Министерство образования Республики Беларусь
Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерного проектирования

Кафедра проектирования информационных компьютерных систем

Дисциплина "Объектно-ориентированное программирование"

К защите допустить:
Руководитель курсовой работы
старший преподаватель
кафедры
_____ А.В.Михалькевич
16.07.2025

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

на тему

Разработка мобильной игры на android

БГУИР КР 1-40 05 01-10 № 126 ПЗ

Студент (подпись студента) Е.С. Ревяко

Курсовая работа
представлена на проверку
16.07.2025
_____
(подпись студента)

Минск 2025

Реферат

БГУИР КР 1-40 05 01-10 № 126 ПЗ, гр. 814303

Е.С. Ревяко, Разработка мобильной игры на android, Минск: БГУИР - 2025.

Пояснительная записка 81075 с., 7 рис., 0 табл.

Ключевые слова: ИГРА, ANDROID, UNITY 3D, C#

Предмет Объектно-ориентированное программирование, А.В.Михалькевич

Предмет: создание мобильной игры. Объект: шаблоны проектирования, разработка игровой концепции. Цель: создание мобильной игры на игровом движке Unity 3D. Методология проведения работы: в процессе решения поставленных задач спроектирована и разработана концепция игры, разработана логика игры и взаимодействие пользователя с игровым персонажем. Результаты работы: разработаны сцены главного меню, меню выбора уровней и 15 уровней. Разработана логика игры, добавлена анимация на игрового персонажа, добавлено звуковое сопровождение. Область применения результатов: удовлетворение пользователей, желающих отдохнуть от повседневных дел или находящихся в транспорте и желающих скоротать время. Ссылка на онлайн-репозиторий GitHub: https://github.com/foxwe11/Infinite-level

Subject: creating a mobile game. Object: design templates, game concept development. Goal: create a mobile game based on the Unity 3D game engine. Methodology of the work: in the process of solving the tasks, the game concept was designed and developed, the game logic and user interaction with the game character were developed. Results: developed scenes of the main menu, level selection menu and 15 levels. Developed game logic, added animation for the game character, added sound. The scope of the results: satisfaction of users who want to take a break from everyday activities or are in transport and want to pass the time. Link to the github online repository: https://github.com/foxwe11/Infinite-level

Содержание

Введение

Актуальность темы Мобильные игры - это своего рода «таймкиллеры»: приложения, созданные для того, чтобы скоротать несколько минут в вагоне метро, занять себя чем-то в перерыве на работе или даже развлечь себя перед сном. Мобильные игры рассчитаны на то, чтобы пользователь смог быстро их запустить и так же быстро из них выйти при необходимости. На

сегодняшний день рынок мобильных игр не стоит на месте и активно развивается. Ежемесячно в магазинах приложений Google Play и App Store появляются тысячи новых игр. Конечно качество многих из них оставляет желать лучшего, но пользователей привлекает не только качество, а также новизна, новые игровые механики или новые сочетания игровых механик. Под новизной понимается новый игровой контент, если пользователь проходит одну из игр и ему нравятся игры данного жанра, то он с большой вероятностью будет искать подобную игру, но более качественную и популярную, с хорошими отзывами других пользователей. Цель и задачи Целью работы является разработка мобильной игры для ОС Android с использованием игрового движка Unity3D. Для достижения поставленной цели необходимо решить следующие задачи: 1. Выбрать жанр мобильной игры; 2. Разработать игровую механику мобильной игры; 3. Реализовать мобильную игру; 4. Протестировать игру на наличие неполадок. Детальная постановка реализации мобильной игры 1. Из игровых объектов составить игровой мир; 2. Написать скрипт управления персонажем; 3. Написать скрипт управления логикой игрового мира; 4. Добавить анимацию на игровые объекты; 5. Добавить звуки на игровые события;

1 1. Описание проекта

Создание игры происходит в несколько этапов: создание концепции игры, выбор жанра и создание игровой механики.

1.1 Концепция игры

Первым шагом в создании дизайна для мобильной F2P игры — будет концепция. Концепция должна включать в себя основную информацию об игре, причины создания именно такой игры и анализ рыночной ситуации. Результат работы должен выглядеть так:

Жанр

Анализ рынка — популярность и доступность жанра

Тематика и целевая аудитория

— Анализ рынка — популярность и доступность тематики

— Возможные франчайзы, которые подойдут для вашей игры

— Определение целевой аудитории — кому ваша игра будет интересна?

Определение “Emulation target” — игра или несколько игр, на основе которых строится концепция

— Это не значит, что надо копировать другую игру, хотя это также является одним из вариантов. — Чтобы другие люди поняли, что имеется введу, примеров из других игр не избежать

-Анализ успешности данных игр, как конкурентов

Краткое описание игры

-Небольшое сюжетное вступление, чтобы представить, кто такой игрок и почему он делает то, что делает

-Описание цели игрока

Описание главных составляющих игры
-Чем игроки будут заниматься большинство времени
-Причины по которым этим интересно заниматься
-Причина по которым игроки останутся в игре на длительное время

Главные отличительные особенности игры
-Чем игра лучше конкурентов
-Что в первую очередь притянет взгляд игроков
-Что заставит игроков остаться, а не уйти обратно к конкурентам

Экономика игры
-Контент, за который игрок будет платить
-Причины, по которым игрок будет платить

Core Loop — последовательность основных действий игрока в игре
-Не забыть включить источники и траты ресурсов

Социальные составляющие игры
-Список социальных составляющих
-Взаимодействие между игроками
-Причины, по которым игроки будут мотивированы социализироваться

Дополнительные детали
-Длинна сессии
-Ожидаемое количество контента
-Ожидаемый срок разработки
-Онлайн/оффлайн игра

Поддержка игры после выпуска
-Какой способ поддержки игры будет оптимальным
-Насколько ресурсно затратным этот способ будет
-Какие ожидания в длительном будущем от игры

1.2 Выбор жанра

Выбирая жанр для своей игры, оптимально будет выбирать из тех, что имеют шанс оказаться в топ 50. Не стоит забывать про конкурентов и, к примеру, выбрав за основу популярную игру, стоит задуматься, сможете ли вы предложить рынку что-то, чего еще нет. Сделав очередной клон этой игры ваша игра маловероятно придет к успеху.

Стоит задуматься о том, что вы или команда, с которой вы планируете работать умеете лучше всего — какая специфика ближе для вас. Если до этого вы делали только шутеры, то

создать ресурс менеджмент игру будет намного сложнее.

И последним, но не менее важным пунктом будут — ресурсы, которыми вы располагаете. Например, если выбранный жанр — Racing, то команда должна быть способна создать не менее 40 различных автомобилей, не менее 5-10 трасс и т.д. У каждого жанра есть свои минимальные требования по контенту.

Следовательно при выборе жанра стоит руководствоваться данными переменными:

-Потенциальная успешность на рынке

-Уровень конкуренции выбранного жанра

-Сможет ли ваша игра предложить что-то при таком уровне конкурентности

-Экспертиза существующей команды

-Ресурсоемкость существующей команды

1.3 Механика игры

Игровая механика — вот что делает игровое пространство действительно очень интересным. «Игровая механика» — это другое определение для того, что многие обычно называют правилами. Между тем в индустрии термин «механика» является общим местом. Механика — это то, как что-то работает. Если ты делаешь X, то произойдет Y. Если X верно, то ты можешь сделать Y. В «Монополии» если ты походил на собственность, то ты можешь её купить. Если ты выбрасываешь наибольшее число, ты ходишь первым. Всё это простые механики.

