Детектор Плагиата v. 2215 - Отчёт оригинальности: 13.06.2024 10:21:19

Проанализированный документ: ПЗ_Бабенко (1).docx Лицензия: ВОЛОДИМИР МАТІЄВСЬКИЙ

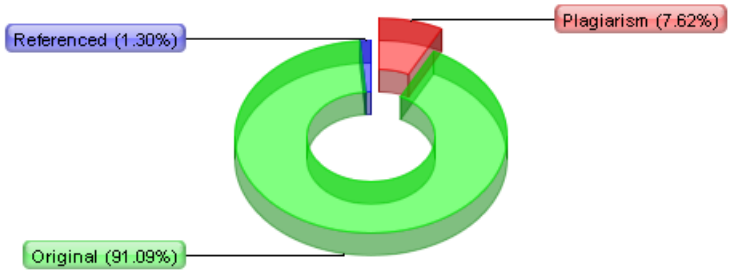
Тип поиска: Поиск переписанного Язык: Uk

Тип проверки: Интернет

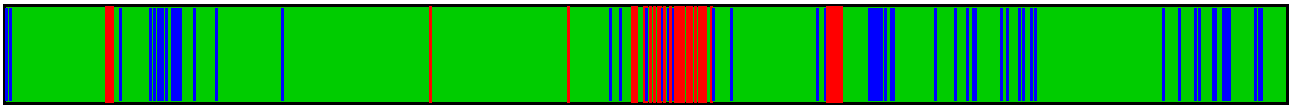
ТЕЕ и кодировка: DocX n/a

Детальный анализ тела документа:

Диаграмма соотношения частей:



Граф распределения зон:



Источники плагиата: 17

5%	606	1.	https://yak.koshachek.com/articles/2-3-klasifikacija-sutnostej-tri-klasi-sutnostej.html
1%	178	2.	https://uk.wikipedia.org/wiki/SQLite
1%	172	3.	https://kievoit.ippo.kubg.edu.ua/kievoit/2016/58_CS_SQLite/index.html

Детали обработанных ресурсов: 212 - OK / 6 - Ошибок

Важные замечания:

Википедия:	Google Книги:	Сервисы платных работ:	Античит:
Обнаружена Wiki!	[не обнаружено]	[не обнаружено]	**Обнаружено сокрытие!**

Античит-отчет UACE:

1. Статус: Анализатор **Включен** Нормализатор **Включен** сходство символов установлено на **100%**
2. Обнаруженный процент загрязнения UniCode: **16,4%** с лимитом: 4%
3. Процент нераспознанных символов после нормализации: **9,2%**
4. Все подозрительные символы будут отмечены фиолетовым цветом: Abcd...
5. Найдены невидимые символы: 0

Рекомендации по оценке:

Особое внимание следует уделить анализу этого отчета! Предполагается, что этот документ содержит значительное количество символов, чуждых языку документа. Это прямое указание на то, что автор документа использовал специальное программное обеспечение\онлайн-веб-сервис, чтобы эффективно скрыть текст в попытке избежать обнаружения потенциального плагиата. Настоятельно рекомендуется передать это дело на более высокий уровень! В случае сомнений обращайтесь: в службу поддержки Детектора плагиата!

Алфавитная статистика и анализ символов:

Активные ссылки (URL-адреса, извлеченные из документа):

URL не найдены

Исключённые ресурсы:

URL не найдены

Включённые ресурсы:

URL не найдены

Детальный анализ документа:

Міністерство освіти і науки України ДЗ

Цитирования: **0,04%** id: **1**

«луганський Національний університет імені тараса шевченка»

Навчально-науковий інститут математики та інформаційних технологій (назва факультету, інституту) Інформаційних технологій та систем (назва кафедри) Пояснювальна записка до дипломного проєкту (роботи) БАКАЛАВРА (освітньо-кваліфікаційний рівень) на тему: АНАЛІЗ ТА РОЗРОБКА ІГРОВОГО ПРОЄКТУ НА ОСНОВІ UNITY3D Виконав: студент 4 курсу напряму підготовки (спеціальності) 121

