеют тип Sprite. При взаимодействии с ними игрок погибает и начинает уровень сначала. Смерть игрока реализована с помощью двух событий которые при прохождении игрока через объект Spikes или Fire уничтожают объект игрока с которым произошло взаимодействие. События приведены на рисунке 18.

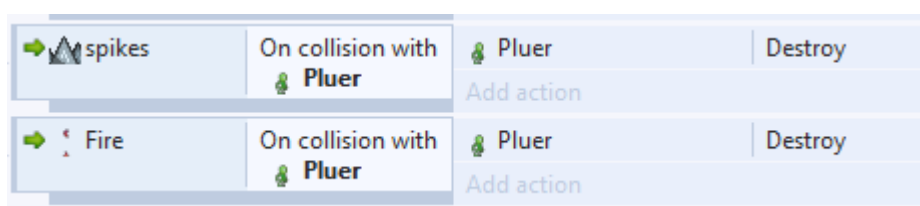


Рисунок 18 – Взаимодействие игрока с объектами Spikes и Fire

При смерти игрока происходит перезагрузка уровня, она реализована с помощью события, которое при уничтожении объекта Puer перезапускает текущий макет и обнуляет глобальную переменную Points Событие приведено на рисунке 19.

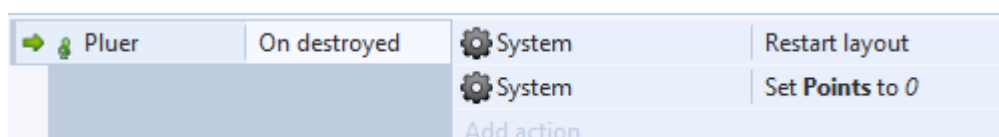


Рисунок 19 – Перезапуск при смерти игрока

Опасность для игрока также представляют ловушки стреляющие стрелами раз в секунду, ловушки и стрелы являются объектами типа sprite, но ловушки имеют в себе дополнительную переменную Enabled означающую её включенность. Стрельба реализована с помощью события, которое создаёт стрелу каждую секунду создаёт стрелу в позиции при каждой ловушке, если она включена. Событие приведено на рисунке 20. Включение происходит с помощью вспомогательного объекта trigger типа sprite, оно реализовано событием, которое изменяет переменную Enabled и анимацию во всех ловушках при пересечении игрока и trigger. Событие приведено на рисунке 21. Стрела также имеет поведение bullet, которое заставляет стрелу передвигаться с

постоянной скоростью. Стрела имеет события, которые при создании стрелы перемещают её на слой объектов, при столкновении стрелы со стеной уничтожают её и при столкновении с игроком уничтожают игрока и стрелу. События приведены на рисунках 22, 23 и 24.

System	Every 1 seconds	System	Create object ➤ ArrowRight on layer 0 at (16, 144)
platformSpa...	Enabled = 1	Add action	
System	Every 1 seconds	System	Create object ➤ ArrowLeft on layer 0 at (464, 304)
platformSpa...	Enabled = 1	Add action	

Рисунок 20 – Стрельба ловушек

Pluer	On collision with	platformSpawnLeft	Set Enabled to 1
	Trigger	platformSpawnLeft	Set animation to "Enabled" (play from beginning)
		platformSpawnRight	Set Enabled to 1
		platformSpawnRight	Set animation to "Enabled" (play from beginning)
		Add action	

Рисунок 21 – Включение ловушек

ArrowLeft	On created	ArrowLeft	Move to layer 4
		Add action	
ArrowRight	On created	ArrowRight	Move to layer 4
		Add action	

Рисунок 22 – Перемещение стрелы при создании

ArrowRight	On collision with	ArrowRight	Destroy
	platformSolid	Add action	
ArrowLeft	On collision with	ArrowLeft	Destroy
	platformSolid	Add action	

Рисунок 23 – Соприкосновение стрелы со стеной

ArrowRight	On collision with	Pluer	Destroy
	Pluer	ArrowRight	Destroy
		Add action	
ArrowLeft	On collision with	Pluer	Destroy
	Pluer	ArrowLeft	Destroy