Иными словами, механика — это правила, которые затрагивают игроков, аватары, игровые частицы, игровое состояние, поле обозрения и описывают все способы изменения игрового состояния.

Механики — ингредиенты геймдизайна. Их понимание крайне необходимо для всех геймдизайнеров. Вот некоторые общие категории механик, которые обычно можно найти в играх:

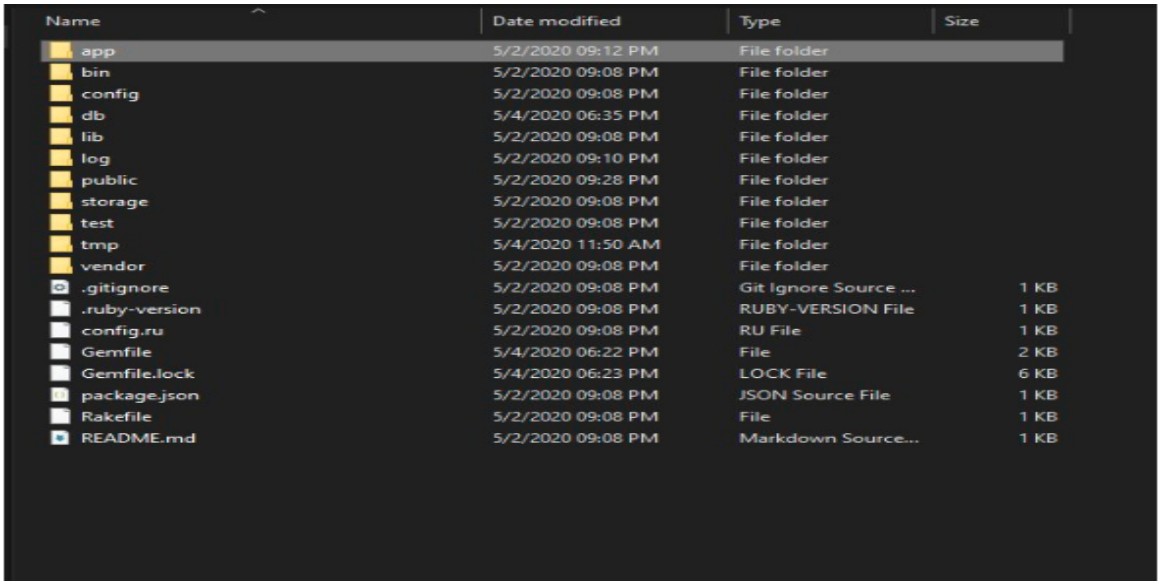
Установка. Всегда должно наличествовать хотя бы одно правило, описывающее то, как игра начинается.
Условия победы. Всегда должно присутствовать хотя бы одно правило, описывающее то, как выиграть в игре. Некоторые игры вроде неограниченных ролевых игр (RPG) не содержат условий победы. Поэтому некоторые дизайнеры не рассматривают таковые в качестве игр. Другие считают, что выполнение особенной задачи — это победа, потому что после этого игрок переключается на следующую задачу.
Ход игры. Кто ходит первым и как? Игра пошаговая или риал-тайм? Если пошаговая, то она начинается с определённого игрока, а потом все ходят по часовой стрелке или игроки выставляют ресурсы на аукцион для борьбы за право ходить первым в каждом раунде, или согласно какому-то другому методу? Если игра риал-тайм и два игрока пытаются делать что-то в одно и то же время, то как это разрешается?
Действия игрока. Иногда определяются как «команды», некоторые из наиболее важных механик, характеризующие то, что игроки могут делать и какой эффект эти действия оказывают на игровое состояние.
Определения поля(-ей) обозрения. Механика точно определяет, какой информацией игрок может обладать в любой отдельный момент времени. Отметим, что некоторые механики могут изменять поле обозрения, как пример — частично рассеивающийся при определенных условиях туман войны в RTS.

Некоторые комбинации механик по сравнению с другими таковыми людям усвоить гораздо проще. Например, в настольной игре Scotland Yard Mr. X скитается по окрестностям Лондона, пытаясь избежать поимки. Он проявляется только на каждый третий ход, а все остальное время остается скрытым. Игра основана исключительно на стратегии. И в нее людям играть не так легко, как, скажем, в Sorry.

1.4 Создание концепции данного проекта

В данном проекте планируется создать игру на основе уже существующей мобильной игры под названием «That level again». Эта игра имеет жанр головоломка. Она представляет из себя 2D платформер со статичной сценой (рисунок 1.1). Целью игры является открыть дверь что справа и выбраться наружу. На первом уровне для того, чтобы дверь открылась нужно встать персонажем на кнопку (рисунок 1.2). После того как проходит в правую область осуществляется переход на следующий уровень. На втором уровне наблюдается такая же локация, однако меняется условие для открытия двери (рисунок 1.3). Условие для открытия двери на каждом уровне описывается в виде подсказки в верхнем левом углу.

Рисунок 1.1 – игра «That level again» уровень 1



Name	Date modified	Type	Size
app	5/2/2020 09:12 PM	File folder	
bin	5/2/2020 09:08 PM	File folder	
config	5/2/2020 09:08 PM	File folder	
db	5/4/2020 06:35 PM	File folder	
lib	5/2/2020 09:08 PM	File folder	
log	5/2/2020 09:10 PM	File folder	
public	5/2/2020 09:28 PM	File folder	
storage	5/2/2020 09:08 PM	File folder	
test	5/2/2020 09:08 PM	File folder	
tmp	5/4/2020 11:50 AM	File folder	
vendor	5/2/2020 09:08 PM	File folder	
.gitignore	5/2/2020 09:08 PM	Git Ignore Source ...	1 KB
.ruby-version	5/2/2020 09:08 PM	RUBY-VERSION File	1 KB
config.ru	5/2/2020 09:08 PM	RU File	1 KB
Gemfile	5/4/2020 06:22 PM	File	2 KB
Gemfile.lock	5/4/2020 06:23 PM	LOCK File	6 KB
package.json	5/2/2020 09:08 PM	JSON Source File	1 KB
Rakefile	5/2/2020 09:08 PM	File	1 KB
README.md	5/2/2020 09:08 PM	Markdown Source...	1 KB

