**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**МЕХАНИКО-МАТЕМАТИЧЕСКИЙ ФАКУЛЬТЕТ**

**Кафедра веб-технологий и компьютерного моделирования**

**СОЗДАНИЕ 2D-ВИДЕОИГРЫ С ПОМОЩЬЮ ИГРОВОГО ДВИЖКА UNITY**

Курсовая работа

Логвиненко Антона Викторовича

студента 2 курса, специальность 1-31 03 08-01 Математика и информационные технологии.

Научный руководитель:  
кандидат физ.-мат. наук,  
доцент А. И. Кравчук

Минск, 2023

**Оглавление**

[введение 3](#_Toc388294027)

[Глава 1 Общие положения 4](#_Toc388294028)

[1.1 Курсовые работы исследовательского характера 4](#_Toc388294029)

[1.2 Курсовые работы реферативного характера 5](#_Toc388294030)

[Глава 2 Иллюстрации, формулы и приложения 7](#_Toc388294031)

[2.1 Общие требования к оформлению 7](#_Toc388294032)

[2.2 Заголовки структурных частей работы 7](#_Toc388294033)

[2.2.1 Заголовки разделов 7](#_Toc388294034)

[2.2.2 Заголовки подразделов 7](#_Toc388294035)

[2.3 Иллюстрации и таблицы 7](#_Toc388294036)

[2.4 Оформление научно-справочного аппарата 7](#_Toc388294037)

[заключение 7](#_Toc388294038)

[Список использованной литературы 7](#_Toc388294039)

[ПРИЛОЖЕНИЕ А. Код программы 7](#_Toc388294040)

введение

Видеоигры в современности – повсеместно известное и популярное развлечение среди всех возрастных групп населения, хотя особый интерес к ним демонстрируют подростки и взрослые до 27-30 лет. Появившаяся относительно недавно индустрия компьютерных и консольных развлечений является самой прибыльной среди конкурентов, и количество ее потребителей растет ежегодно. С появлением смартфонов каждое такое устройство предоставляет доступ к мобильным играм через приложения Play Market(для Android) и App Store(для техники Apple). Одними из самых популярных игр для телефонов являются аркадные игры, т.е. игры, которые ведутся на счет и потенциально являются бесконечными, из чего следует, что основной целью такой игры является соревнование с другими игроками на больший счет.

При создании видеоигр в основном используются так называемые игровые движки. Игровой движок – базовое программное обеспечение компьютерной игры, пригодное для повторного использования и расширения. Базовые вещи, необходимые при создании каждой игры, уже встроены в них, и разработчику не нужно заново прописывать физику тел и их отображение для каждой игры. Самыми популярными платформами для создания игр на данный момент являются Unity, Unreal Engine и Godot.

Unity (в переводе с англ. — «единство») — кроссплатформенная среда разработки компьютерных игр, разработанная американской компанией Unity Technologies. Она снабжена движками для расчета физики объектов как в 3D, так и в 2D проектах. Редактор Unity предоставляет возможность добавлять в сцены – отдельные файлы, содержащие информацию об игровых объектах внутри отдельной игровой среды – как объекты, обладающие визуальным представлением в мире игры, так и пустые объекты. Каждый объект является совокупностью компонентов, с помощью которых происходит взаимодействие скриптов с ним. Для того, чтобы, например, применить силу гравитации на объект, необходимо добавить соответствующий компонент, который отвечает за физику объекта.

Для написания скриптов используется язык C# – язык с С-подобным синтаксисом, наиболее близкий к языкам С++ и Java. Он является статически типизированным, реализует механизмы и принципы ООП, при этом избавившись от проблематичных моделей своих предшественников, такие как множественное наследование С++. Unity поддерживает и другие языки для своих сценариев, такие как JavaScript(или его специализированная версия UnityScript), но наибольшая поддержка оказывается C#-разработчикам.

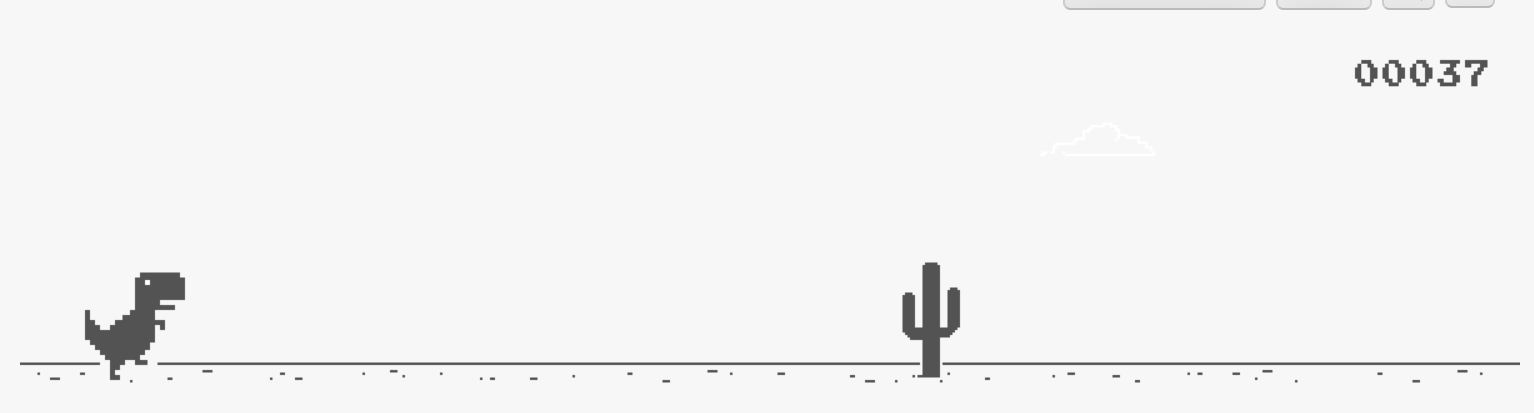
Преимуществами Unity являются наличие визуальной среды разработки, что упрощает прототипирование и тестирование игры; наличие большого количества обучающего контента для ознакомления с игровым движком новичкам; простота интерфейса приложения и существование магазина Unity Asset Store, где разработчики могут делиться различными элементами, используемыми для создания игр, например, моделями, аудио файлами, библиотеками и т.д. Так же Unity версии Personal является бесплатным для разработчиков, которые не используют платформу для коммерческих целей.

Среди недостатков можно отметить ограничение визуального редактора при наличии внутри проекта комплексных вложенных структур объектов, при которых затрудняется взаимодействие со сценой; самостоятельная настройка работы с внешними библиотеками.

«Название\_игры» – это 2D-игра, в которой игроку нужно преодолевать препятствия и противников, набирая при этом очки. Персонаж игрока движется с увеличивающейся со временем скоростью вправо, при этом на его пути встречаются ловушки и противники, которые при соприкосновении с игроком наносят ему урон – отнимают одну единицу жизни. Когда количество жизней иссякает, игра заканчивается, и игроку необходимо начинать сначала. Препятствия появляются со случайным промежутком времени и случайным образом, создавая таким образом множество уникальных ситуаций, проверяющие мастерство и реакцию пользователя. Игрок соревнуется с самим собой на установление нового лучшего счета.

Рассмотрим игры-аналоги: браузерная игра Dinosaur Game, мобильная игра Subway Surfer.

Dinosaur Game – встроенная в браузер Google Chrome игра, спрятанная на странице ошибки при отсутствии соединения с интернетом. Игрок берет на себя управление бегущего динозавра с помощью клавиш вверх и вниз, чтобы перепрыгивать или пригибаться перед препятствиями. На экране отображается счетчик очков, которые увеличиваются во время хода игры. Со временем скорость динозавра и частота появления препятствий увеличивается, что делает игру сложнее. Игра заканчивается, если игрок соприкасается с любым из препятствий или игровое время достигает 17 миллионов лет.

Достоинствами данной игры является простое управление, возрастающая сложность и система очков, позволяющая игроку сохранять интерес к игре на продолжительное время с целью увеличения своего лучшего результата, соревнуясь с другими людьми. Среди недостатков выделяется однообразный визуальный стиль, отсутствие разнообразия среди видов препятствий.

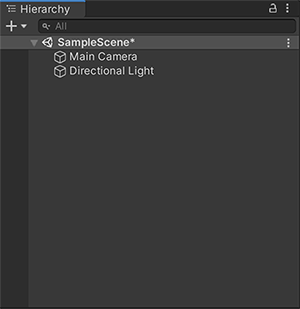
Другим аналогом является мобильная 3D-игра Subway Surfer. Главному персонажу нужно убегать от охранника по железнодорожным путям, уворачиваясь от поездов и минуя другие препятствия, встречающиеся на его пути. Игроку предоставлены три пути передвижения, во время игрового процесса ему будет необходимо своевременно сменять пути движения для предотвращения столкновений, а также использовать прыжки и кувырки для уворота от некоторых препятствий, когда смена дорожки невозможна. Одной из главных игровых механик Subway Surfer является система бонусов, которые игрок подбирает во время своего пути. Они помогают игроку преодолевать препятствия, добавляют мультипликатор счета для улучшения результата, помогают собирать монеты и так далее. Например, ракетный ранец позволяет игроку на некоторое время подняться в воздух, где столкновение с объектами ему не грозит, или специальные сапоги, которые позволяют перепрыгивать через поезда.