Рисунок 24 – Соприкосновение стрелы с игроком

Последней опасностью для игрока является передвигающийся противник. Он представляет собой объект типа `sprite` с поведением `platform` и переменными направления и включенности. Перемещение реализовано с помощью события, которое производит перемещение влево или вправо в зависимости от значения переменной направления, потом происходит проверка на то, что противник не будет идти в стену или упадёт с платформы, если проверка неудачная то противник будет развёрнут в обратную сторону. Противник будет включён при первом попадании на экран, и при столкновении с игроком, игрок будет уничтожен. Все события приведены на изображении 25.

enemySimple	enabled = 1	Add action
enemySi...	direction = 0	enemySimple Simulate Platform pressing Left
		Add action
enemySi...	Is overlapping platformSolid at offset (-30, 0)	enemySimple Set direction to 1
		enemySimple Set Mirrored
		Add action
enemySi...	Is overlapping platformSolid at offset (-60, 1)	enemySimple Set direction to 1
		enemySimple Set Mirrored
		Add action
enemySi...	direction = 1	enemySimple Simulate Platform pressing Right
		Add action
enemySi...	Is overlapping platformSolid at offset (30, 0)	enemySimple Set direction to 0
		enemySimple Set Not mirrored
		Add action
enemySi...	Is overlapping platformSolid at offset (60, 1)	enemySimple Set direction to 0
		enemySimple Set Not mirrored
		Add action
enemySimple	Is on-screen	enemySimple Set enabled to 1
		Add action
enemySimple	On collision with Pluer	Pluer Destroy
		Add action

Рисунок 25 – События противника

Концом уровня считается момент когда игрок достигает выхода. При

соприкосновении игрока с выходом вызывается событие меняющее текущий макет и лист событий. При переключении устанавливается значение, равное количеству собранных монет, в текстовый объект. События выполняющие эти действия представлены на рисунках 26 и 27. Вид финального экрана представлен на рисунке 28

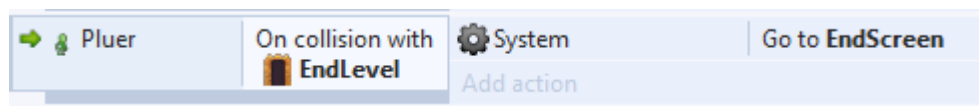


Рисунок 26 – Выход с уровня

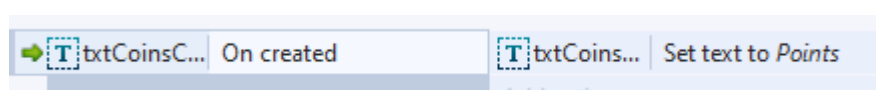


Рисунок 27 – Установка количества собранных монет



Рисунок 28 – Финальный экран

3.2 Реализация на Unity

При первом запуске открывается стартовый экран идентичный по виду реализованному на Construct. Стартовый экран изображён на рисунке 11. Для его изображения используется объект, который находится на объекте со свойством grid, со свойствами tilemap и tilemap renderer для отображения фона стартового экрана, и объект, который находится на объекте с компонентами canvas, canvas scaler и graphic raycaster, с компонентами canvas renderer и text для отображения текста с управлением. Для того чтобы убрать стартовый экран используется скрипт находящийся в объекте P1uer. В скрипте находятся переменные являющиеся объектами отвечающими за стартовый экран и переменная означающая начальный ли экран сейчас. При выполнении скрипта во время нажатия любой кнопки объекты отвечающие за стартовый экран становятся неактивными, тем самым исчезая с поля зрения игрока, и переменная начального экрана изменяет значение на противоположное. Часть этого скрипта выглядит следующий образом:

```
1  public class NewBehaviourScript : MonoBehaviour
2  {
3      ...
4      private bool started = true;
5      ...
6      // Start is called before the first frame update
7      void Start()
8      {
9          ...
10     }
11
12     // Update is called once per frame
13     void Update()
14     {
15         ...
16         if (started)
17         {
18             if (Input.anyKeyDown)
19             {
20                 started = false;
21                 uistart.SetActive(false);
22                 startScreen.SetActive(false);
23             }
24         }
25     }
26 }
```



```

24         }
25         ...
26     }
27 }

```

Игрок управляет объектом `Player` со свойствами `Box Collider 2D` и `Circle Collider 2D` для определения границ взаимодействия объекта, `Rigidbody 2D` для определения физических свойств объекта, `Sprite Renderer` и `Animator` для отображения объекта в виде статичного и динамичного изображения, последним свойством является скрипт отвечающий за управление пользователем персонажа.