Цитирования: **0,02%** id: **2**

«Інженерія програмного забезпечення»_

_____ (шифр і назва напряму підготовки, спеціальності) Бабенко В. В. (прізвище та ініціали) Керівник ___ Донченко В. Ю. (прізвище та ініціали) Рецензент _____ Козуб Ю. Г. (прізвище та ініціали) Полтава – 2024 року ЗМІСТ ВСТУП4 РОЗДІЛ 1. ПОСТАНОВКА ЗАДАЧІ ТА АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ7 1.1. Класифікація комп’ютерних ігор7 1.2. Порівняльна характеристика ігрових движків11 1.2.1. Unity12 1.2.2. Torque 2D/3D16 1.2.3. CryEngine 318 1.2.4. UDK19 1.2.5. Frostbite Engine21 1.3. Функціональні можливості Unity 3D23 1.4. Установка Unity і короткий опис можливостей26 1.5. Інтерфейс редактора Unity28 1.6. Висновки до розділу32 РОЗДІЛ 2. РОЗРОБКА АРХІТЕКТУРИ ІГРОВОГО ПРОЄКТУ33 2.1. Основні модулі ігрового додатку33 2.2. Ігровий цикл34 2.3. Система меню36 2.4. Висновки до розділу39 РОЗДІЛ 3. ОСНОВНІ ЕТАПИ РЕАЛІЗАЦІЇ ІГРОВОГО ПРОЄКТУ40 3.1. Проєктування інфологічної моделі бази даних40 3.2. Проєктування даталогічної моделі бази даних44 3.3. Процес створення ігрового додатку52 3.3.1. Створення проєкту52 3.3.2. Налаштування редактора - роздільна здатність ігри, зміна платформи53 3.3.3. Створення головного меню ігрової програми54 3.3.4. Створення контролерів59 3.3.5. Створення музичного контролера64 3.4. Створення карти67 3.5. Висновки до розділу68 ВИСНОВКИ69 СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ70 ДОДАТКИ74 Додаток 1. Даталогічна модель бази даних74 Додаток 2. Лістинг програми75 ВСТУП На сьогоднішній день мобільні та ігрові технології досягли того рівня, що при використанні смартфонів, планшетів, всіляких гаджетів, стали невід'ємною частиною сучасного суспільства. Не можна залишити без уваги ігрову індустрію. Для створення ігор використовується безліч різних технологій, мов, додатків і платформ. Кожна з них має свої плюси і мінуси. Без стільникового зв'язку сучасне людство не може уявити собі й дня. І буквально зовсім недавно головними функціями були тільки дзвінки та відправка повідомлень. Однак сьогодні пріоритети дещо змінюються. Телефон став далеко не просто засобом для комунікації людей, але і багатофункціональним пристроєм, що дозволяє пізнавати світ, проводити дозвілля і навіть заробляти гроші. І, звичайно, все це стало можливо тільки завдяки розвитку Інтернету для мобільних пристроїв і створених з цієї нагоди додатків. У XXI столітті почався бурхливий розвиток ринку мобільного контенту, зокрема, мобільних додатків. Здешевити мобільний інтернет трафік дозволило поява нових технологій - GPRS і EDGE. Користувачі починають викачувати із Всесвітньої мережі величезну кількість картинок, ігор, рингтонів і пр.[25] Мобільні ігри набули небувалої популярності за останні роки та посіли почесне місце на ринку розваг та дозвілля. Для задоволення потреб розробників було створено величезну кількості інструментальних засобів та продовжується розробка нових. Загальна ціль всіх інструментальних засобів - полегшення та покращення розробки, використання передових технологій обробки графіки, фізики, забезпечення кросплатформенності розроблених проєктів. Додаток Unity являє собою професійний ігровий двигун, який використовується в створенні відеоігор для різних платформ. Це інструмент, яким щодня користуються досвідчені розробники, а також один із доступніших інструментів для новачків. До недавнього часу людина, яка вирішила навчитися програмуванню ігор (особливо тривимірних), відразу ж стикався з безліччю серйозних перешкод, в той час як інструмент Unity дозволив значно полегшити життя новачкам [1]. Створення відеоігор в своїй основі не відрізняється від написання будь-якого іншого ПО. Здебільшого відмінності виявляються в кількісній площині. Гра більш інтерактивна, ніж більшість веб-сайтів, а отже, потрібно зовсім інший тип коду, але при цьому в обох випадках будуть задіяні подібні навички і процеси [1]. Будь-який ігровий двигун надає безліч функціональних можливостей, які застосовуються в різних іграх. Реалізована на цьому двигуну гра отримує всі ці функціональні можливості, крім того, додаються її власні ігрові ресурси і код ігрового сценарію. Unity пропонує моделювання