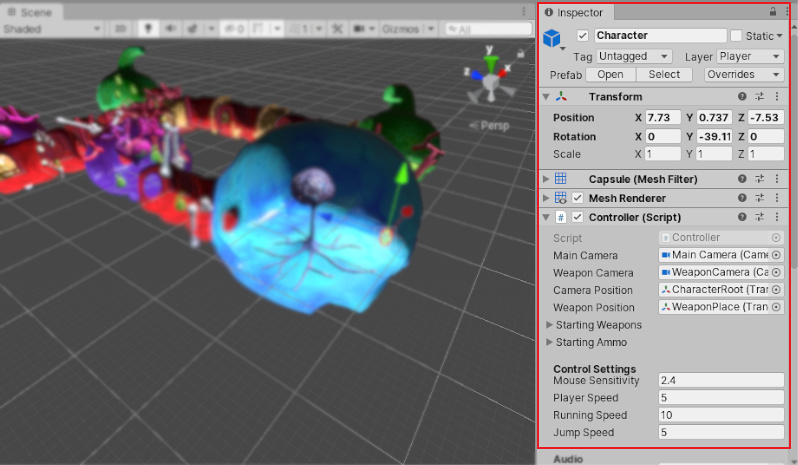
Среди преимуществ игры Subway Surfer имеются привлекающий визуальный стиль, наличие нескольких способов для решения одной и той же ситуации, наличие системы бонусов, помогающих игроку в прохождении. Недостатком же можно назвать то, что бонусы применяются сразу после подъема, хотя нужды в этом в данный момент может и не быть.

# Subway Surfers MOD 3.9.0 - Скачать для Android APK бесплатно

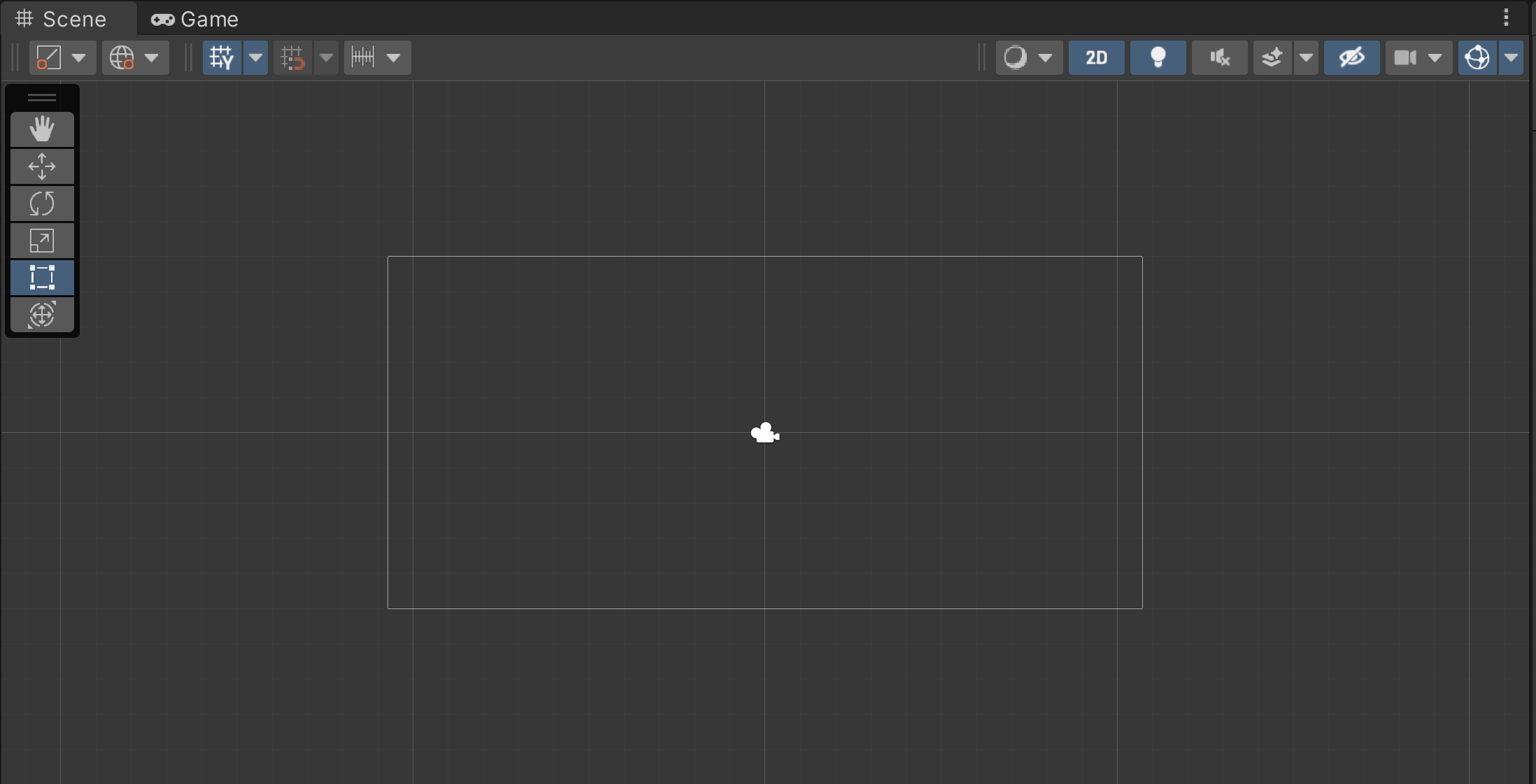
При создании «Название\_игры» были учтены достоинства и недостатки игр-аналогов. Из них были позаимствованы возрастающая сложность, ведение счета очков, простота управления, система бонусов для помощи игроку. При этом будут исправлены упомянутые недочеты: однообразный визуальный стиль, отсутствие возможности отложить применение бонусов для более подходящих ситуаций.

При создании проекта в Unity открывается редактор движка Unity Editor с открытой сценой, где находится единственный объект – камера. Камера служит для задания области игрового мира, которая будет видна на экране монитора. В окне Heirarchy находится иерархия объектов внутри сцены, здесь же можно создавать, выбирать и удалять объекты сцены.

Каждый объект в игре создается с уже добавленным компонентом Transform. Этот компонент используется для хранения позиции, угла поворота, масштаба объекта внутри игрового мира. Положение объектов определяется координатами x,y,z внутри сцены. Каждый Transform может являться потомком другой сущности, и его размеры и положение будут определяться относительно своего родителя. Это и позволяет Unity создавать иерархию объектов, которую мы видим в окне Heirarchy.

Чтобы посмотреть компоненты объекта, нужно выбрать этот объект в окне иерархии или внутри сцены простым нажатием на объект Левой кнопки мыши. Вся информация появится в окне Inspector, где можно добавлять компоненты, удалять и изменять свойства объекта. Также каждый объект имеет тег и слой. Теги нужны для разбиения объектов на группы, присваивая им строку-идентификатор, что упрощает написание скриптов, если нужно применить какое-то действие на несколько объектов. Слои в свою очередь нужны для определения взаимодействий объектов друг с другом внутри сцены, например, скрыть от камеры определенные объекты при отображении.

С помощью окна Project можно взаимодействовать с файлами проекта и библиотеками. Console используется для отображения ошибок при компиляции скриптов, ошибок, появляющихся во время игры, а также для вывода пользовательской информации во время выполнения скриптов.

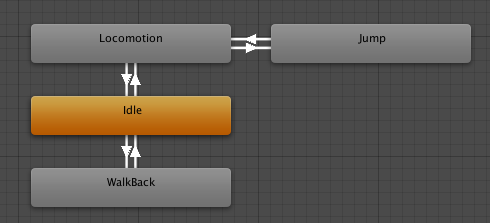
Отображение сцены со всеми объектами, находящимися в ней, находится в окне Scene, где можно манипулировать объектами с помощью инструментов перемещения, поворота, увеличения и уменьшения и т.д. То, как игра будет выглядеть для конечного пользователя, находится в окне Game, где происходит сам игровой процесс при запуске игры.

Игровой движок Unity дает возможность переключаться на разные способы отображения игры: 2D и 3D. В 2D режиме вся графика будет отображаться в виде плоских картинок, именуемых спрайтами, в то время как трехмерные модели требуют 3D-визуализации. Для того, чтобы добавить изображение на определенный объект внутри сцены, необходимо добавить ему компонент Sprite Renderer, и перетаскиванием(или выбором в выпадающем меню) добавить в соответствующее поле нужную картинку.