Основной частью скрипта является перемещение персонажа при нажатии клавиши. Для этого в скрипте объявлены переменные `JumpForce`, `MovementSmoothing`, `runSpeed` являющиеся настраиваемыми напрямую из движка и отвечающие за силу прыжка, сглаживание передвижения и скорость бега. Переменные `FloorCheckRadius`, `FloorCheck`, `OnFloor`, `jump`, `diRight`, `velocity`, `horMove`, `dead` отвечающие за радиус проверки стоит ли персонаж, точка относительно которой проходит проверка, стоит ли персонаж на земле, в воздухе ли он, направление по горизонтали, ускорение, скорость перемещения по горизонтали, жив ли персонаж. Эти переменные изменяются программно и не могут быть изменены вне редактора кода. Функция `Start` определяет компонент `Rigidbody2D` принадлежащий игроку, потом если персонаж жив в функции `Update` происходит проверка на нажатие клавиш, в функции `FixedUpdate` происходит реакция на нажатие клавиш вызывая функции `floorCheck`, отвечающую за проверку на земле ли игрок, `Flip`, переворачивает систему координат игрока по горизонтали, и `Move`, отвечающую за изменение текущей скорости игрока. Код отвечающий за перемещение персонажа представлен ниже:

```

1  public class NewBehaviourScript : MonoBehaviour
2  {
3      [SerializeField] private float JumpForce = 400f;
4      [Range(0, 0.3f)] [SerializeField] private float MovementSmoothing = 0.05f;
5      [SerializeField] private LayerMask Ground;
6      [SerializeField] private Transform FloorCheck;
7      [SerializeField] private float runSpeed = 40f;
8      private bool dead = false;
9      const float FloorCheckRadius = 0.2f;

```

```

10     private bool OnFloor;
11     private bool jump = false;
12     private bool doubleJump = true;
13     private Rigidbody2D myRigidbody2D;
14     private bool diRight = true;
15     private Vector3 velocity = Vector3.zero;
16     private float horMove = 0f;
17     ...
18     // Start is called before the first frame update
19     void Start()
20     {
21         myRigidbody2D = GetComponent<Rigidbody2D>();
22         ...
23     }
24
25     // Update is called once per frame
26     void Update()
27     {
28         ...
29         if (!dead)
30         {
31             horMove = Input.GetAxis("Horizontal") * runSpeed;
32             if (Input.GetButtonDown("Jump"))
33             {
34                 ...
35                 jump = true;
36             }
37             if (Input.GetButtonDown("Cancel"))
38             {
39                 Application.Quit();
40             }
41         }
42     }
43     private void FixedUpdate()
44     {
45         if (!dead)
46         {
47             floorCheck();
48             Move(horMove * Time.fixedDeltaTime, jump);
49             jump = false;
50             floorCheck();

```

```

51         ...
52     }
53 }
54
55 private void floorCheck()
56 {
57     bool onfloor = OnFloor;
58     OnFloor = false;
59
60     Collider2D[] colliders = Physics2D.OverlapCircleAll(FloorCheck.position,
61         FloorCheckRadius, Ground);
62     for (int i = 0; i < colliders.Length; i++)
63     {
64         if (colliders[i].gameObject != gameObject)
65         {
66             OnFloor = true;
67             doubleJump = true;
68         }
69     }
70 }
71 private void Move(float move, bool jump)
72 {
73     Vector3 targetVelocity = new Vector2(move * 10f, myRigidbody2D.velocity.y);
74     myRigidbody2D.velocity = Vector3.SmoothDamp(myRigidbody2D.velocity,
75         targetVelocity, ref velocity, MovementSmoothing);
76
77     if (move > 0 && !diRight)
78     {
79         Flip();
80     }
81     else if (move < 0 && diRight)
82     {
83         Flip();
84     }
85     if ((OnFloor && jump) || (doubleJump && curState == State.jump && jump)
86         || (doubleJump && curState == State.fall && jump))
87     {
88         if (!OnFloor)
89         {
90             doubleJump = false;
91         }

```

```

92         OnFloor = false;
93         if (myRigidbody2D.velocity.y < 0.1f) myRigidbody2D.velocity =
94             new Vector2(myRigidbody2D.velocity.x, 0);
95         myRigidbody2D.velocity = new Vector2(myRigidbody2D.velocity.x
96             , JumpForce);
97         jump = false;
98     }
99 }
100 }
101