фізичних середовищ, карти нормалей, загородження навколишнього світу в екранному просторі (Screen Space Ambient Occlusion, SSAO), динамічні тіні, тощо [1]. Подібний набір функціональних можливостей є в багатьох ігрових двигунів, але у Unity є дві основні переваги перед іншими передовими інструментами розробки ігор: продуктивний візуальний робочий процес і потужна міжплатформенна підтримка [1]. Візуальний робочий процес виділяє даний інструмент з-посеред інших середовищ розробки ігор. Частина інструментів розробки ігор часто являють собою сукупність розрізнених частин, які потрібно контролювати, чи бібліотеку, для роботи з якою потрібно налаштовувати власне інтегроване середовище розробки, процес збирання проєкту, тощо. Робочий процес в Unity прив'язаний до ретельно продуманого візуального редактору. Редактор надає можливість компонувати сцени майбутньої гри, пов'язуючи ігрові ресурси і код в інтерактивні об'єкти. Він дозволяє швидко і раціонально створювати професійні гри, забезпечуючи високу продуктивність праці розробників і надаючи в їх розпорядження вичерпний перелік сучасних технологій в області відеоігор. Редактор дозволяє редагувати об'єкти в редакторі і рухати елементи в сцені при запущеній грі. Unity надає можливість налаштовувати редактор за допомогою сценаріїв, що додають нові функціональні можливості і елементи меню до інтерфейсу [1]. Unity підтримує наступні платформи: Windows, OSX, Linux, Android, WebGL, iOS, BlackBerry, Tizen, Xbox One, Xbox 360, PS3, PS4, PS Vista, Windows Store, Samsung TV [1]. Об’єкт дослідження – ігровий рушій UNITY3D. Предмет дослідження - розробка ігрового проєкту на основі UNITY3D. Мета роботи - аналіз та розробка ігрового проєкту на основі UNITY3D. Відповідно до мети роботи необхідно вирішити наступні завдання: Проаналізувати актуальні підходи до створення відеогри; Виконати аналіз сучасних засобів розробки комп’ютерних ігор та досвіду їх застосування. Зробити порівняльний аналіз актуальних ігрових рушіїв та їх функціоналу. Разробити ігровий проєкт на основі UNITY3D Об’єкт дослідження – ігровий рушій UNITY3D. Предмет дослідження - розробка ігрового проєкту на основі UNITY3D. У першому розділі проведено огляд та аналіз популярних сучасних засобів для розробки ігор та їх представників, розбиття засобів на класи за їх базовими заявленими функціональними можливостями. Визначено оптимальні сфери використання для інструментальних засобів спираючись на аналіз процесу розробки прототипів. Приводиться порівняльна характеристика ігрових движків (Unity, Torque 2D/3D, CryEngine 3, UDK, Frostbite Engine). Проведено аналіз функціональних можливостей Unity 3D. У другому розділі в результаті узагальнення всієї наявної інформації по роботі з Unity 3D, була розроблена архітектура ігрового проєкту. Описано основні модулі ігрового додатку та ігровий цикл. В третьому розділі описано основні етапи реалізації ігрового проєкту. РОЗДІЛ 1.
ПОСТАНОВКА ЗАДАЧІ ТА АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ 1.1. Класифікація комп’ютерних ігор Зараз розроблено безліч різноманітних комп'ютерних ігор, а їх творці випускають у світ все нові. Так як створення комп'ютерних ігор ставиться до підтримки сфери розваг, то їх класифікація не так проста, як може здатися на перший погляд. Жанри ігор формувалися без структурно і інтуїтивно протягом тривалого часу.