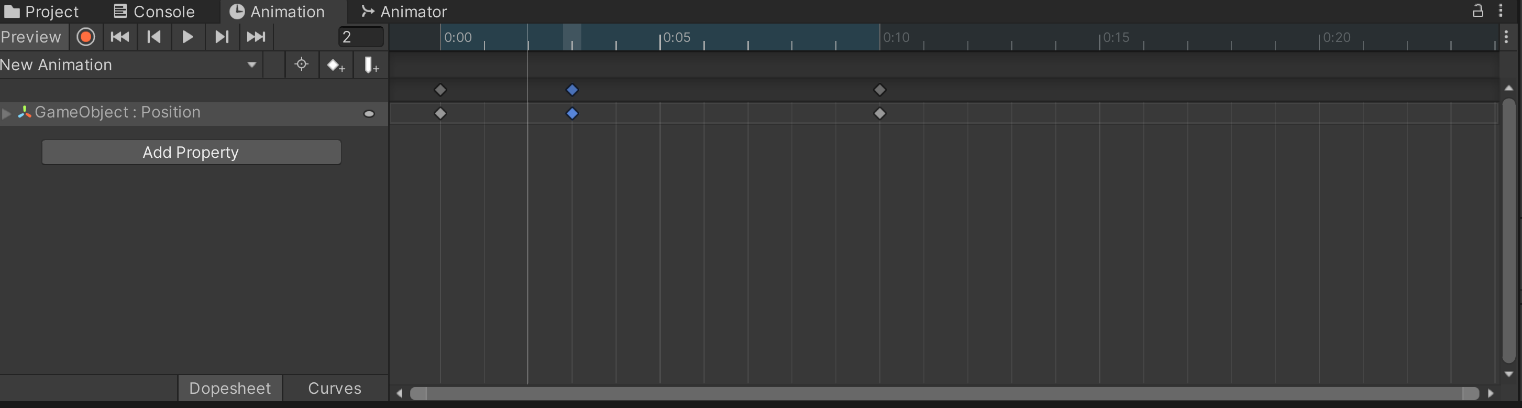
Для работы с кодом проекта подойдет любая среда разработки, однако больше всего инструментов для работы с платформой Unity предоставляют Rider от JetBrains и Visual Studio от Microsoft. При создании данного проекта использовалась бесплатная версия Visual Studio Enterprise с установленным модулем инструментов для Unity-проектов.

Анимация – представление движения статических картинок путем отображения последовательности кадров с высокой частотой. Unity Editor, помимо возможности импортировать записи анимаций сторонних приложений, предоставляет разработчику встроенные инструменты, чтобы анимировать игровые сущности. Для этого в редакторе существует Animator Controller – контроллер анимаций, использующий для управления переходами между анимациями механизм Unity State Machine(машину состояний). Главная идея State Machine заключается в том, что анимируемый объект находится в определенном состоянии в любой момент времени. Такими состояниями могут быть положение покоя, бег, прыжок и другие действия. Чтобы перейти от одного состояния в другое, необходимо указать условие перехода посредством изменения значений переменных состояния. В совокупности наборы состояний, переходы между ними и переменные состояния образуют State Machine.

Чтобы анимировать игровой объект, необходимо добавить компонент Animator, а затем в поле Animator Controller поместить контроллер, который будет управлять анимациями данной сущности. Анимации в Unity могут быть применены для изменения положения, поворота, масштабирования, внешнего вида объекта и переменных скриптов, примененных к объекту.

Для настройки этих состояний и переходов между ними используется окно Animator. Здесь визуализирован граф всех связей внутри контроллера анимаций, а также условия переходов между разными записями, скорость перехода и другое.

Чтобы добавить условие для перехода к новому состоянию, в окне Animator необходимо добавить новый параметр, имеющий один из следующих типов: Int, Float, Bool и Trigger. Параметры типа Int и Float вызывают переход, только если значение параметра стало больше или меньше определенного значения. Условие типа Bool срабатывает при значении равном либо true, либо false, в то время как Trigger позволяет сразу активировать переход при вызове функции SetTrigger внутри скрипта.

Для записи анимации внутри редактора Unity, используется окно Animation. При этом необходимо добавить только ключевые кадры анимации, все остальное будет просчитано и добавлено платформой Unity автоматически. Кадры анимации располагаются на временной шкале, определяющей, в какой момент времени будет отображено новое состояние. Таким образом возможно контролировать протяженность анимации.

Unity предоставляет возможность вызывать функции в любой момент анимации, например, в момент удара мечом. Выделив мышкой место на временной шкале и нажав на кнопку «Add event», создается событие, которое будет вызывать указанные функции, примененных к анимируемому объекту, в момент проигрывания анимации. Функции, которые упомянуты выше, должны быть публичными и принадлежать сценарию того же игрового объекта, чтобы все работало правильно.

Unity Asset Store – это площадка, где разработчики делятся элементами игр, такими как картинки, модели, аудио файлы, анимации и многое другое для использования в других проектах Unity. Unity Editor позволяет импортировать купленные пакеты в проект, добавляя их в файловую систему. После этого содержимым пакета можно воспользоваться внутри редактора.

Для создания уровня был использован инструмент Tilemap, встроенный в Unity Editor. Компонент Tilemap позволяет размещать изображения внутри сетки, которая создается компонентом Grid. Grid разбивает экран на сетку, состоящую из точек и линий, где каждая ячейка имеет одинаковый размер. Обычно ячейки имеют одинаковый размер, при этом обладая различными формами, например, квадраты, шестиугольники или параллелограммы. Картинки, из которых будет состоять Tilemap, помещаются в Tile Pallette – инструмент, схожий по смыслу с палитрой и кисточкой. С помощью кисточки можно выбрать спрайт, находящийся в палитре, и начать размещать картинки внутри сетки. Такой способ разметки упрощает создание уровней и позволяет быстро создавать прототипы при создании 2D-миров внутри игры.

Чтобы движок рассчитывал поведение объектов при столкновении с плитками Tilemap, на карту необходимо добавить компонент Tilemap Collider 2D в сочетании с Composite Collider 2D, который объединит все картинки в одну форму, если спрайты находятся в соседних ячейках сетки.

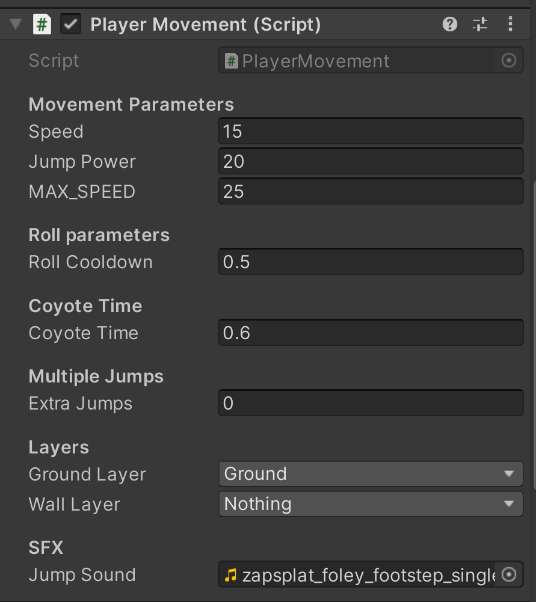
Для расчета поведения объектов при столкновении используется встроенный компонент Unity. Collider2D – это компонент, который позволяет разработчику определить форму объекта для определения столкновения с другими объектами при пересечении данной формы. Дочерний компонент BoxCollider2D позволяет определить форму как прямоугольник. Внутри компонента есть свойства, позволяющие настроить поведения объекта при столкновении: просто проинформировать скрипты про наличие столкновения, позволяя другой сущности пройти насквозь, или просчитать поведение объектов с помощью физических формул, чтобы объекты оттолкнулись друг от друга или остановились после столкновения.

RigidBody2D – компонент, накладывающий контроль физического движка над поведением объекта, то есть к сущности будут применяться силы гравитации, она будет обладать своим весом, скоростью, коэффициентом трения и прочими свойствами, необходимыми для применения силы по направлению к объекту.

Для определения пользовательского поведения объекта или для программирования событий внутри игры используются скрипты(сценарии), которые находятся среди компонентов объектов. По умолчанию Unity Editor создает внутри скрипта класс, наследуемый от MonoBehaviour – базовый класс для всех сценариев Unity, которые впоследствии станут компонентами игровых сущностей. Сценарий в Unity не является традиционной идеей программы, где код выполняется последовательно, пока задача не будет выполнена. Вместо этого Unity передает контроль скрипту поочередно, вызывая определенные функции внутри сценария. После выполнения функции контроль возвращается обратно к Unity. Такие функции известны как функции событий, так как они вызываются при реагировании платформы на некоторые события внутри игры.

Одной из таких функций событий является функция Update. Игра похожа на анимацию в том смысле, что новые положения и поведения игровых сущностей отображаются в виде кадров. Каждое изменение рассчитывается непосредственно перед новым кадром и для этой цели используется функция Update, которая вызывается всякий раз перед отрисовкой кадра и перед расчетом анимаций.

Если необходимо вызвать код инициализации перед началом обновлений, то применяются методы Start и Awake, которые срабатывают при включении скрипта и при создании объекта соответственно.

Поля класса сценария, помимо встроенных в язык программирования C# модификаторов доступа, могут принимать модификатор типа [SerializeField]. В таком случае поле называется сериализуемым. Этот вид доступа, а также модификатор public позволяют редактировать значение поля напрямую из Unity Editor в компоненте соответствующего сценария, но в отличие от последнего, сериализуемое поле класса будет недоступно для изменения другим сценариям проекта.

Для того, чтобы добавить аудио сопровождение внутри игры, Unity предоставляет компонент AudioListener(слушатель аудио), который играет роль микрофона в 3D пространстве. Этот компонент размещается на объекте, который будет записывать звуки вокруг себя и проигрывать их игроку через аудио устройства компьютера. В сцене может быть только один слушатель и его обычно размещают на главной камере, что является выбором по умолчанию.

Для воспроизведения аудиодорожки на источнике звука должен находится компонент AudioSource(источник звука). В поле AudioClip помещают файл звука, с помощью других свойств можно настроить громкость, зацикленность звука и другое.