```

Для определения какой вид анимации должен быть проигран в данный момент определяется с помощью эnumератора State, который включает в себя все возможные состояния, переменной curState, указывающей на текущее состояние, и компонента Animator, который позволяет изменять порядок анимации. Функция Start определяет компонент Animator, принадлежащий игроку, функция Update изменяет положение на прыжок при нажатии клавиши отвечающей за прыжок, функция FixedUpdate вызывает функцию StateChanger, которая отвечает за изменение переменной curState, и изменяет значение состояния внутри компонента Animator. Код отвечающий за изменение анимации представлен ниже:

```

1  public class PloverMovement : MonoBehaviour
2  {
3      ...
4      private Animator anime;
5      private enum State { idle, run, jump, fall, deth, spawn, end }
6      private State curState = State.idle;
7      ...
8      void Start()
9      {
10         ...
11         anime = GetComponent<Animator>();
12         ...
13     }
14
15     void Update()
16     {
17         ...
18         if (!dead)

```

```

19         {
20             ...
21             if (Input.GetButtonDown("Jump"))
22             {
23                 curState = State.jump;
24                 ...
25             }
26             ...
27         }
28     }
29
30     private void FixedUpdate()
31     {
32         if (!dead)
33         {
34             ...
35             StateChanger();
36             anime.SetInteger("AnimeState", (int)curState);
37         }
38     }
39     private void StateChanger()
40     {
41         switch (curState)
42         {
43             case State.jump:
44             {
45                 if (myRigidbody2D.velocity.y < -0.001f) curState = State.fall;
46             }
47             break;
48             case State.fall:
49             {
50                 if (OnFloor) curState = State.idle;
51             }
52             break;
53             default:
54             {
55                 if (Mathf.Abs(myRigidbody2D.velocity.x) > 1f)
56                 {
57                     curState = State.run;
58                 }
59                 else

```

```

60         {
61             curState = State.idle;
62         }
63         if (myRigidbody2D.velocity.y < -0.001f && !OnFloor)
64             curState = State.fall;
65     }
66     break;
67 }
68 }
69 }

```

Шипы, огонь, стрелы и противник являются сущностями с тэгом DETH. С этим тэгом персонаж может понять с как взаимодействовать при соприкосновении или прохождении сквозь объект с тэгом. При тэге DETH игрок будет считаться погибшим, для этого используются переменные dead и curState, для определения смерти и возрождения персонажа. Функция Start определяет компонент Animator, принадлежащий игроку, функция OnCollisionEnter2D происходит при соприкосновении, OnTriggerEnter2D происходит при прохождении, обе функции изменяют состояние и анимацию.

```

1  public class PluerMovement : MonoBehaviour
2  {
3      ...
4      private bool dead = false;
5      private State curState = State.idle;
6      ...
7      private void OnCollisionEnter2D(Collision2D collision)
8      {
9          if (!dead)
10             switch (collision.gameObject.tag)
11             {
12                 case "DETH":
13                     {
14                         dead = true;
15                         curState = State.deth;
16                         myRigidbody2D.velocity = new Vector3(0, 0, 0);
17                         anime.SetInteger("AnimeState", (int)curState);
18                     }break;
19                 default:
20                     {

```

```

21         }break;
22     }
23 }
24
25 private void OnTriggerEnter2D(Collider2D collision)
26 {
27     if (!dead)
28         switch (collision.gameObject.tag)
29         {
30             case "DETH":
31             {
32                 dead = true;
33                 curState = State.deth;
34                 myRigidbody2D.velocity = new Vector3(0, 0, 0);
35                 anime.SetInteger("AnimeState", (int)curState);
36             }break;
37             ...
38             default:
39             {
40                 }break;
41             }
42     }
43     ...
44 }

```

За воскрешение игрока отвечают функции `SpawnStart` и `SpawnEnd`. В этих функциях используются компонент `Animator` для изменения анимации, `disabler` для выключения ловушек, `transform` для изменении позиции игрока, `Camera` для изменения камеры. `SpawnStart` вызывается при конце анимации смерти игрока и изменяет положение игрока в стартовую точку с нулевой скоростью, в эту же точку перемещается камера, и изменяется анимация на воскрешение. `SpawnEnd` вызывается после анимации воскрешения и снова включает управление игроку.

```

1 public class PluerMovement : MonoBehaviour
2 {
3     ...
4     private void SpawnStart()
5     {
6         curState = State.spawn;