Обнаружен Плагиат: **0,65%** https://card-file.ontu.edu.ua/bitstream... id: **3**

Розробники ігор проводили сміливі експерименти, створюючи нові ігрові механіки. Невдалі експерименти забувалися, а вдалі ігри ставали прикладом для інших розробників. Розробники копіювали популярну ігрову механіку, додаючи трохи ідей від себе, таким чином навколо найбільш популярних ігор утворювалися цілі класи схожих між собою ігор, ці класи і стали називати ігровими жанрами. Поділ на жанри дуже корисно на практиці. Ігри певного жанру цікаві вже сформувалася ігровий аудиторії. Розробник оголошує, що випускає гру в такому-то жанрі, і гравці вже приблизно уявляють собі, що буде відбуватися в грі, навіть без уточнюючих коментарів розробників.

Часто фірма-розробник однозначно не може вказати жанр нової гри, так як має намір продати цей програмний продукт більшій кількості кінцевих користувачів, що не звужуючи тематикою коло потенційних покупців. Зазвичай ігри в основному класифікуються за жанрами, а також за кількістю гравців. Вони поділяються на кілька типів: квести, екшн, рольові ігри (рпг), файтинги, стратегії, симулятори, логічні та азартні та інші (

Цитирования: **0,02%** id: **4**

«рис.1.1»

). Рис. 1.1. Типи ігор Наразі оглянемо найпопулярніші жанри ігор: Квести - здійснюють подорож одного або декількох персонажів до поставленої мети шляхом подолання різноманітних труднощів. Екшн (action) - ігри від першої особи - популярні бродилки-

стрілялки. Рольові ігри або РПГ - гравець виконує роль певного персонажа і виконує поставлені перед ним завдання. Стратегії та логічні ігри наслідують діяльності управлінця. Симулятор імітують керування автомобілем, космічним кораблем, літаком і т.п. Азартні і логічні добре розвивають розумову діяльність. Існуючі платні і безкоштовні ігри, в яких є персонаж, різноманітні: це екшн, РПГ, деякі види стратегій, квести і т.д. Ігри без персонажа - логічні, симулятори. Вони не мають сюжету, не викликають сильного звикання і не впливають на психіку. Часто пропонують інтелектуальні завдання, що добре розвиває мислення. Стратегії - ігри з відсутністю персонажа, де процесом гри управляє людина від свого обличчя. Відмінною особливістю стратегії є особлива система внутрішньоігрових факторів. Спочатку гравець приймає певні рішення і спостерігає за їх наслідками, а потім вибудовує логіку подальших дій. Найвідоміші ігри, в яких є право вибору - це РПГ, іноді екшн і квести. Гравець самостійно вибирає персонаж, одягає його, наділяє здібностями. У грі передбачена свобода дій, яка впливає на процес і результат гри. Даний тип гри викликає стійке звикання, гравець ототожнює себе з персонажем, розвиває його за своїм образом і подобою. Ігровий процес сприймається як реальність. Багатьом РПГ - гри нагадують фантастичні книги з можливістю управління головним героєм, що становить особливий інтерес для читаючих ігроманів. Розумне захоплення подібними іграми схоже на пристрасть до читання книг, воно не шкідливо, навпаки, розвиває фантазію, захоплює пізнанням історії, відображає реальність. Ігри з відсутністю вибору мають лінійний сюжет (стрілялки). Гравець ототожнює себе з готовим персонажем, вже наділеним характеристиками. Розгалуження сюжету гри залежать не від вибору гравця, а від успіху чи випадковості. В деякі стрілялки можна грати командою. Такі ігри розряджають людини емоційно, а успішне проходження часто залежить від швидкої реакції гравця. На стику рольових ігор і стратегій знаходиться грань

Цитирования: **0,01%** id: **5**

«абстракція».

Це повна протилежність

Цитирования: **0,01%** id: **6**

«бойовикам»,

які й борються зі

Цитирования: **0,01%** id: **7**

«скучность»

за допомогою видовищності. Для ігор цієї грані характерна схематичність, обмеження волі, відсутність реалістичності, і відсутність видовищності. З боку стратегій в абстракціях головним стає виконання

Цитирования: **0,01%** id: **8**

«цілі».