Движение персонажа происходит следующим образом: при создании объекта игрока в сцене внутри функции Awake скрипта PlayerMovement(движение игрока) берутся ссылки на другие компоненты персонажа, такие как RigidBody2D для симуляции движения, Animator для проигрывания анимаций бега, кувырка и прыжка и BoxCollider2D для определения положения игрока относительно пола.

body = GetComponent<Rigidbody2D>();

anim=GetComponent<Animator>();

boxCollider= GetComponent<BoxCollider2D>();

Скорость тела внутри компонента RigidBody2D задается двумерным вектором с координатами x и y. В функции событий Update на тело накладывается скорость, заданная заранее, что приводит тело в движение в направлении оси X:

body.velocity = new Vector2(speed, body.velocity.y);

При этом главная камера сцены движется вместе с главным персонажем, чтобы игрок мог отслеживать его перемещение.

Чтобы сложность игры возрастала с течением времени, необходимо увеличивать скорость, с какой игрок движется во время игрового процесса. Чтобы этого достичь, скорость персонажа будет увеличиваться на одну десятую от времени, прошедшего с прошлого кадра, до тех пор, пока она не достигнет максимального заданного значения:

if (speed<MAX\_SPEED)

{

speed += Time.deltaTime/10;

}

Для определения положения персонажа относительно пола, создается небольшой виртуальный прямоугольник ниже BoxCollider2D, установленном на игроке. Если этот прямоугольник регистрирует пересечение с объектом, у которого значение слоя является равным groundLayer(слой земли), то персонаж находится на земле.

private bool isGrounded()

{

RaycastHit2D raycastHit = Physics2D.BoxCast(boxCollider.bounds.center,boxCollider.bounds.size,0,Vector2.down,0.1f,groundLayer);

return raycastHit.collider != null;

}

Прыжок, в отличии от движения персонажа, управляется игроком посредством нажатия кнопки Пробел на клавиатуре. Если в этот момент управляемый объект находится на земле, скорость тела по координатам y становится равным заданному значению силы прыжка, при этом высота прыжка будет зависеть от продолжительности нажатия Пробела игроком:

if(Input.GetKeyUp(KeyCode.Space) && body.velocity.y > 0)

{

body.velocity = new Vector2(body.velocity.x, body.velocity.y / 2);

}

Для персонажа предусмотрена настройка количества прыжков, которые тот может совершить подряд без касания земли. Если это число больше двух, то если Пробел был нажат, пока игрок находился в воздухе, то будет сделан еще один прыжок и количество возможных прыжков уменьшается на один.

Для совершения кувырка, применяемый для уворота от стрел, летящих слишком низко, необходимо нажать кнопку C клавиатуры. При этом размер BoxCollider2D игрока уменьшится вдвое по оси Y, а также запуститься соответствующая запись анимации кувырка. На последнем кадре анимации вызывается функция, возвращающая размеры в исходное состояние.

Для бесконечного бега персонажа в направлении оси X требуется наличие бесконечно длинного пола, по которому игрок передвигается. Роль земли в проекте отведена Tilemap с повторяющимися изображением почвы. Создать карту огромной длины было бы непроизводительно и неэффективно, потому что отображаться в это время на экране будет мизерная часть от всей сетки.

Идея для решения данной проблемы заключена в создании новой почвы на некотором расстоянии перед игроком при каждой смене кадра и удалении ненужных элементов земли, вышедший за пределы камеры. Для этого берется положение правой границы видимости главной камеры, следящей за движением игрока. Затем прибавляется некоторое расстояние по координате x к координатам границы, чтобы появление новых элементов почвы происходило за кадром. И, наконец, переведя полученный двумерный вектор в координаты сетки, устанавливается новый спрайт земли. Все спрайты, вышедшие за левую границу области видимости, впоследствии удаляются, установив нулевое значение на месте ненужных спрайтов.

//getting position of where new tile should be planted and position of the one to be deleted

int cameraLeftPosition = (int)Camera.main.ViewportToWorldPoint(Vector3Int.left).x - 3;

cameraRightPosition = (int)Camera.main.ViewportToWorldPoint(Vector3Int.right).x + 2;

//deleting tiles out of view to the left of the camera

for (int i = -2; i < cameraTopPosition; i++)

{

tilemap.SetTile(tilemap.WorldToCell(new Vector3Int(cameraLeftPosition, i, 0)), null);

}

tilemap.SetTile(tilemap.WorldToCell(new Vector3Int(cameraRightPosition, -2, 0)), tile);

Другим вариантом при создании сценария бесконечной генерации земли был основан на случайности, то есть для каждого блока делался выбор: будет ли на его месте установлена земля или останется пустое место – «пропасть». Однако реализация такого метода на практике оказалась чрезмерно сложной и неудобной, потому что провалы в земле, которые игроку необходимо перепрыгнуть, по смыслу мало отличались от ловушки Spikes. Поэтому от этого варианта пришлось отказаться в пользу более контролируемого метода.

Базовый класс Health(здоровье) ответственен за систему здоровья внутри игры, а сценарии HealthPlayer(здоровье игрока) и HealthEnemy(здоровье противника) наследуются от этого класса для определения того, как будет работать здоровье непосредственно для игрока и его противников. Изначальное число жизней, которые определяют сколько ударов может выдержать персонаж перед тем как он будет деактивирован, задается предварительно либо внутри кода, либо посредством сериализации внутри Unity Editor. Базовый класс объявляет, но не определяет метод TakeDamage(получить урон), так как реакция на получение урона у игрока и противника будет разной: например, при смерти игрока должны запускаться сценарии конца игры, в то время как смерть противника приводит к падению бонуса, который поможет при прохождении, и начислению очков игроку. Также реализован метод AddHealth(добавить здоровье), который восстанавливает утраченные очки здоровья на заданное значение.

Чтобы помочь игроку быстрее восстановить фокус к игре после получения урона, была создана механика неуязвимости на короткое время после удара. Это значит, что игрок не будет получать урон от последующих попаданий игрока в ловушки и ударов противников на заданный промежуток времени. Неуязвимость демонстрируется несколькими «вспышками» – изображение героя становится красным, после чего возвращает исходный цвет. Реализовано это с помощью сопрограммы, также называемой Coroutine.

Coroutine – это объект Unity, который позволяет начать параллельное выполнение действий. Используются сопрограммы в Unity в тех случаях, когда код необходимо выполнить в течение нескольких кадров, а не одного. Когда внутри сценария запускается функция StartCoroutine, создается экземпляр сопрограммы, который выполняет некоторые действия до тех пор, пока не доходит до строки yield return, в которой возвращается IEnumerator. В основном, возвращается объект, который отсчитывает время ожидания для выполнения остальной части кода после операторов yield return, например:

yield return new WaitForSeconds(5);

Затем сценарий выполняет остальной код после StartCoroutine. Когда проходит время ожидания для выполнения отложенной части программы, управление возвращается к следующей строке после оператора yield. На листинге продемонстрировано, как это было реализовано для создания неуязвимости в игре:

StartCoroutine(Invulnerability());

protected IEnumerator Invulnerability()

{

invulnerability= true;

Physics2D.IgnoreLayerCollision(playerLayer, enemyLayer, true);

for(int i = 0;i < numberOfFlashes;i++)

{

spriteRend.color = new Color(1, 0, 0, 0.5f);

yield return new WaitForSeconds(iframeDuration/(numberOfFlashes \* 2));

spriteRend.color = Color.white;

yield return new WaitForSeconds(iframeDuration / (numberOfFlashes \* 2));

}

Physics2D.IgnoreLayerCollision(playerLayer, enemyLayer, false);

invulnerability = false;

}

Функция Invulnerability начинается с установления флага invulnerability на true, что предотвращает снятие очков здоровья игрока при выполнении функции TakeDamage. Затем все столкновения объекта игрока и объектов, у которых значения слоя равно enemyLayer(слой противников), перестают регистрироваться. Далее запускается цикл, где в зависимости от количества «вспышек» изменяется цвет изображения игрока с помощью свойства color компонента SpriteRenderer. Между сменами цветов устанавливается время ожидания, и после выхода из цикла регистрация столкновений становится исходной и неуязвимость заканчивается.