```

```

7      Camera.position = new Vector3(13.63f, -0.4199999f, -10f);
8      transform.position = SpawnPoint.position;
9      myRigidbody2D.velocity = Vector3.zero;
10     Animator disabler = LeftShooter.GetComponent<Animator>();
11     disabler.SetBool("Enabled", false);
12     disabler = RightShooter.GetComponent<Animator>();
13     disabler.SetBool("Enabled", false);
14     anime.SetInteger("AnimeState", (int)curState);
15 }
16
17 private void SpawnEnd()
18 {
19     dead = false;
20     Camera.position = new Vector3(13.63f, -0.4199999f, -10f);
21     curState = State.idle;
22     anime.SetInteger("AnimeState", (int)curState);
23 }
24 }

```

Стрельба происходит в сущности Shooter, включающей в себе скрипт с функцией включения и выстрела. Функция включения активируется при прохождении игроком положения триггера, функция стрельбы же активируется с помощью события вызываемого в конце включенной анимации и создающих стрелы в точках создания. Код этих функций представлен ниже:

```

1     private void Shoot()
2     {
3         Instantiate(Arrow, SpawnPoint.position, SpawnPoint.rotation);
4     }
5
6     private void OnTriggerEnter2D(Collider2D collision)
7     {
8         if (collision.gameObject.tag == "Player")
9         {
10             Enabled = true;
11             Anime.SetBool("Enabled", Enabled);
12         }
13     }

```

При создании стрел в зависимости от переменной isRight устанавливается скорость и направление в которой летит стрела. Стрелы используют

скрипты для изменения состояния на сломанную, и уничтожения стрелы. При прохождении сквозь стену или игрока функция OnTriggerEnter2D стрела становится сломанной и начинает падать вниз. Функция EndArrow вызывается после прохождения 0.05 секунд после вызова OnTriggerEnter2D. Код этих функций представлен ниже:

```
1
2     void Start()
3     {
4         anime = GetComponent<Animator>();
5         myrb = GetComponent<Rigidbody2D>();
6         if (isRight) myrb.velocity = new Vector2(speed, 0);
7         else myrb.velocity = new Vector2(-speed, 0);
8     }
9
10    private void OnTriggerEnter2D(Collider2D collision)
11    {
12        switch (collision.gameObject.tag)
13        {
14            case "Ground":
15                {
16                    myrb.velocity = new Vector2(0, -0.5f);
17                    myrb.gravityScale = 1;
18                    anime.SetBool("Broken", true);
19                }
20                break;
21            case "Player":
22                {
23                    myrb.velocity = new Vector2(0, -0.5f);
24                    myrb.gravityScale = 1;
25                    anime.SetBool("Broken", true);
26                }
27                break;
28        }
29    }
30
31    private void EndArrow()
32    {
33        Destroy(gameObject);
34    }
```

Подбор монет осуществляется с помощью скрипта в сущности Player и

скрипта в сущности Coin. Скрипт в Player изменяет количество собранных монет и состояние монеты. Скрипт монет имеет только функцию EndSparkle, которая уничтожает монету при конце анимации сбора. Код функций представлен ниже:

```
1 private void OnTriggerEnter2D(Collider2D collision)
2 {
3     if (!dead)
4         switch (collision.gameObject.tag)
5         {
6             ...
7             case "Coin":
8             {
9                 counter.text = coinCount.ToString();
10                Animator animeCoin =
11                    collision.gameObject.GetComponent<Animator>();
12                if(!animeCoin.GetBool("Collected")) coinCount++;
13                animeCoin.SetBool("Collected", true);
14            }break;
15            ...
16        }
17    }
18
19    void EndSparkle()
20    {
21        Destroy(gameObject);
22    }
```