Рисунок 1.2 – игра «That level again» уровень 1. Прохождение

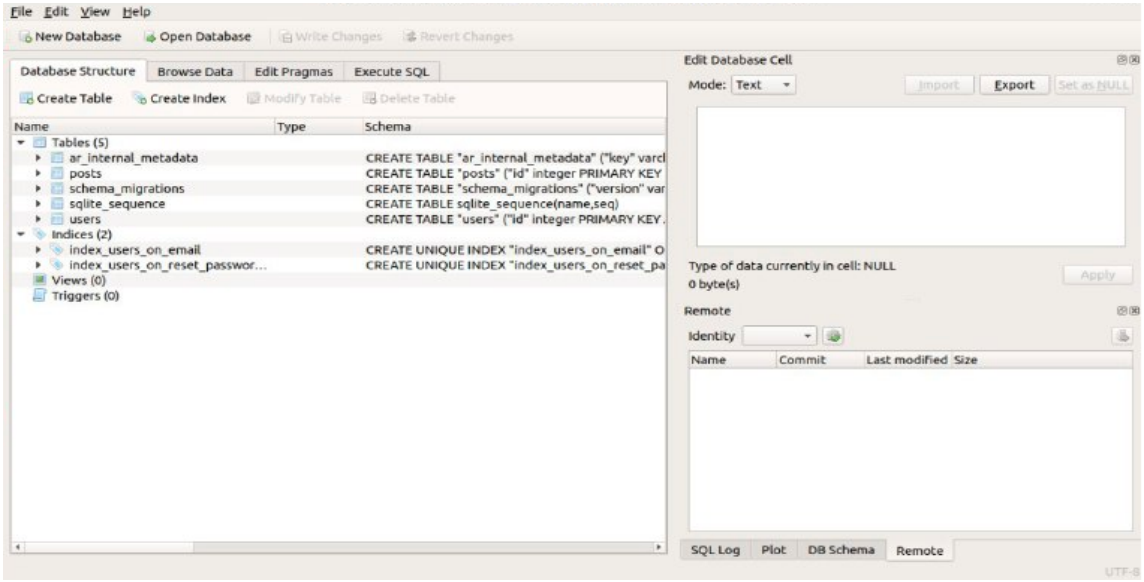


Рисунок 1.3 – игра «That level again» уровень 2

Создание игры будет происходить на игровом движке. Для начала требуется создать спрайты (графические объекты компьютерной графики для 2D игр) или найти в интернете уже готовые объекты. Затем спрайты надо перенести на игровую сцену и составить из них уровень. После добавить управляемому персонажу и поверхностям (таким как пол, стена, потолок) физические элементы. Также требуется написать скрипты осуществляющие логику игры. В конце добавляется звуковое сопровождение.

2 2. Обоснование выбора технологий

2.1 Технологии программирования, используемые для решения поставленных задач

Unity 3D

Unity – кросс-платформенный инструмент для разработки двухмерных и трехмерных приложений и игр, работающий под операционными системами Windows и OS X. Позволяет разрабатывать под все самые известные платформы, такие как: PC, Linux, Mac, IOS, Android, Xbox One, PS4 и т.д. Unity позволяет использовать такие языки программирования как C#, BOO и Javascript.

Unity имеет очень простой интерфейс, который разбит на несколько окон: Hierarchy, где находятся названия всех объектов на сцене, которые можно группировать; Scene, где можно рассмотреть игровое поле под нужным ракурсом; Inspector, в котором находятся все свойства выделенного объекта и его компоненты; Project, где находятся все материалы проекта; Toolbar, где находится меню с инструментами. Проект в Unity делится на сцены – отдельные файлы, содержащие свой набор объектов, скриптов и настроек. Основным объектом игровой логики является игровой объект – сущность, которая включает в себя компоненты. Также Unity предоставляет интегрированные сервисы для вовлечения, удержания и монетизации игроков.

Достоинства:

1) удобство использования и простота освоения;

2) качественная документация;

3) большое сообщество разработчиковиспользующих Unity;

4) возможность настроить и доработать среду разработки под нужный проект;

5) интегрированные сервисы монетизации и аналитики;

6) кросс-платформенность.

Недостатки:

1) необходимо глубокое знание одного из используемых языков программирования;

2) обновления могут испортить уже рабочий код;

3) условно-бесплатный.

Unreal Engine 4

Unreal Engine 4 – кросс-платформенный инструмент для разработки игр, работающий под операционными системами Windows и OS X. Позволяет вести разработку игр под все популярные платформы, такие как: PC, Mac, Linux, Android, IOS и другие. Исходные коды движка находятся в открытом доступе, поэтому при желании можно его доработать под свои нужды.

Окно редактора состоит из стандартных окон, такие как: Scene outliner, где находятся содержимое сцены; Content Browser, где находятся все файлы и материалы проекта; Details – окно свойств объекта; Modes – режим работы с контентом. Основным объектом игровой логики является элемент Blueprint – чертеж. Это сборка из компонентов, которая образует сложный объект игрового мира. Управление этим объектом осуществляется с помощью C++ класса или редактора графов. Вместе они дополняют друг друга.

Достоинства:

1) качественная документация;

2) большое сообщество разработчиков использующих Unreal Engine;

3) большое количество обучающего материала;

4) кросс-платформенность.

Недостатки:

1) сложность в освоении;

2) необходимость отдавать 5 % прибыли от игры.

Defold

Defold – кросс-платформенный инструмент для профессиональной разработки игр от компании King, известной своими 2D играми. Является абсолютно бесплатным. Defold имеет простой и легкий в освоении интерфейс, а набор инструментов предназначен для работы с 2D проектами. Вся игровая логика контролируется с помощью скриптов, написанных на языке Lua. При создании пользовательских материалов имеется возможность использования OpenGL ES. Платформа была запущена в марте 2016 года, и до сих пор находится в бета тестировании, поэтому у нее нет большого сообщества разработчиков.

Достоинства:

1) направленность на 2D;

2) абсолютно бесплатный;

3) перспективный.

Недостатки:

1) мало обучающих материалов;

2) поддержка только одного языка программирования.

По итогу обзора, для разработки игрового приложения, была выбрана платформа Unity 3D, так как она обладает наибольшим числом достоинств. Готовые игровые ресурсы для проекта я буду искать на следующих сайтах:

OpenGameArt.Org — один из самых известных сайтов с бесплатными игровыми ресурсами, доступными как Creative Commons. Но я хочу отметить раздел Collect с готовыми тематическими подборками, которые очень экономят время.

Craftpix, раздел Freebies — более 80 бесплатных пакетов 2D-графики для RPG, стратегий, аркад, платформеров и других типов игр. Тайлы, персонажи, GUI, иконки, фоны.

Open Game Graphics — гора плюшек для 2D-игр. Помимо дизайна интерфейсов, персонажей и уровней можно скачать 25 полных коллекций графики под игру конкретного жанра и антуража: мрачный Sci-Fi-платформер, красочный Top-down shooter, 8-битный рогалик и так далее. Много спрайтов в мультяшном стиле.

PixelGameArt — фэнтезийные и Sci-Fi-ассеты в стиле пиксель-арт с возможностью предпросмотра демок в браузере!

Kenney — поставщик игровых ресурсов, который предлагает около 60 бесплатных наборов ассетов, в том числе тайлы, изометрические спрайты, шаблоны-конструкторы для персонажей и построек, 3D-модели, музыку и звуки, элементы UI.

Game Developer Studio — более 100 бесплатных 2D-ассетов, которые можно отфильтровать в магазине по принципу «сначала дешевые». Автор сайта и всех материалов — Роберт Брукс. Вы можете отправлять ему идеи нового контента и голосовать за чужие предложения в разделе Suggest an asset.