Чтобы справляться с вражескими рыцарями, попадающимися у игрока на пути, персонаж имеет возможность атаковать противника обратно. Атака происходит при нажатии Левой кнопки мыши, и если после прошлой атаки прошло заданное количество времени, то начинает проигрываться запись анимации атаки игрока. В середине кадра в момент удара мечом происходит проверка, находится ли противник в пределах некоторой зоны перед игроком. Для этого перед игроком создается виртуальный прямоугольник, длину и отдаленность от игрока которого определяется разработчиком предварительно с помощью сериализованных полей range и colliderDistance внутри Unity Editor.

private bool EnemyInSight()

{

RaycastHit2D hit = Physics2D.BoxCast(boxCollider.bounds.center + transform.right \* range\*transform.localScale.x \* colliderDistance,

new Vector3(boxCollider.bounds.size.x \* range,boxCollider.bounds.size.y, boxCollider.bounds.size.z)

, 0,Vector2.left,0,EnemyLayer);

if (hit.collider != null){

enemyHealth = hit.transform.GetComponent<Health>();

}

return hit.collider!=null;

}

 Если во время удара мечом в этом прямоугольнике находился противник, то мы получаем доступ к его здоровью и наносим ему урон. Во время создания прототипов и итерирования механики атаки было необходимо было визуальное представление зоны атаки игрока, поэтому была применена функция OnDrawGizmos, которая выполнила это требование. Gizmos – это инструмент Unity Editor, предоставляющий возможность разработчику визуализировать компоненты объекта, такие как BoxCollider2D, во время запуска игры. Это помогло при определении оптимального расстояния, в пределах которых противник будет получать урон. На изображении эта хона обозначена красным цветом.

private void OnDrawGizmos()

{

Gizmos.color = Color.red;

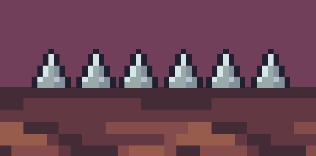
Gizmos.DrawWireCube(boxCollider.bounds.center + transform.right \* range \* transform.localScale.x\* colliderDistance,

new Vector3(boxCollider.bounds.size.x \* range, boxCollider.bounds.size.y, boxCollider.bounds.size.z)

);

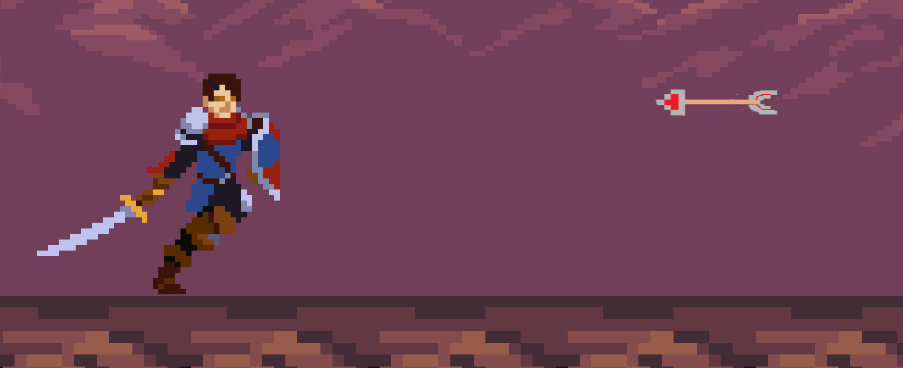
}

Основными препятствиями в «Название\_игры» являются ловушки и противники. Разные виды преград должны требовать от игрока разного поведения, что делает способы взаимодействия игрока с игровым миром более разнообразным.

Ловушка Spikes(шипы) – это препятствие, которое располагается непосредственно на земле. Шипы необходимо перепрыгнуть, в противном случае игрок получает урон при касании к любому месту ловушки. При этом игрок может беспрепятственно проходить через ловушку, не прекращая движения.

Нанесение урона игроку посредством препятствий реализовано с помощью сценария EnemyDamage(урон противника). Базовый класс сценариев MonoBehaviour объявляет метод OnTriggerEnter2D, который вызывается при регистрации любого столкновения объекта с другими сущностями, передавая в качестве аргумента информацию о столкновении. При этом у одного из объектов столкновения должно быть отмечено свойство IsTrigger(является переключателем) внутри компонента BoxCollider2D, которое не будет препятствовать прохождению других тел через объект. Регистрация столкновений будет происходить как и раньше.

Внутри функции OnTriggerEnter2D, которая была определена в сценарии EnemyDamage, происходит проверка тега объекта столкновения: если тег имеет значение Player(Игрок), то персонажу наносится некоторый урон. При этом функция вызывается только в случае вхождения в ловушку, то есть пока игрок передвигается внутри ловушки и при выходе из нее урон наноситься не будет.

Другим видом ловушки является летящая стрела: она двигается в воздухе на уровне игрока или чуть выше него, что ограничивает мобильность персонажа. Чтобы избежать получения урона, необходимо либо перепрыгнуть через стрелу, либо сделать кувырок.

Ловушка состоит из двух частей: сама ловушка и стрела. К стреле применен сценарий EnemyProjectile(вражеский снаряд), который наследуется от EnemyDamage, чтобы в последствии наносить урон игроку при попадании. Метод ActivateProjectile(активировать снаряд) активирует объект вместе с регистрацией столкновений. В функции событий Update стреле присваивается скорость, для движения против оси X, а также отсчитывается время после запуска снаряда. Если время станет больше некоторой заданной величины, то стрела будет деактивирована. Сделано это для повышения производительности, так как стрела потенциально может продолжать лететь, хотя игрок уже давно ее преодолел. В случае, если стрела столкнется на своем пути с любым объектом, то она деактивируется и передает классу-родителю информацию об столкновении.

Ловушка не имеет визуального представления в мире игры, она служит как источник выстрела стрелы. Сразу же после создания объекта в мире игры, она выпускает стрелу через функцию Attack(атаковать) сценария ArrowTrap(ловушка со стрелой). Внутри метода генерируется случайное целое число с помощью встроенных инструментов C#, которое определяет, на какой высоте полетит стрела. В конце мы обращаемся к методу ActivateProjectile объекта стрелы для выпуска снаряда.

Противником в игре являются вражеские рыцари, которые нападают на героя, когда тот попадает в зону атаки. Игрок может либо перепрыгнуть через противника, либо атаковать его в ответ, поразив его перед тем, как тот нанесет ему урон.

 Сценарий MeleeEnemy(рукопашный бой противника) функционирует схожим образом, как и сценарий атаки игрока, однако отличия есть. Вместо нажатия кнопки Левой кнопки мыши, катализатором для начала атаки для противника является вход игрока внутрь виртуального прямоугольника перед противником. То есть противник будет реагировать на появление игрока в «зоне видимости», чтобы начать нападение.

Получение очков является главной мотивацией игрока становиться лучше и продолжать играть, постепенно улучшая свой результат. Основную массу очков игрок получает за расстояние, которое он смог пробежать. Плюс к этому за каждого поверженного противника начисляется фиксированное число баллов. Если в забеге игрок набрал результат, превышающий свой предыдущий рекорд, то после перезапуска игры новый лучший результат перезапишет старый.

За управление системой создания новых препятствий перед игроком ответственен объект SpawnManager(управляющий созданием), который является пустым объектом с единственным компонентом, называющимся идентично. Функционирует это следующим образом: внутри Update запускается таймер. Когда время превышает время ожидания, то в игре создается новый случайный объект из списка, который заполнен экземплярами каждого из видов препятствий. Максимальное количество одновременно существующих в игровом мире препятствий ограничено сверху некоторым числом. Затем новая сущность помещается правее крайней границы камеры на высоте, которая вычисляется в зависимости от вида преграды. Частота появления новых препятствий будет случайной внутри все уменьшающегося диапазона, что будет увеличивать сложность игры с течением времени.

Для проигрывания всех звуков был создан специальный объект с компонентом AudioSource. Этот объект под названием SoundManager(управляющий звуками), станет источником всех звуков в игре. Сценарий SoundManager предоставляет публичный метод для воспроизведения звуков – PlaySound, принимающий аудиодорожку и тут же ее запускающий. Воспроизвести звук в других объектах можно следующим образом: сначала мы получаем ссылку на экземпляр источника звука внутри других скриптов, предварительно объявив ее как статическое поле instance, доступ к которому имеется всем объектам сцены, а затем на этом экземпляре вызываются все необходимые звуки. При этом сам объект не будет уничтожен после загрузки новой сцены, чтобы все настройки уровня громкости сохранялись между переходом между игрой и главным меню.

Объект GameManager(управляющий игрой) отвечает за конец и перезапуск игры, который происходят после смерти игрока. Фоновая музыка обрывается, а далее происходит отложенный вызов функции, которая показывает экран проигрыша. Отложенный вызов достигнут использованием встроенной функции Invoke, первым аргументом которой передается строка, содержащая название функции, которую необходимо запустить, а вторым – время ожидания.

После этого сцена загружается заново, перед этим сохранив лучший счет, если таковой был достигнут в этом сеансе. Все объекты принимают состояние, как в начале игры, что создает зацикленность игрового процесса.

Для сохранения данных игрока необходим специальный обработчик, который будет способ записывать и считывать информацию из файла, чтобы хранить лучший счет игрока даже после выключения игры. В сценарии DataManager контейнером для данных служит объект внутреннего класса Data, в полях которого в свою очередь хранится информация, которую необходимо сохранить, в нашем случае это лучший счет. В сценарии также хранится путь файла расширения json, в котором будет наход данные игры. При первом запуске игры лучший счет будет равным нулю. Для того, чтобы установить новое значение и получить текущее поля bestScore объекта типа Data, существуют соответствующие функции SetGameData и GetGameData.

Самыми важными, однако, являются методы Load и Save. Чтобы загрузить данные из файла, создается экземпляр класса StreamReader, встроенный в C#, затем все содержимое файла помещается в строку, из которой потом извлекается объект класса Data. В случае сохранения информации необходимо сначала перевести объект в Json-строку, а затем записать ее содержимое в файл посредством StreamWriter.

Одной из особенностей игры является система бонусов, помогающих игроку во время игрового процесса, которые появляются в мире игры на месте поверженного врага. Эти предметы разделяются на защитные(или предметы поддержки) и атакующие. Поднятые бонусы либо применяются мгновенно, либо помещаются в “инвентарь”, хранящий только первые 3 поднятых предмета. При этом остальные бонусы будут пропадать, если инвентарь заполнен на максимум.