З боку рольових ігор -

Цитирования: **0,01%** id: **9**

«підпорядкування».

У підсумку виходить підпорядкування чужим цілям, тобто виконання нудної роботи. При цьому можуть відбуватися абсолютно нові події, ніколи раніше не траплялися, але під час цих подій все одно доведеться підкоряться чужій волі (це головна відмінність від іншого елемента -

Цитирования: **0,01%** id: **10**

«буденності»

). Центральна точка елемента - це рабство і беспрікословно виконання наказів. Таке не має з іграми нічого спільного. Змістовний склад палітри відеоігор Буденність. На стику стратегій і бойовиків знаходиться грань

Цитирования: **0,01%** id: **11**

«симуляція»

(

Цитирования: **0,02%** id: **12**

«рис. 1.2»

). Це повна протилежність

Цитирования: **0,01%** id: **13**

«рольових ігор»,

які борються з

Цитирования: **0,01%** id: **14**

«буденністю»

цікавим

Цитирования: **0,01%** id: **15**

«сюжетом».

Для ігор цієї грані характерна імітація реальних процесів, обмеження волі, відсутність новизни і повна відсутність сюжету. Рис. 1.2. Межа факторів жанру ігор З боку стратегій в симуляціях головним стає сам

Цитирования: **0,01%** id: **16**

«процес».

З боку бойовиків -

Цитирования: **0,01%** id: **17**

«реалістичність»

відбувається на екрані.У підсумку виходить реалістична імітація процесу, тобто звичайне життя. При цьому існує кілька подій, які постійно відбуваються, і крім них нічого більше немає, нічого нового не з'являється. Центральна точка елемента не використовується, оскільки повна реалістичність недосяжна в іграх, та й кому потрібна точна, стовідсоткова копія реальності, якщо вона вже існує в житті. На стику рольових ігор та бойовиків знаходиться грань

Цитирования: **0,01%** id: **18**

«свобода».

Це повна протилежність

Цитирования: **0,01%** id: **19**

«стратегій»,

які протиставляють

Цитирования: **0,01%** id: **20**

«самотності»

Цитирования: **0,01%** id: **21**

«численність»

підлеглих у розпорядженні. (Стратегії, звичайно ж, не борються з самотністю в звичайному розумінні, але вони імітують владу - найприйнятніше з видів спілкування). Для ігор цієї грані характерні великі відкриті простори, відсутність сімуляторной реалістичності, відсутність новизни, відсутність контролю над іншими учасниками. З боку рольових ігор в

Цитирования: **0,01%** id: **22**

«іграх свободи»

головним стає відсутність готових рольових зв'язків, можливість самостійного вибору, тобто

Цитирования: **0,01%** id: **23**

«рольова свобода».

З боку бойовиків -

Цитирования: **0,01%** id: **24**

«спрощення»

реальних подій і процесів, у порівнянні з симуляторами. У підсумку виходить рольова свобода у спрощеній моделі реального світу. Свобода одного істоти закінчується там, де починається свобода іншого, тому сама крайня точка свободи - це повна самотність. Сама крайня точка спрощення - порожній світ, свобода в просторі. Дія, що має найбільшу

свободу - це створення чогось абсолютно нового. У підсумку виходить центральна точка елемента: повна самотність в порожньому світі, в якому можна створювати - редактор рівнів. Прямо в процесі гри можливості редагування не використовуються, але окремі елементи процесу створення в іграх є. З цього шестикутнику легко можна зрозуміти: що найголовніше в певних видах ігор, між чим доводиться знаходити компроміс розробникам ігор, що ні в якому разі не повинно бути в іграх, що можна між собою змішувати, що змішання не піддається. Щоб пом'якшити переходи грані, розробникам потрібно якісно продумати геймплей - ігровий процес з точки зору гравця - те, без чого вся гра втрачає всякий інтерес, привабливість. Геймплей включає в себе різні аспекти гри, в тому числі технічні, такі як внутрішньоігрова механіка, сукупність певних методів взаємодії гри