Game assets на **itch.io** — золотые россыпи 2D- и 3D-графики для ваших игр. Тысячи ассетов от участников сообщества. Много красивого пиксель-арта, выразительные персонажи, детализированные тайлы карт и уровней. Никакой рекламы на страницах.

2.2 Реализация объектно–ориентированных технологий программирования в современных программно–математических средах

Объектно-ориентированное программирование (ООП) — методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определённого класса, а классы образуют иерархию наследования.

Идеологически ООП — подход к программированию как к моделированию информационных объектов, решающий на новом уровне основную задачу структурного программирования: структурирование информации с точки зрения управляемости, что существенно улучшает управляемость самим процессом моделирования, что, в свою очередь, особенно важно при реализации крупных проектов.

Управляемость для иерархических систем предполагает минимизацию избыточности данных (аналогичную нормализации) и их целостность, поэтому созданное удобно управляемым — будет и удобно пониматься. Таким образом, через тактическую задачу управляемости решается стратегическая задача — транслировать понимание задачи программистом в наиболее удобную для дальнейшего использования форму.

Основные принципы ООП: инкапсуляция, наследование, полиморфизм.

Инкапсуляция— свойство языка программирования, позволяющее пользователю не задумываться о сложности реализации используемого программного компонента а взаимодействовать с ним посредством предоставляемого интерфейса (публичных методов и членов), а также объединить и защитить жизненно важные для компонента данные. При этом пользователю предоставляется только спецификация (интерфейс) объекта. Пользователь может взаимодействовать с объектом только через этот интерфейс. Реализуется с помощью ключевого слова: public. Пользователь не может использовать закрытые данные и методы. Реализуется с помощью ключевых слов: private, protected, internal. Инкапсуляция — один из четырёх важнейших механизмов объектно-ориентированного программирования (наряду с абстракцией, полиморфизмом и наследованием). Сокрытие реализации целесообразно применять в следующих случаях: предельная локализация изменений при необходимости таких изменений, прогнозируемость изменений (какие изменения в коде надо сделать для заданного изменения функциональности) и прогнозируемость последствий изменений.

Наследование— один из четырёх важнейших механизмов объектно-ориентированного программирования (наряду с инкапсуляцией, полиморфизмом и абстракцией), позволяющий описать новый класс на основе уже существующего (родительского), при этом свойства и функциональность родительского класса заимствуются новым классом. Другими словами, класс-наследник реализует спецификацию уже существующего класса (базовый класс). Это позволяет обращаться с объектами класса-наследника точно так же, как с объектами базового класса. Простое наследование Класс, от которого произошло наследование, называется базовым или родительским (англ. base class). Классы, которые произошли от базового, называются потомками, наследниками или производными классами (англ. derived class). В некоторых языках используются абстрактные классы. Абстрактный класс — это класс, содержащий хотя бы один абстрактный метод, он описан в программе, имеет поля, методы и не может использоваться для непосредственного создания объекта. То есть от абстрактного класса можно только наследовать. Объекты создаются только на основе производных классов, наследованных от абстрактного. Например, абстрактным классом может быть базовый класс «сотрудник вуза», от которого наследуются классы «аспирант», «профессор» и т. д. Так как производные классы имеют общие поля и функции (например, поле «год рождения»), то эти члены класса могут быть описаны в базовом классе. В программе создаются объекты на основе

классов «аспирант», «профессор», но нет смысла создавать объект на основе класса «сотрудник вуза».

Полиморфизм— возможность объектов с одинаковой спецификацией иметь различную реализацию. Язык программирования поддерживает полиморфизм, если классы с одинаковой спецификацией могут иметь различную реализацию — например, реализация класса может быть изменена в процессе наследования[1]. Кратко смысл полиморфизма можно выразить фразой: «Один интерфейс, множество реализаций». Полиморфизм — один из четырёх важнейших механизмов объектно-ориентированного программирования (наряду с абстракцией, инкапсуляцией и наследованием). Полиморфизм позволяет писать более абстрактные программы и повысить коэффициент повторного использования кода. Общие свойства объектов объединяются в систему, которую могут называть по-разному — интерфейс, класс. Общность имеет внешнее и внутреннее выражение: внешняя общность проявляется как одинаковый набор методов с одинаковыми именами и сигнатурами (именем методов и типами аргументов и их количеством); внутренняя общность — одинаковая функциональность методов. Её можно описать интуитивно или выразить в виде строгих законов, правил, которым должны подчиняться методы. Возможность приписывать разную функциональность одному методу (функции, операции) называется перегрузкой метода (перегрузкой функций, перегрузкой операций).

То есть фактически речь идёт о прогрессирующей организации информации согласно первичным семантическим критериям: «важное/неважное», «ключевое/подробности», «родительское/дочернее», «единое/множественное». Прогрессирование, в частности, на последнем этапе даёт возможность перехода на следующий уровень детализации, что замыкает общий процесс.

Объект. Поля, свойства и методы объекта.

Объекты — это крупнейшее достижение в современной технологии программирования. Изобретение и практическая реализация объектов являются подвигом человеческого гения. Объекты позволили строить программу не из чудовищных по сложности процедур и функций, а из кирпичиков-объектов, заранее наделенных нужными свойствами. При этом внутренняя сложность объектов скрыта от программиста, он просто пользуется готовым строительным материалом.

Свойство — это имеющий имя атрибут объекта. Свойства определяют характеристики объекта (цвет, положение на экране, состояние объекта). **Методы** — это действия или задачи, которые выполняет объект (то, что можно делать с объектами).

Классы

Класс является типом данных, определяемым пользователем. В классе задаются свойства и поведение какого-либо предмета или процесса в виде полей данных (аналогично структуре) и функций для работы с ними. Создаваемый тип данных обладает практически теми же свойствами, что и стандартные типы. Существенным свойством класса является то, что детали его реализации скрыты от пользователей класса за интерфейсом. Интерфейсом класса являются заголовки его открытых методов. Таким образом, класс как модель объекта реального мира является черным ящиком, замкнутым по отношению к внешнему миру.

Конструкторы и их свойства

Конструктор предназначен для инициализации объекта и вызывается автоматически при его создании. Ниже перечислены основные свойства конструкторов.

1. Конструктор не возвращает значения, даже типа void. Нельзя получить указатель на конструктор;

2. Класс может иметь несколько конструкторов с разными параметрами для разных видов инициализации (при этом используется механизм перегрузки);

3. Конструктор, который можно вызвать без параметров, называется конструктором по умолчанию;

4. Параметры конструктора могут иметь любой тип, кроме этого же класса. Можно задавать значения параметров по умолчанию. Их может содержать только один из конструкторов;

5. Если программист не указал ни одного конструктора, компилятор создает его автоматически (кроме случая, когда класс содержит константы и ссылки, поскольку их необходимо инициализировать). Такой конструктор вызывает конструкторы по умолчанию для полей класса и конструкторы базовых классов;

6. Конструкторы не наследуются;

7. Конструктор не может быть константным, статическим и виртуальным (нельзя использовать модификаторы const, virtual и static) ;

8. Конструкторы глобальных объектов вызываются до вызова функции main. Локальные объекты создаются, как только становится активной область их действия. Конструктор запускается и при создании временного объекта (например, при передаче объекта из функции).