Бонусы, которые можно будет применить самим игроком, хранятся в компоненте Bonuses. Все виды поднимаемых предметов находятся в перечислении CollectibleTypes, доступ к которому есть у всех сценариев приложения. При создании сцены сразу же берутся ссылки на компоненты GameManager для запуска анимаций внутри пользовательского интерфейса, SpriteRenderer игрока, чтобы визуально продемонстрировать применение некоторых бонусов, числовое значение слоев Player и Spike, и ссылку на элемент интерфейса, отображающий бонусы, которые игрок может использовать.

В каждом кадре сценарий ждет нажатия на кнопку F на клавиатуре, отвечающая за применение бонуса. Все предметы находятся в очереди Queue<CollectibleTypes> collected, ограниченного сверху заданным числом. Структура данных Очередь или еще называемое Queue реализует принцип "первый вошел - первый вышел". Новые элементы очереди добавляются в “хвост” списка, при этом взять можно только первый элемент, в данный момент находящийся в “голове”. Примером очередей могут служить очереди в магазине, в которых приоритетом при обслуживании обладает покупатель во главе очереди.

Первый попавший в список предмет будет использован в первую очередь. Поэтому после нажатия кнопки, switch-выражение проверяет значение первого элемента списка, и в зависимости от его вида, выполняет различные действия, в которых и будет заключаться функция каждого из бонусов.

Например, поднимаемый предмет DestructionCollectible уничтожает все препятствия, находящиеся в данный момент в игровой сцене. Чтобы это реализовать, для начала берется ссылка на экземпляр SpawnManager, чтобы затем вызвать на ней функцию emptySpawnedList(опустошить список), уничтожающая все объекты преград. Затем проигрываются анимации удара молнии с соответствующим звуком, которая символизирует разрушительную способность предмета, и из списка доступных бонусов убирается первый элемент. При этом содержимое “инвентаря” обновляется на экране.

case CollectibleTypes.Destruction:

SpawnManager spawnManager = (SpawnManager)FindAnyObjectByType(typeof(SpawnManager));

SoundManager.instance.PlaySound(lightningSound);

gameManager.PlayLightningAnim();

spawnManager.emptySpawnedList();

ui.RefreshUIList();

break;



Если же был применен бонус SteelShoes, то вызывается сопрограмма InvulnerabilityFromSpikes(неуязвимость от шипов), которая работает схожим образом с функцией Invulnerability сценария Health, отключающая регистрацию столкновения слоев игрока и шипов, при этом изменяя цвет персонажа на серый.

case CollectibleTypes.SteelShoes:

SoundManager.instance.PlaySound(steelShoeSound);

ui.RefreshUIList();

StartCoroutine(InvulnerabilityFromSpikes());

 break;

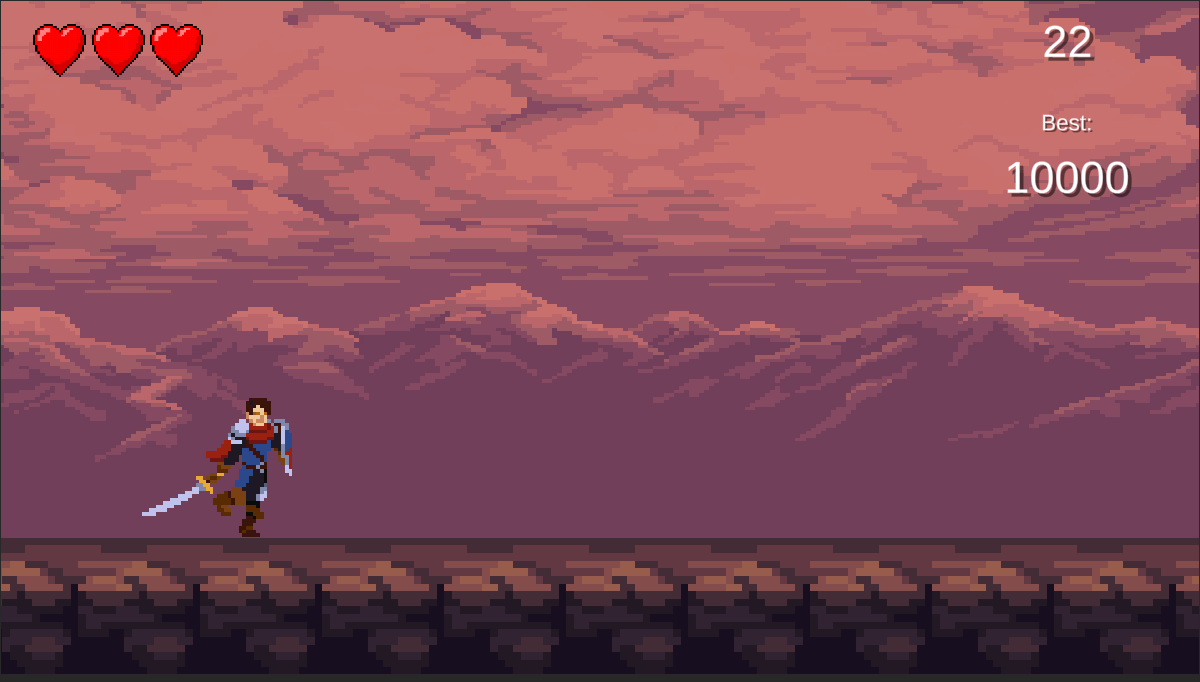
Объекты поднимаемых предметов при касании с игроком, должны либо моментально примениться, либо добавиться в инвентарь игрока. Общим свойством, которым обладают все поднимаемые предметы, является уничтожение объекта, если его не подняли в течении некоторого времени. Это находится внутри скрипта Collectible, от которого будут наследоваться сценарии всех видов бонусов.

Бонус HealthCollectible повышает здоровье игрока на одну единицу сразу после столкновения с игроком. DestructionCollectible и SteelShoesCollectible сохраняются в “инвентаре” игрока для будущего использования.

Чтобы игрок обладал всей информацией, необходимой ему для прохождения игры, того, что происходит на экране, недостаточно. Информация о количестве здоровья, сколько было набрано очков и так далее, показать напрямую в игровом мире зачастую невозможно или неудобно. Для этой цели служит специальный экран, который накладывается поверх игрового процесса, называемый UI(user interface) или пользовательский интерфейс.

В Unity все элементы интерфейса должны находиться внутри специальной зоны Canvas, играющую роль холста для всего интерфейса. Если создать элемент UI без существующего холста в сцене, то он будет автоматически добавлен за нас и изначальный элемент станет потомком Canvas. Отображаться пользовательский интерфейс может либо поверх всей сцены, либо прямо перед камерой игры или вообще являться таким же объектом мира, как и остальные сущности. В зависимости от предназначения интерфейса могут реализоваться разные варианты.

Элементы UI являются дочерними по отношению к холсту. Их положение на экране, угол поворота и размер меняется теми же инструментами, что остальные объекты в Unity, то есть чтобы разместить элемент в углу интерфейса, достаточно просто перетащить его туда. Это достигается тем, что каждый объект интерфейса Unity имеет компонент RectTransform, который имеет положение, масштаб и угол поворота, как и обычный Transform, но в дополнению к ним добавлены значения ширины и высоты, определяющие размерность прямоугольника, внутри которого будет находиться текст, изображения и так далее.

Интерфейс “Название\_игры” во время игрового процесса выглядит следующим образом:

Изображения сердец обозначают количество очков здоровья игрока, при получении урона одно из сердец закрашивается черным цветом, пока жизнь игрока не закончится. В правом углу сверху находится счетчик очков, полученных в этой сессии, в то время как ниже можно увидеть свой лучший результат за все время игры.

В нижнем левом углу экрана расположены бонусы, которые игрок может использовать. Они обновляются каждый раз, когда игрок применяет один из них, или подбирает новый. Для этого внутри холста интерфейса есть объект, содержащий некоторое количество элементов с компонентом для отображения картинок. Чтобы отобразить в этих элементах список бонусов, проходим циклом по всем доступным картинкам, для каждой картинки устанавливая изображение бонуса, который находится в “инвентаре” игрока. Если бонусов меньше, чем максимально возможное количество, то для картинок, которым не досталось своего бонуса, устанавливается пустое значение.

toVisualize = bonuses.getList();

for(int i=0;i< imageList.Count;i++)

{

imageBuffer = imageList[i].GetComponent<Image>();

if (i< toVisualize.Count)

{

switch (toVisualize[i])

{

case CollectibleTypes.Destruction:

imageBuffer.enabled = true;

imageBuffer.sprite = destructImage;

break;

case CollectibleTypes.SteelShoes:

imageBuffer.enabled = true;

imageBuffer.sprite = steelShoesImage;

break;

case CollectibleTypes.Default:

imageBuffer.enabled = false;

imageBuffer.sprite = null;

break;