Цитирования: **0,02%** id: **25**

"ігрового штучного інтелекту"

з гравцем і ін. Але саме поняття геймплея вкрай узагальнено і зазвичай використовується для вираження отриманих відчуттів в ході проходження гри, під впливом таких чинників, як графіка, звук і сюжет. Тому в першу чергу потрібно продумати поведінкову модель, під фундаментальний жанр гри, а далі розвивати його та покращувати взаємодію з гравцем. 1.2. Порівняльна характеристика ігрових движків Більшість інструментальних засобів для розробки ігор можна поділити на 3 групи: Фреймворк - програмна платформа, яка визначає структуру програмної системи; програмне забезпечення, що полегшує розробку і об'єднання різних компонентів великого програмного проєкту. Фреймворк відрізняється від поняття бібліотеки тим, що бібліотека може бути використана в програмному продукті просто як набір підпрограм близькою функціональності, не впливаючи на архітектуру програмного продукту і не накладаючи на неї ніяких обмежень. У той час як фреймворк диктує правила побудови архітектури додатку, задаючи на початковому етапі розробки поведінка за умовчанням, каркас, який потрібно буде розширювати і змінювати відповідно до зазначених вимог. Ігровий рушій - центральний програмний компонент комп'ютерних та відеоігор або інших інтерактивних додатків з графікою, оброблюваної в реальному часі. Він забезпечує основні технології, спрощує розробку і часто дає грі можливість запускатися на декількох платформах, таких як ігрові консолі та настільні операційні системи, наприклад, GNU / Linux, Mac OS X і Microsoft Windows. Основну функціональність зазвичай забезпечує ігровий движок, що включає движок рендерінга (

Цитирования: **0,01%** id: **26**

«візуалізатор»

), фізичний движок, звук, систему скриптів, анімацію, штучний інтелект, мережевий код, управління пам'яттю і багатопоточність. Часто на процесі розробки можна заощадити за рахунок повторного використання одного ігрового движка для створення безлічі різних ігор. Конструктор ігор - програма, яка об'єднує в собі ігровий рушій і інтегроване середовище розробки, і, як правило, включає в себе редактор рівнів, що працює за принципом WYSIWYG. Такі програми значно спрощує процес розробки ігор, роблячи його доступним любителям-непрограмістів, і можуть бути використані в початковому навчанні програмуванню [1-3]. 1.2.1. Unity За останні роки з'явилося безліч інструментів для розробки ігор. Додаток Unity являє собою професійний ігровий двигун, який використовується в створенні відеоігор для різних платформ. Це інструмент, яким щодня користуються досвідчені розробники, а також один із доступніших інструментів для новачків. До недавнього часу людина, яка вирішила навчитися програмуванню ігор (особливо тривимірних), відразу ж стикався з безліччю серйозних перешкод, в той час як інструмент Unity дозволив значно полегшити життя новачкам [1]. Створення відеоігор в своїй основі не відрізняється від написання будь-якого іншого ПО. Здебільшого відмінності виявляються в кількісній площині. Гра більш інтерактивна, ніж більшість веб-сайтів, а отже, потрібно зовсім інший тип коду, але при цьому в обох випадках будуть задіяні подібні навички і процеси [1]. Будь-який ігровий двигун надає безліч функціональних можливостей, які застосовуються в різних іграх. Реалізована на цьому двигуну гра отримує всі ці функціональні можливості, крім того, додаються її власні ігрові ресурси і код ігрового сценарію. Unity пропонує моделювання фізичних середовищ, карти нормалей, загородження навколишнього світу в екранному просторі (Screen Space Ambient Occlusion, SSAO), динамічні тіні, тощо [1]. Подібний набір функціональних можливостей є в багатьох ігрових двигунів, але у Unity є дві основні переваги перед іншими передовими інструментами розробки ігор: продуктивний візуальний робочий процес і потужна міжплатформенна підтримка [1]. Візуальний робочий процес виділяє даний інструмент з-