При объявлении объектов вызывается один из конструкторов. При отсутствии инициализирующего выражения в объявлении объекта вызывается конструктор по умолчанию, при инициализации другим объектом того же типа – конструктор копирования, при инициализации полей – один из явно определенных конструкторов инициализации (т.е. конструкторов, которым передаются параметры для инициализации полей объекта).

Шаблоны функций

При создании функций иногда возникают ситуации, когда две функции выполняют одинаковую обработку, но работают с разными типами данных (например, одна использует параметры типа int, а другая типа float). С помощью механизма перегрузки функций можно использовать одно и то же имя для функций, выполняющих разные действия и имеющих разные типы параметров. Однако, если функции возвращают значения разных типов, то тогда следует использовать для них уникальные имена.

Шаблон определяет набор операторов, с помощью которых ваши программы позже могут создать несколько функций. Программы часто используют шаблоны функций для быстрого определения нескольких функций, которые с помощью одинаковых операторов работают с параметрами разных типов или имеют разные типы возвращаемых значений. Шаблоны функций имеют специфичные имена, которые соответствуют имени функции, используемому вами в программе. После того как программа определит шаблон функции, она в дальнейшем может создать конкретную функцию, используя этот шаблон для задания прототипа, который

включает имя данного шаблона, возвращаемое функцией значение и типы параметров.

Шаблоны классов

Шаблон класса позволяет задать класс, параметризованный типом данных. Передача классу различных типов данных в качестве параметра создает семейство родственных классов. Наиболее широкое применение шаблоны находят при создании контейнерных классов. Контейнерным называется класс, который предназначен для хранения каким-либо образом организованных данных и работы с ними. Преимущество использования шаблонов состоит в том, что как только алгоритм работы с данными определен и отлажен, он может применяться к любым типам данных без переписывания кода.

3 3. Инструментарий

При реализации курсовой работы я использовал IDE Microsoft Visual Studio 2019.

IDE(Integrated Drive Electronics или же интегрированная среда разработки) – комплекс программных средств, используемый программистами для разработки программного обеспечения. Среда разработки включает в себя:

текстовый редактор
компилятор
отладчик и прочее

Суть IDE заключается в объединении нескольких компонентов для максимальной концентрации программиста на решении алгоритмических задач, без потерь времени на сохранение файла в текстовом редакторе, затем вызове компилятора и так далее. Таким образом, повышается производительность труда разработчика. Также огромным преимуществом среды разработки является наличие статического анализатора кода, который позволяет выявить ошибки в синтаксисе и другие мелкие ошибки по ходу написания программы.

Microsoft Visual Studio 2019 – это среда разработки от компании Microsoft. Она включает в себя поддержку многих компонентов таких как: Visual Basic, Visual C++, Visual C#, Python, .NET и её компоненты, имеет встроенный веб–сервер который может пригодиться при веб–разработке, присутствует интеграция с Unity и Unreal Engine. Можно разрабатывать как консольные приложения, так и приложения с графическим интерфейсом. Visual Studio позволяет создавать и подключать сторонние дополнения для расширения функциональности при помощи NuGet, также присутствует интеграция с системой контроля версий Git.

Использование системы контроля версий GIT

Git — распределённая система управления версиями(СУВ). Среди проектов, использующих Git — ядро Linux, Swift, Android, Drupal, Cairo, GNU Core Utilities, Mesa, Wine, Chromium, Compiz Fusion, FlightGear, jQuery, PHP, NASM, MediaWiki, Doku Wiki, Qt. Основное отличие Git От любой другой системы контроля версий это подход к работе с данными. Большинство СУВ хранят информацию в виде списка изменений в файле. Эти системы представляют хранимую информацию в виде набора файлов и изменений, сделанных в каждом файле, по времени. Git же в свою очередь хранит информацию в виде ссылок на файлы

предыдущих версий, с возможностью доступа к ним. Также к преимуществам Git можно отнести то, что для работы большинства операций ему достаточно локальных файлов и ресурсов.

В системах контроля версий существует ветвление. Оно позволяет отделяться от основной линии разработки и при этом продолжать работу. В большинстве СУВ эта функция реализована но при этом это достаточно ресурсозатратный процесс поскольку приходится создавать новую копию директории что может занимать значительное время при работе с большими проектами. В Git же этот процесс выполняется гораздо быстрее из–за того что файлы хранятся локально.

Для работы с Git необходимо знать ряд базовых команд. В эти команды входит: git add, git commit, gid status, git rm, git mv,. Функция git add добавляет содержимое рабочей директории в индекс для последующего коммита. Git commit берёт все данные, добавленные в индекс с помощью `git add`, и сохраняет их слепок во внутренней базе данных, а затем сдвигает указатель текущей ветки на этот слепок. Команда `git status` показывает состояния файлов в рабочей директории и индексе: какие файлы изменены, но не добавлены в индекс, какие ожидают коммита в индексе. Вдобавок к этому выводятся подсказки о том, как изменить состояние файлов. Команда git rm это команда, обратная команде git add. То есть она удаляет файлы из текущей директории или индекса. Git mv или же git move это простое перемещение файлов между директориями.

Существует отдельный блок команд для создания и перемещения между ветками. Для создания ветки используется команда git branch. Команда git checkout создает ветку и переходит на неё. Если ветка с заданным именем уже существует, эта команды позволят перейти на неё.

Ветка в Git — это простой перемещаемый указатель на один из ваших коммитов. По умолчанию, имя основной ветки в Git — `master`. Как только вы начнёте создавать коммиты, ветка `master` будет всегда указывать на последний коммит в основной ветке. Каждый раз при создании коммита указатель ветки `master` будет передвигаться на следующий коммит автоматически. Так же существует ещё один указатель – HEAD. Это указатель на текущее местоположение. При создании новой ветки указатель master останется на основной ветке, а HEAD перейдет на новую.