}

}

else

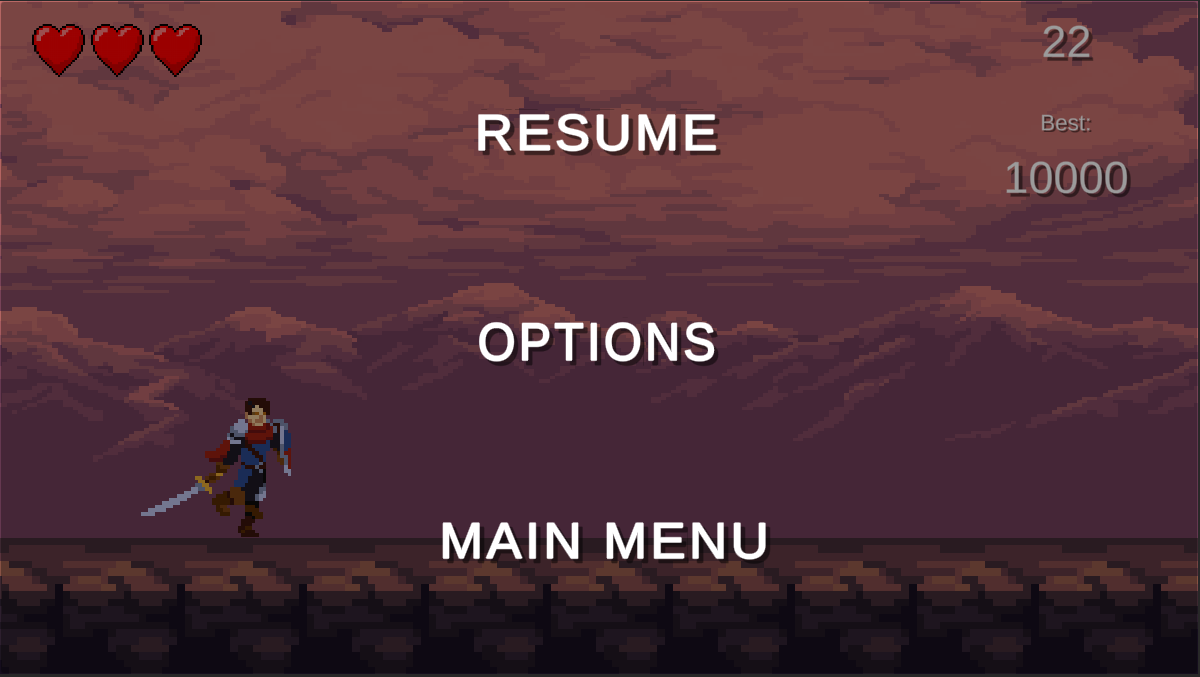
{

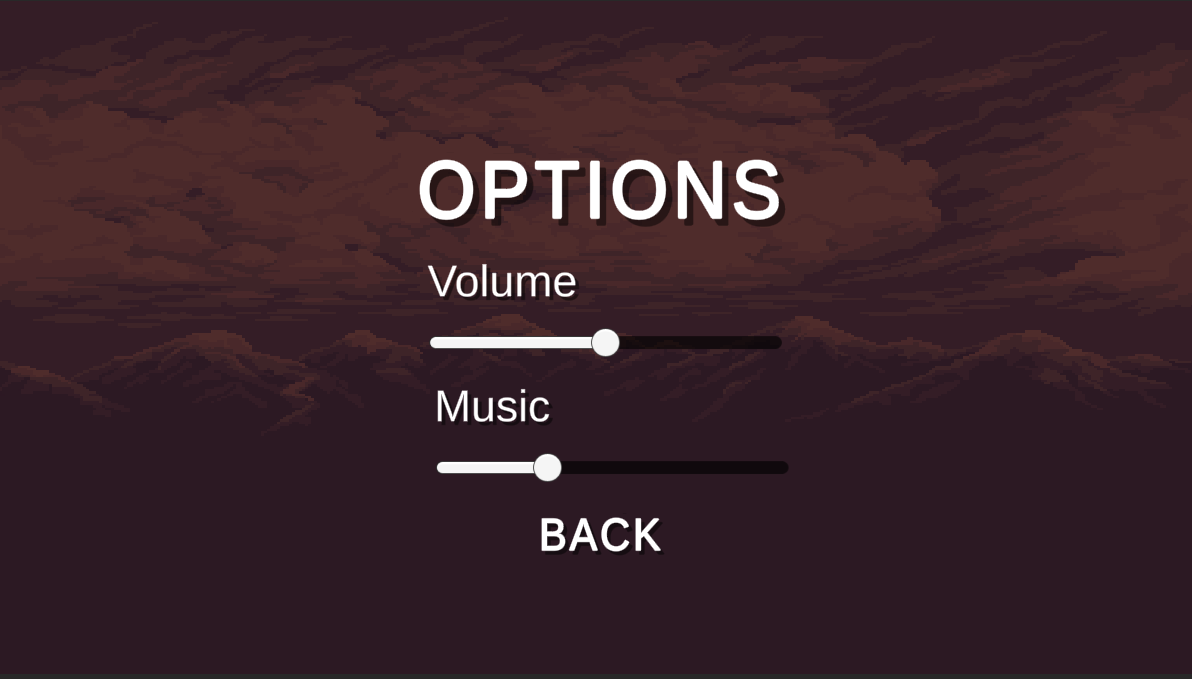
imageBuffer.enabled = false;

imageBuffer.sprite = null;

}

}

Чтобы приостановить игру на некоторое время, игрок может нажать клавишу Escape. В результате появится меню паузы, в котором игрок может выбрать продолжить игру, изменить настройки громкости звука и музыки игры или выйти в главное меню приложения. При этом игра будет остановлена на фоне путем установления значения скорости, с которой идет время внутри игры, на 0. Выйти из меню можно нажав кнопку Escape еще раз.

При запуске приложения игры сначала загружается сцена главного меню, из которой можно перейти к игровому процессу, нажав кнопку Play. При нажатии на кнопку Options, появляется окно, где можно изменить громкость игры отдельно для звуков и музыки. Чтобы выйти из приложения, есть возможность нажать на Quit, что выключит игру.

Когда игрок погибает, прямо перед перезапуском уровня появляется экран, возвещающий о конце игры.

Изображения, необходимые для создания внешнего вида игры, были взяты из Unity Asset Store, где они распространялись бесплатно. При выборе спрайтов отдавалось предпочтение тем картинкам, где автор придерживался стиля игр 60-80-х годов, называемый сейчас Pixel Art. Pixel Art – это форма цифрового рисунка, при котором пиксели используются как единственный инструмент для создания изображения. Художникам, работающим в таком стиле, по причине ограниченной свободы выражения должны проявлять находчивость и изобретательность, чтобы создать красивое полотно, пусть и цифровое.

Покадровая анимация удара молнии при использовании бонуса разрушения в игре была нарисована специально для игры посредством веб-приложения PixilArt. Этот сайт позиционируется как социальная платформа, где художники со всего света могли поделиться своими картинами, созданными в стиле пиксельной графики. Помимо этого веб-приложение предоставляет бесплатные инструменты для создания пиксельного изображения. Картины можно разбивать на слои, рисуя отдельные части на разных полотнах, в конце объединяя все воедино. Можно выбрать различные цветовые палитры из встроенных или создать их самому. PixilArt дает возможность создавать анимации, позволяя нарисовать каждый кадр по отдельности, которые можно затем импортировать уже готовой записью в файле с расширением gif, или расположить каждый кадр друг за другом на одном изображении, что облегчает процесс переноса анимации для реализации внутри платформ для создания игр, например, Unity.

<https://ru.wikipedia.org/wiki/Dinosaur_Game>

<https://trex-runner.com/>

<https://unity.com/ru>

<https://learn.microsoft.com/en-us/visualstudio/gamedev/unity/get-started/visual-studio-tools-for-unity?pivots=windows>

<https://docs.unity3d.com/ru/2019.4/Manual/Animator.html>

<https://docs.unity3d.com/ru/2019.4/Manual/AnimationClips.html>

<https://docs.unity3d.com/ru/2019.4/Manual/AnimationOverview.html>

<https://docs.unity3d.com/ru/2019.4/Manual/StateMachineBasics.html>

<https://docs.unity3d.com/Manual/Hierarchy.html>

<https://docs.unity3d.com/Manual/UsingTheSceneView.html>

<https://docs.unity3d.com/Manual/GameView.html>

<https://docs.unity3d.com/Manual/Layers.html>

<https://docs.unity3d.com/Manual/use-layers.html>

<https://docs.unity3d.com/Manual/GameObjects.html>

<https://docs.unity3d.com/Manual/Tags.html>

<https://ru.wikipedia.org/wiki/C_Sharp>

<https://ru.wikipedia.org/wiki/Аркада_(жанр)>

<https://ru.wikipedia.org/wiki/Игровой_движок>

<https://ru.wikipedia.org/wiki/Unity_(игровой_движок)>

<https://docs.unity3d.com/ScriptReference/Tilemaps.Tilemap.html>

<https://docs.unity3d.com/ScriptReference/Grid.html>

<https://blog.logrocket.com/fundamentals-unity-2d-colliders/>

<https://docs.unity3d.com/ScriptReference/Rigidbody2D.html>

<https://docs.unity3d.com/ru/2019.4/ScriptReference/MonoBehaviour.Start.html>

<https://docs.unity3d.com/ru/2019.4/Manual/EventFunctions.html>

<https://habr.com/ru/articles/682320/>

<https://docs.unity3d.com/ScriptReference/MonoBehaviour.StartCoroutine.html>

<https://metanit.com/sharp/tutorial/4.11.php>

<https://docs.unity3d.com/ScriptReference/SerializeField.html>

<https://docs.unity3d.com/ScriptReference/MonoBehaviour.OnCollisionEnter2D.html>

<https://docs.unity3d.com/ScriptReference/Collider.OnTriggerEnter.html>

<https://docs.unity3d.com/ScriptReference/Gizmos.html>

<https://docs.unity3d.com/ScriptReference/MonoBehaviour.OnDrawGizmos.html>

https://www.pixilart.com/

# Общие положения

## Курсовые работы исследовательского характера

Курсовые работы исследовательского, опытно-экспериментального, программно-исследовательского характера предполагают формирование у студентов навыков теоретических и экспериментальных исследований, в том числе умения самостоятельной критической обработки научных материалов и источников и их практической реализации; умения анализировать современный опыт; способностей формулировать выводы, предложения, рекомендации, делать оценки результатов исследования, проводить инженерные расчеты и др. Они нацелены на углубление знаний студентов по актуальным проблемам; стимулирование их к самостоятельному научному поиску.