Противник использует собственный скрипт для определения перемещения. Он включает в себя переменные и компоненты FloorPoint, isRight, speed, checkRadius, Ground, myrb отвечающие за тоже что и в Pluer. Start определяет компонент myrb и первоначальное ускорение, Update изменяет ускорение, FixedUpdate определяет направление. Код представлен ниже:

```
1 public class EnemyMovement : MonoBehaviour
2 {
3     [SerializeField] private Transform FloorPoint;
4     [SerializeField] private bool isRight;
5     [SerializeField] private float speed;
6     [SerializeField] private float checkRadius;
7     [SerializeField] private LayerMask Ground;
```

```

8     private Rigidbody2D myrb;
9
10    // Start is called before the first frame update
11    void Start()
12    {
13        myrb = GetComponent<Rigidbody2D>();
14        if (isRight) myrb.velocity = new Vector2(speed, 0);
15        else myrb.velocity = new Vector2(-speed, 0);
16    }
17
18    // Update is called once per frame
19    void Update()
20    {
21        if (isRight) myrb.velocity = new Vector2(speed, 0);
22        else myrb.velocity = new Vector2(-speed, 0);
23    }
24
25    private void FixedUpdate()
26    {
27        if(isRight)
28        {
29            isRight = false;
30
31            Collider2D[] colliders = Physics2D.OverlapCircleAll(FloorPoint.position,
32                checkRadius, Ground);
33            for (int i = 0; i < colliders.Length; i++)
34            {
35                if (colliders[i].gameObject != gameObject)
36                {
37                    isRight = true;
38                }
39            }
40            if (!isRight)
41            {
42                transform.Rotate(0f, 180f, 0f);
43            }
44        }
45        else
46        {
47            isRight = true;
48

```

```

49         Collider2D[] colliders = Physics2D.OverlapCircleAll(FloorPoint.position,
50             checkRadius, Ground);
51         for (int i = 0; i < colliders.Length; i++)
52         {
53             if (colliders[i].gameObject != gameObject)
54             {
55                 isRight = false;
56             }
57         }
58         if(isRight)
59         {
60             transform.Rotate(0f, 180f, 0f);
61         }
62     }
63
64 }
65 }

```

При прохождении игроком выхода функция OnTriggerEnter2D показывает финальный экран идентичный реализованному в Construct. Код представлен ниже:

```

1 private void OnTriggerEnter2D(Collider2D collision)
2     {
3         if (!dead)
4             switch (collision.gameObject.tag)
5             {
6                 ...
7                 case "Exit":
8                 {
9                     endgame = true;
10                    uiend.SetActive(true);
11                    endScreen.SetActive(true);
12                    curState = State.idle;
13                    myRigidbody2D.velocity = Vector3.zero;
14                    transform.position = new Vector3(39.02f, 23.25f, 0f);
15                    anime.SetInteger("AnimeState", (int)curState);
16                    dead = true;
17                }break;
18                ...
19            }
20     }

```

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- 1 *Roberto, D.* HTML5 Game Development from the Ground Up with Construct 2 / D. Roberto. — New York: CRC Press, 2014.
- 2 *Stemkoski, L.* Game Development with Construct 2: From Design to Realization / L. Stemkoski. — New York: Apress, 2017.
- 3 Официальный сайт Construct 2 [Электронный ресурс]. — URL: <https://www.scirra.com/store/construct-2> (Дата обращения 25.02.2020). Загл. экр. Англ. рус.
- 4 *Bond, J. G.* Introduction to Game Design, Prototyping, and Development: From Concept to Playable Game with Unity and C# / J. G. Bond. — Boston: Addison-Wesley, 2014.
- 5 *Halpern, J.* Developing 2D Games with Unity : Independent Game Programming with C# / J. Halpern. — New York: Apress, 2018.
- 6 C# docs [Электронный ресурс]. — URL: <https://docs.microsoft.com/en-us/dotnet/csharp/> (Дата обращения 14.04.2020). Загл. экр. Англ. рус.
- 7 *Ferrone, H.* Learning C# by Developing Games with Unity 2019: Code in C# and build 3D games with Unity, 4th Edition / H. Ferrone. — Burmingham: Packt, 2019.
- 8 Официальный сайт Unity [Электронный ресурс]. — URL: <https://unity.com> (Дата обращения 11.04.2020). Загл. экр. Англ. рус.
- 9 Unity Game Engine Review [Электронный ресурс]. — URL: <https://www.gamesparks.com/blog/unity-game-engine-review/> (Дата обращения 24.04.2020). Загл. экр. Англ. рус.