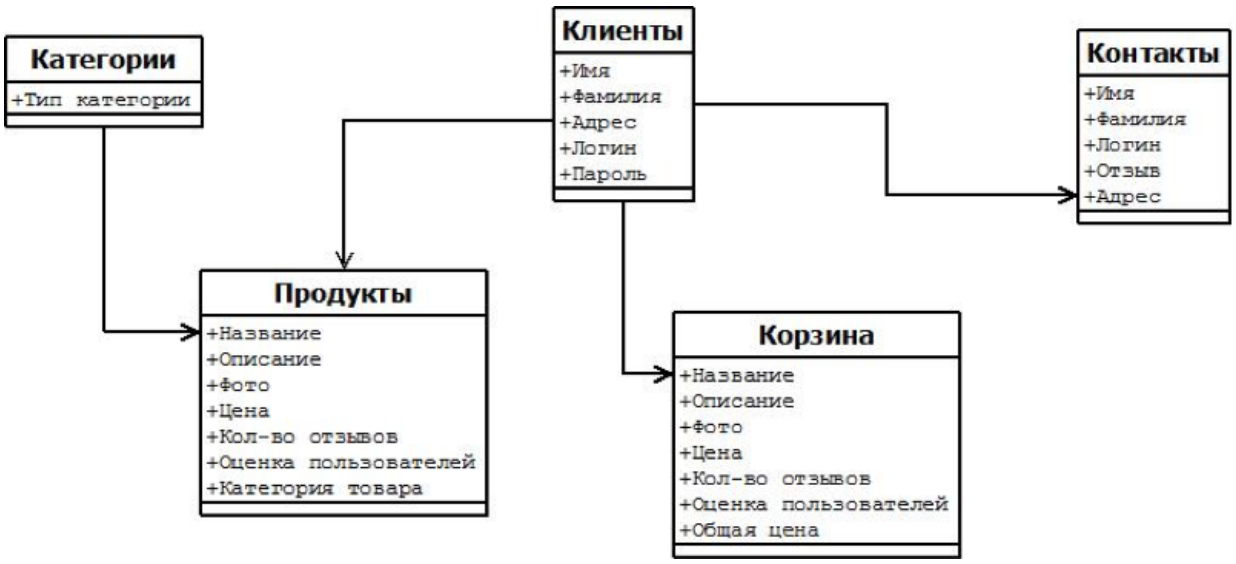


Рисунок 3.1 – Ветвление

При работе над проектом более одного человека необходимо взаимодействовать и обмениваться кодом, в этом может помочь GitHub.

GitHub — сервис онлайн-хостинга репозиториев, обладающий всеми функциями распределённого контроля версий и функциональностью управления исходным кодом — всё, что поддерживает Git и даже больше. Обычно он используется вместе с Git и даёт разработчикам возможность сохранять их код онлайн, а затем взаимодействовать с другими разработчиками в разных проектах. Он имеет полную совместимость с Git, что позволяет загружать обновление на сайт из интерфейса командной строки Git.

Ссылка на Git репозиторий с моим проектом:

https://github.com/foxwe11/Infinite-level

4 4. Движок Unity 3D

Интерфейс Unity, как и интерфейс многих современных приложений, имеет настраиваемое расположение окон (window layout), их расположение можно настраивать, перетаскивая за закладки. Если перетащить закладку в область закладок уже существующего окна, то она будет добавлена к присутствующим там закладкам.Также можно прикрепить окно к краю экрана или краю другого окна.

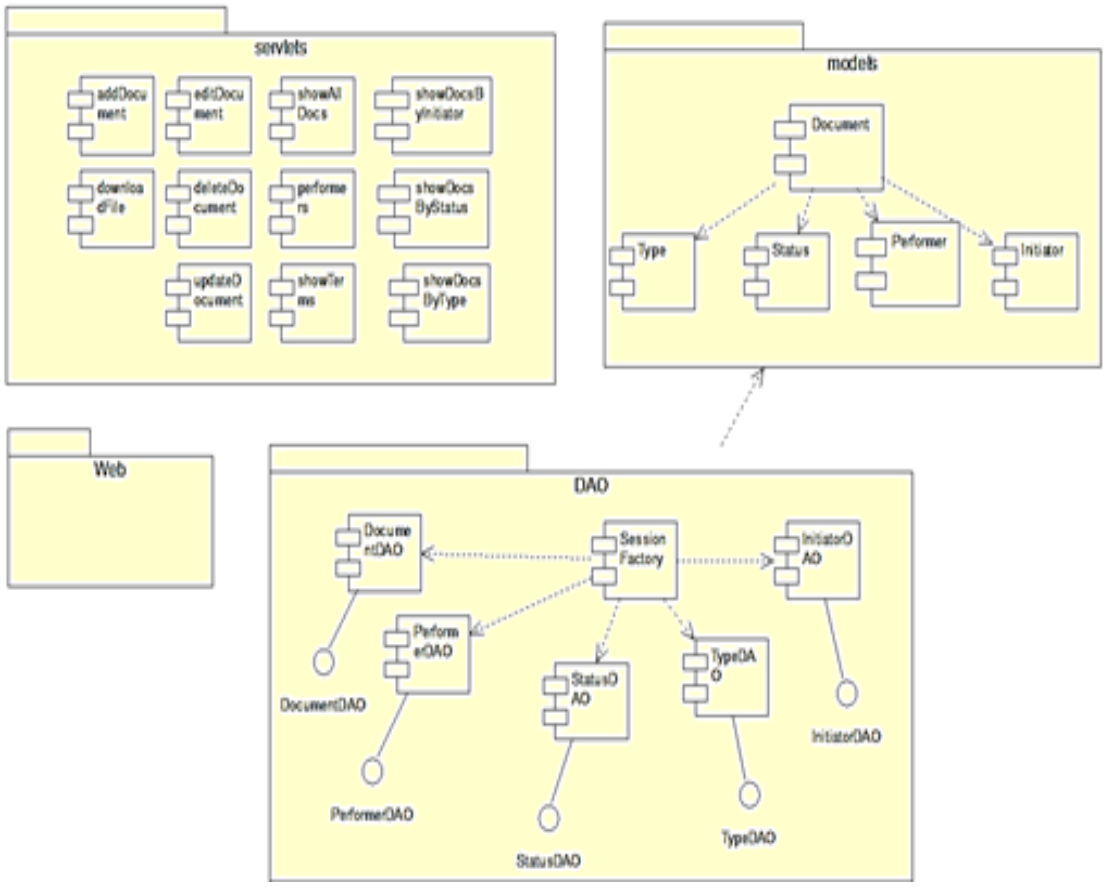


Рисунок 4.1 – Интерфейс Unity

Как показано на рисунке 4.1 (для PC версии), есть пять различных элементов, с которыми вы будите иметь дело:

• Scene (сцена) [1] — где создается игра;

• Hierarchy (иерархия) [2] — список GameObjects на сцене;
• Inspector (инспектор) [3] — настройки выбранного ресурса/объекта (asset/object);
• Game (игра) [4] — окно предварительного просмотра;
• Project (проект) [5] — список ресурсов проекта.

Окно Сцены и Иерархия.

Окно Сцены (Scene window) – это область, в которой вы будите создавать свою игру в Unity. В верхнем правом углу окна располагается Scene Gizmo. Он обозначает положение камеры сцены и позволяет быстро изменять ракурс обзора. Нажатие на ручки Scene Gizmo перемещает камеру в соответствующую позицию (вид сверху, слева, спереди и т. д.) или включает изометрический режим (Isometric Mode). Перемещение ресурсов в это окно, сделает их активным объектами игры. Scene (сцена) связана с Hierarchy (иерархией), в которой перечислены все активные объекты в открытой сцене в алфавитном порядке по возрастанию.

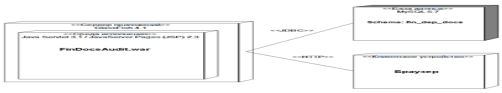


Рисунок 4.2 – Кнопки управления сценой

Окно сцены сопровождается четырьмя кнопками управления, показанных на рисунке 4.2. Их дублируют горячие клавиши Q, W, E и R.

• The Hand tool (рука) [Q]: Этот инструмент позволяет осуществлять навигацию в окне сцены. Сама по себе она (рука) служит для линейного перемещения вверх- вниз, вправо-влево, с зажатой клавишей Alt – позволяет вращать камеру, а с Ctrl – приближать-удалять. Если при этом зажать клавишу Shift скорость перемещения увеличиться.
• The Translate tool (перемещение) [W]: Позволяет изменять позицию выбранного объекта или группы объектов используя «ручки» гизмо.
• The Rotate tool (поворот) [E]: схож и инструментом Translate, но этот инструмент позволяет менять угол наклона объекта(ов).
• The Scale tool (масштаб) [R]: опят же, схож с инструментами Translate и Rotate, но изменяет размер, масштаб.