Курсовая работа исследовательского, опытно-экспериментального, программно-исследовательского характера должна содержать результаты теоретических и (или) экспериментальных исследований по группе учебных дисциплин, характеристику методов экспериментальной работы, обоснование выбранного метода, основные этапы эксперимента, обработка и анализ результатов опытно-экспериментальной работы.

Основная часть курсовой работы исследовательского, опытно-экспериментального характера состоит из:

* раздела, содержащего теоретические основы разрабатываемой темы, где даны история вопроса, уровень разработанности проблемы в теории и практике;
* разделов, в которых содержится самостоятельный анализ проблемы, излагаемой в курсовой работе, либо план проведения эксперимента, в том числе в форме математического моделирования на виртуальном (программном) объекте и/или в форме практического испытания на реальном объекте, характеристики методов экспериментальной работы, обоснование выбранного метода, основные этапы эксперимента, обработка и анализ результатов опытно-экспериментальной работы.

Курсовую работу программно-исследовательского характера можно отнести к разновидности опытно-экспериментальной работы. Целью такой работы является исследование недокументированных (плохо документированных) функций, параметров и характеристик разработанной другими лицами программы или программного пакета. Основная часть курсовой работы программно-исследовательского характера состоит из:

* раздела, содержащего информационные основы разрабатываемой темы, где даны краткие описания известных функций, параметров и характеристик исследуемой программы (пакета), полученные из литературных источников, включая Интернет, обоснование требований на исследования неизвестных функций, параметров и характеристик программы (пакета);
* разделов, в которой содержится план исследований, указаны основные этапы исследований, выполнена обработка, анализ и формулировка полученных результатов в виде описания полученных параметров, характеристик и исследованных функций программы (пакета).

## Курсовые работы реферативного характера

Курсовая работа реферативного характера является реферативным обобщением оригинальных теоретических исследований по конкретной научной проблеме

Она представляет собой полный и систематизированный критический обзор литературы по конкретной научной проблеме, включая патентный поиск с анализом материалов по одному из вопросов. В ней может анализироваться история вопроса, уровень разработанности проблемы в теории и практике, полученные посредством сравнительного анализа литературы.

# Иллюстрации, формулы и приложения

## Общие требования к оформлению

Набор текста работы осуществляется с использованием текстового редактора Word. При этом рекомендуется использовать шрифты типа Times New Roman размером 14 пунктов. Количество знаков в строке должно составлять 60–70, межстрочный интервал должен составлять 18 пунктов, количество текстовых строк на странице – 39–40.

Устанавливаются следующие размеры полей: верхнего и нижнего – 20 мм, левого – 30 мм, правого – 10 мм.

Шрифт печати должен быть прямым, светлого начертания, четким, черного цвета, одинаковым по всему объему текста работы. Разрешается использовать компьютерные возможности акцентирования внимания на определениях, терминах, важных особенностях, применяя разное начертание шрифта: курсивное, полужирное, курсивное полужирное, выделение с помощью рамок, разрядки, подчеркивания и другое.

Объем работы, как правило, должен составлять для курсовых – 25–35 страниц, для дипломных – 50–70 страниц. Иллюстрации, таблицы, список использованных источников и приложения при подсчете объема работ не учитываются.

## Заголовки структурных частей работы

Заголовки структурных частей работы «Оглавление», «Перечень условных обозначений», «Введение», «Глава», «Заключение», «Список использованных источников», «Приложения» печатают прописными буквами в середине строк, используя полужирный шрифт с размером на 1–2 пункта больше, чем шрифт в основном тексте. Так же печатают заголовки глав.

### Заголовки разделов

Заголовки разделов печатают строчными буквами (кроме первой прописной) с абзацного отступа полужирным шрифтом с размером на 1–2 пункта больше, чем в основном тексте.

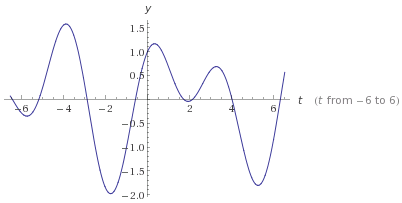


Рисунок 2.1 График функции

### Заголовки подразделов

Заголовки подразделов печатают с абзацного отступа строчными буквами (кроме первой прописной) полужирным шрифтом с размером шрифта основного текста.

Пункты, как правило, заголовков не имеют. При необходимости заголовок пункта печатают с абзацного отступа полужирным шрифтом с размером шрифта основного текста в подбор к тексту.

В конце заголовков глав, разделов и подразделов точку не ставят. Если заголовок состоит из двух или более предложений, их разделяют точкой (точками). В конце заголовка пункта ставят точку.

## Иллюстрации и таблицы

Иллюстрации (фотографии, схемы, диаграммы, графики, карты и другое) и таблицы служат для наглядного представления в работе характеристик объектов исследования. Не допускается одни и те же результаты представлять в виде иллюстрации и таблицы.

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |

Таблица 2.1 Пустая таблица

Иллюстрации и таблицы следует располагать в работе непосредственно на странице с текстом после абзаца, в котором они упоминаются впервые, или отдельно на следующей странице. Они должны быть расположены так, чтобы их было удобно рассматривать без поворота работы или с поворотом по часовой стрелке. Иллюстрации и таблицы, которые расположены на отдельных листах работы, включают в общую нумерацию страниц.

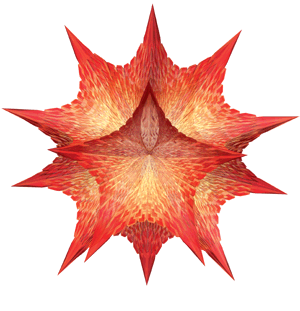


Рисунок 2.2 Логотип Mathematica

Иллюстрации и таблицы обозначают соответственно словами «рисунок» и «таблица» и нумеруют последовательно в пределах каждой главы. На все таблицы и иллюстрации должны быть ссылки в тексте работы. Слова «рисунок» «таблица» в подписях к рисунку, таблице и в ссылках на них не сокращают.

Номер иллюстрации (таблицы) должен состоять из номера главы и порядкового номера иллюстрации (таблицы), разделенных точкой. Например: «рисунок 1.2» (второй рисунок первой главы), «таблица 2.5» (пятая таблица второй главы). Если в главах приведено лишь по одной иллюстрации (таблице), то их нумеруют последовательно в пределах работы в целом, например: «рисунок 1», «таблица 3».

## Оформление научно-справочного аппарата

Студент обязан при написании работы давать ссылки на источники, материалы или отдельные результаты из которых приводятся в его работе или на идеях и выводах которых разрабатываются проблемы, задачи, вопросы, изучению которых посвящена работа. Такие ссылки дают возможность найти соответствующую литературу и проверить достоверность цитирования, а также необходимую информацию об этом источнике (его содержание, язык, объем и другое). Если один и тот же материал переиздается неоднократно, то следует ссылаться на его последнее издание. На более ранние издания можно ссылаться лишь в тех случаях, когда в них есть нужный материал, не включенный в последние издания.

При использовании сведений из источника с большим количеством страниц студент должен указать в том месте работы, где дается ссылка на этот источник, номера страниц, иллюстраций, таблиц, на которые дается ссылка в работе. Например: «[14, с. 26, таблица 2]» (здесь 14 – номер источника в списке использованной литературы, 26 – номер страницы, 2 – номер таблицы), или «[18, с. 44]» (здесь 18 – номер источника в списке использованной литературы, 44 – номер страницы).

заключение

В заключении логически и последовательно излагаются теоретические и практические выводы по каждому разделу курсовой работы. Выводы и предложения должны быть конкретными, реальными и обоснованными, вытекать из результатов проведенного исследования.

Выводы пишутся тезисно (по пунктам). Из каждого подраздела теоретической части рекомендуется в заключение включать не более одного вывода.

Список использованной литературы

Шафрин Ю. Информационные технологии: в 2 частях / Ю. Шафрин. – М.: Бином, Лаборатория знаний, 2004. – Часть 1. Основы информатики и информационных технологий.

Голубева Л. Л. Компьютерная математика. Автоматизированное рабочее место математика: курс лекций / Л. Л. Голубева, А. Э. Малевич, Н. Л. Щеглова. – Минск: БГУ, 2008. – 139 с.

MathWorks MATLAB Documentation [Электронный ресурс] – Режим доступа: <http://www.mathworks.com/help/matlab/index.html> – Дата доступа: 15.04.2014.

1. Код программы

Ниже приведен код алгоритма на Mathematica

ListPlot[Tally[RandomInteger[BinomialDistribution[50, 0.3], 10^4]],

**PlotRange -> {{0, 40}, All}, Filling -> Axis]**

ParametricPlot[

Table[{x[t], y[t]} /. % /. {a -> 1/(13 + m), b -> 1/(15 + m)}, {m, 0,

20, 7}], {t, -2, 2}, Evaluated -> True]