Выбрав объект или на Scene или в Hierarchy, он сразу же станет выбранным в обоих, свойства этого объекта отобразятся в Inspector (инспекторе (свойств)). Для фокусировки камеры на объекте выборном его в Hierarchy, наведите курсор на окно сцены и нажмите клавишу F, окно сцены будет центрировано и масштабировано по объекту.

Инспектор (свойств).

Думайте об Inspector (инспекторе), как о своем личном наборе инструментов, для настройки каждого элемента какого-нибудь объекта игры или ресурсов в вашем проекте. Подобные инспекторы свойств (Property Inspector), есть во многих приложениях. Следует знать, что это контекстно-зависимая окно, т.е. не зависимо, какой объект или ресурс вы выберете в текущий момент, Inspector сразу отобразит соответствующие свойства — он чувствителен к контексту, с которым вы работаете. Inspector покажет каждый компонент объекта, который вы выбрали, и позволит вам изменять переменные этих компонентов, используя простые формы интерфейса, такие как текстовые окна ввода, ползунки, кнопки и выпадающие списки, так же поддерживается система drag-and-drop , где это необходимо. Любые свойства, отображаемые

в Inspector, могут быть изменены, там же это относится и к переменным скриптов, прикреплённых к объекту. Можно использовать Inspector для изменения переменных во время предпросмотра для экспериментов и поиска магического баланса игры.

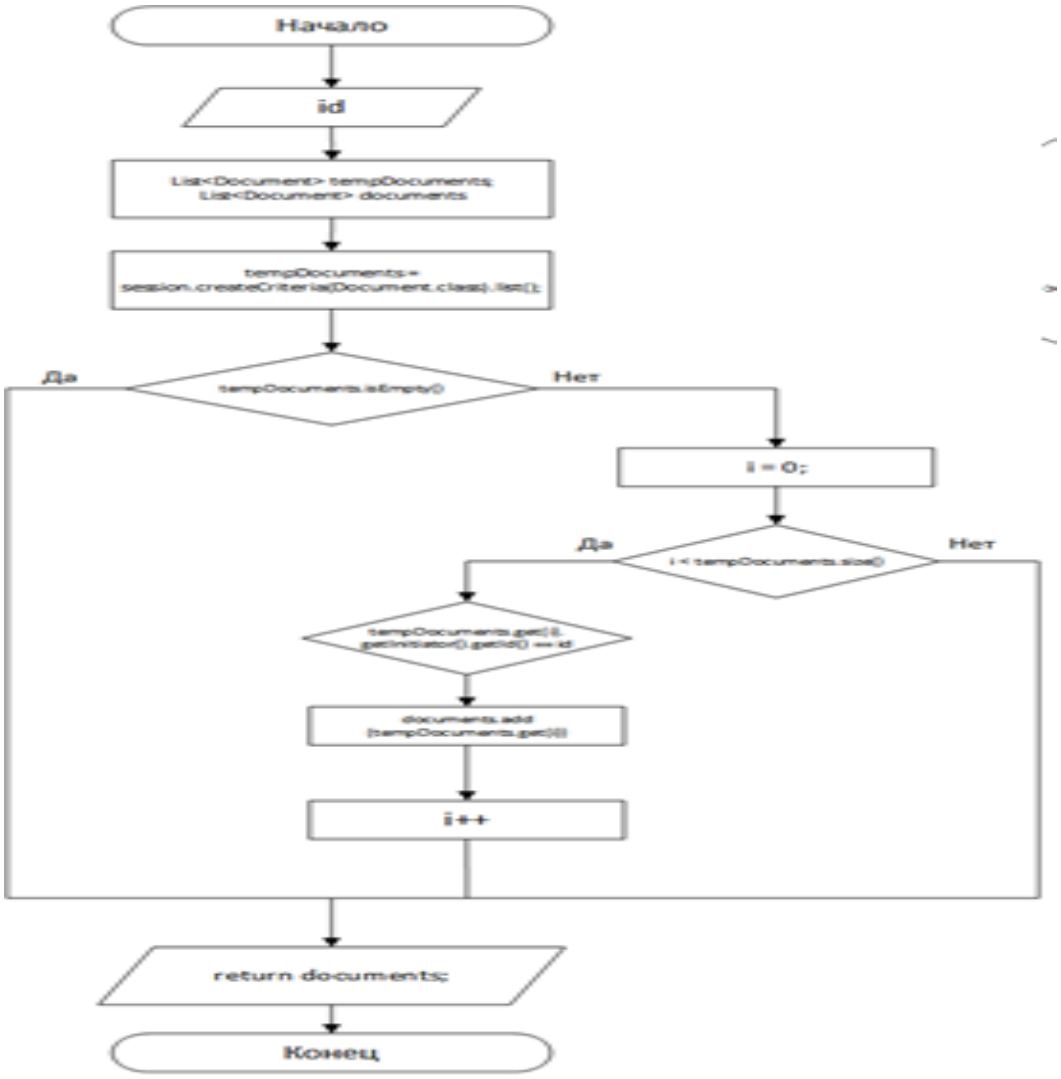


Рисунок 4.3 - Inspector

На рисунке 4.3, Inspector отображает свойства выбранного игрового объекта. Сам объект включает в себя два компонента: Transform (преобразование) и Animation (анимация). Инспектор позволит вам внести изменения в настройки любого из них. Отметим также, что возможно временно отключить любой компонент в любое время, что очень полезно для тестирования и экспериментов - вы можете просто снять флажок слева от названия компонента. Кроме того, если вы хотите отключить весь объект, то вы можете снять флажок рядом с его именем в верхней части окна инспектора.

Окно Проекта.

Каждый проект содержит папку Assets, содержимое этой папки представлено в окне Project. Помещая ресурсов(assets) в папку Assets вы сразу же сможете увидеть их в окне проекта, и они будут автоматически импортированы в ваш проект Unity. Кроме того, изменение какого либо ресурса, находящегося в папке Assets, с помощью сторонних программ, например, в Photoshop-е, заставит Unity заново импортировать (reimport) его, и отразить эти изменения в вашем проекте и любых активных сценах, которые используют этот ресурс. В окне проекта (Project

window) есть кнопка Create, которая позволит создать любой ресурс доступный в рамках Unity, например, скрипт, префаб или материал. Важно помнить, при перемещении ресурсов внешними инструментами (например, Проводником) будут потеряны все метаданные, связанные с ним. Метаданные хранят информацию о ресурсе и его связях с другими ресурсами. Всегда перемещайте ресурсы только в окно Project-а (перетащить файл из Проводника в окно проекта.).

Окно Игры.

Game window (окно игры) вызывается нажатием на кнопку Play и выступает в качестве практического испытания вашей игры. Нажав на Play, крайне важно, иметь в виду следующие рекомендации: В режиме воспроизведения (play mode), изменения, вносимые в любую часть вашей игровой сцены, временные – это задумано, как способ тестирования, при остановке игры, все внесенные изменения будут отменены. Это часто сбивать с толку новых пользователей, так что не забывайте об этом! Имеется ряд настроек режима воспроизведения (play mode), через меню окна Game. Первое выпадающее меню в окне Game — это контроль пропорций изображения (Aspect Drop-down). На некоторых дисплеях это соотношение отличается от стандартного 4:3 (например, на широкоформатных мониторах — 16:10). Далее идёт кнопка Maximize on Play. Если она нажата, то окно Game растягивается на всё окна редактора при предпросмотре. Кнопка Gizmos включает отображение контейнеров гизмо в окне Game . Последняя кнопка — Stats. Она показывает статистку рендеринга (Rendering Statistics), полезную при оптимизации игры.

5 5. Шаблоны (паттерны) проектирования практических задач

Паттерн проектирования — это часто встречающееся решение определённой проблемы при проектировании архитектуры программ.

В отличие от готовых функций или библиотек, паттерн нельзя просто взять и скопировать в программу. Паттерн представляет собой не какой-то конкретный код, а общую концепцию решения той или иной проблемы, которую нужно будет ещё подстроить под нужды вашей программы.

Паттерны часто путают с алгоритмами, ведь оба понятия описывают типовые решения каких-то известных проблем. Но если алгоритм — это чёткий набор действий, то паттерн — это высокоуровневое описание решения, реализация которого может отличаться в двух разных программах.

Если привести аналогии, то алгоритм — это кулинарный рецепт с чёткими шагами, а паттерн — инженерный чертёж, на котором нарисовано решение, но не конкретные шаги его реализации.

Описания паттернов обычно очень формальны и чаще всего состоят из таких пунктов:

проблема, которую решает паттерн;
мотивации к решению проблемы способом, который предлагает паттерн;
структуры классов, составляющих решение;
примера на одном из языков программирования;

особенностей реализации в различных контекстах;
связей с другими паттернами.

Такой формализм в описании позволил создать обширный каталог паттернов, проверив каждый из них на состоятельность.

Существует три вида шаблонов:

Поведенческие
Порождающие
Структурные

Поведенческие шаблоны – это паттерны, которые решают задачи эффективного и безопасного взаимодействия между объектами программы. Поведенческие паттерны:

цепочка обязанностей – который позволяет передавать запросы последовательно по цепочке обработчиков. Каждый последующий обработчик решает, может ли он обработать запрос сам и стоит ли передавать запрос дальше по цепи
команда – превращает запросы в объекты, позволяя передавать их как аргументы при вызове методов, ставить запросы в очередь, логировать их, а также поддерживать отмену операций
итератор – даёт возможность последовательно обходить элементы составных объектов, не раскрывая их внутреннего представления
посредник – позволяет уменьшить связанность множества классов между собой, благодаря перемещению этих связей в один класс–посредник
снимок – позволяет сохранять и восстанавливать прошлые состояния объектов, не раскрывая подробностей их реализации
наблюдатель – создаёт механизм подписки, позволяющий одним объектам следить и реагировать на события, происходящие в других объектах
состояние – позволяет объектам менять поведение в зависимости от своего состояния
стратегия – определяет семейство схожих алгоритмов и помещает каждый из них в собственный класс, после чего алгоритмы можно взаимозаменять прямо во время исполнения программы
шаблонный метод – определяет скелет алгоритма, перекладывая ответственность за некоторые его шаги на подклассы
посетитель – позволяет добавлять в программу новые операции, не изменяя классы объектов, над которыми эти операции могут выполняться

Порождающие шаблоны – шаблоны проектирования, которые отвечают за удобное и безопасное создание новых объектов или даже целых семейств объектов. Существуют следующие порождающие шаблоны:

фабричный метод – определяет общий интерфейс для создания объектов в суперклассе, позволяя подклассам изменять тип создаваемых объектов
абстрактная фабрика – позволяет создавать семейства связанных объектов, не привязываясь к конкретным классам создаваемых объектов
строитель – позволяет создавать сложные объекты пошагово
прототип – позволяет копировать объекты, не вдаваясь в подробности их реализации
одиночка – гарантирует, что у класса есть только один экземпляр, и предоставляет к нему глобальную точку доступа

Структурные шаблоны – это паттерны, которые отвечают за построение удобных в

поддержке иерархий классов. Структурные шаблоны:

адаптер – позволяет объектам с несовместимыми интерфейсами работать вместе
мост – разделяет один или несколько классов на две отдельные иерархии – абстракцию и реализацию, позволяет изменять их независимо друг от друга
компоновщик – позволяет сгруппировать множество объектов в древовидную структуру, а затем работать с ней так, будто это единичный объект
декоратор – позволяет динамически добавлять объектам новую функциональность, оборачивая их в полезные ”обёртки”
фасад – предоставляет простой интерфейс к сложной системе классов, библиотеке или фреймворку
легковес – позволяет вместить большее количество объектов в отведённую оперативную память
заместитель – позволяет подставлять вместо реальных объектов специальные объекты–заменители

Использование шаблонов помогает избежать многих ошибок и позволяет сократить время, используя проверенные решения. Также при работе в команде с другими разработчиками не обязательно объяснять, что вы сделали, а можно сказать название шаблона.

Каждый из шаблонов решает конкретную проблему, поэтому прежде чем использовать какой-либо шаблон, нужно убедиться, что он подходит под решение вашей задачи.

Заключение

В процессе выполнения курсового проекта были получены навыки работы с игровым движком Unity. Познакомился и изучил язык программирования C#. В ходе проектирования курсового проекта были получены знания из области игровой индустрии, в частности удалось примерить на себя роль геймдизайнера. В результате была разработана мобильная игра жанра головоломка – “Infinite level”. В ходе разработки мобильной игры были получены практические знания об использовании методик объектно-ориентированного программирования. Убедившись в работоспособности мобильной игры, можно сказать, что поставленная цель была достигнута.

Список использованных источников

1. [печатное издание] **Разработка мобильной игры для ОС Android на платформе Unity** https://dspace.susu.ru/xmlui/bitstream/handle/0001.74/16571/2017_402_kazantsevsa.pdf?sequence=1
2. [url] **2. Технология объектно-ориентированного программирования** https://studfile.net/preview/5430382/page:4/
3. [url] **Создание концепции игры** https://stopgame.ru/blogs/topic/66687
4. [url] **Уроки Unity3d - Интерфейс** http://gamesmaker.ru/3d-game-engines/unity3d/interface/
5. [url] **Паттерны проектирования** https://refactoring.guru/ru/design-patterns
6. [url] **Бесплатные ресурсы для ваших игр** https://geekbrains.ru/posts/gamedev_resources

Приложения

1. [электронный документ] 5ebd66884ae9e_algoritm.docx
2. [электронный документ] 5ebd668d2fd8d_Sostoyanie.docx
3. [Задание] **Задание** 5ebd681ecc663_Kursovaya rabota oop.docx
4. [электронный документ] 5ec2e444e812e_4.png

5. [электронный документ] 5ec2f3b74f289_Skriny.docx
6. [электронный документ] 5ec2f3ca38e8c_Listing_koda.docx