посеред інших середовищ розробки ігор. Частина інструментів розробки ігор часто являють собою сукупність розрізнених частин, які потрібно контролювати, чи бібліотеку, для роботи з якою потрібно налаштовувати власне інтегроване середовище розробки, процес збирання проєкту, тощо. Робочий процес в Unity прив'язаний до ретельно продуманого візуального редактору. Редактор надає можливість компонувати сцени майбутньої гри, пов'язуючи ігрові ресурси і код в інтерактивні об'єкти. Він дозволяє швидко і раціонально створювати професійні гри, забезпечуючи високу продуктивність праці розробників і надаючи в їх розпорядження вичерпний перелік сучасних технологій в області відеоігор. Редактор дозволяє редагувати об'єкти в редакторі і рухати елементи в сцені при запущеній грі. Unity надає можливість налаштовувати редактор за допомогою сценаріїв, що додають нові функціональні можливості і елементи меню до інтерфейсу [1]. Unity підтримує наступні платформи: Windows, OSX, Linux, Android, WebGL, iOS, BlackBerry, Tizen, Xbox One, Xbox 360, PS3, PS4, PS Vista, Windows Store, Samsung TV [1]. Швидкість виконання гри вимірюється в кількості кадрів за секунду (FPS – frame per second). На рисунку 1.3. наведено структуру кадру гри, реалізованої на Unity3D [2]. Рис. 1.3. Структура кадру Unity3D У Unity3D існує цілий ряд подій, які виконуються в певному порядку [2]. Наступні функції викликаються при запуску сцени (один раз для кожного об'єкта в сцені): Awake: функція завжди викликається перед будь-якими функціями Start, а також тільки після того, як створено екземпляр prefab (якщо GameObject неактивний під час запуску, функція не викликається, поки об’єкт не стане активним). OnEnable: функція викликається тільки після того, як об'єкт активується. OnLevelWasLoaded: подія настає при завантаженні сцени [2]. Перед першим викликом Update настає подія Start лише якщо об’єкт активовано [2]. OnApplicationPause настає між кадрами, параметр функції вказує на те, чи перебуває додаток в стані паузи [2]. Для побудови основної логіки гри використовуються функції Update: FixedUpdate. Функція викликається фіксовану кількість раз на кадр незалежно від FPS (частіше чи рідше). Після FixedUpdate відбувається розрахунок фізики гри. Update. Функція викликається одразу ж після обробки вхідних даних з частотою відповідною до FPS. LateUpdate. Подія настає після завершення всіх Update. Зручно використовувати для зміни позиції та повороту камери від 3-го лиця [2]. Для прорисовки GUI викликається функція OnGUI (викликається декілька разів на кадр) [2]. Функції yield використовуються для обробки підпрограм, оскільки додаток Unity3D виконується в одному потоці [2]. OnDisable настає при деактивації об’єкта [2]. OnApplicationQuit викликається при завершенні роботи додатку [2]. Гра в Unity складається з декількох сцен, кожна з який містить об’єкти типу GameObject. Об

Цитирования: **0,09%** id: **27**

‘єкти на сцені складають ієрархічну структуру. GameObject – основний клас для всіх об’

єктів на сцені. Кожний GameObject має декілька компонентів (Component), що приєднанні до нього. Обов’язковий компонентом є transform. Transform містить дані про абсолютну позицію об’єкта на сцені, позицію відносно батька, розмір об’єкта, абсолютний поворот об’єкта в сцені та поворот відносно батька. Компоненти реалізують логіку гри. Для створення власного сценарію необхідно унаслідуватися від класу MonoBehaviour, що є компонентом та може бути приєднаний до ігрового об’єкта на сцені [3]. Ігровий двигун Unity3D є універсальним двигуном для розробки ігрових додатків, міжплатформенним та легким у вивченні та використанні. 1.2.2. Torque 2D/3D Torque 2D / 3D, був свого часу лідером, але під натиском Unity втратив свої позиції. Проте до цих пір на ньому розробляється безліч успішних проєктів, оскільки він активно розвивається спільнотою. Відмінності між двовимірної і тривимірної версіями досить значні, але є і загальні елементи, наприклад розвинена мережева підсистема. Після виходу в світ open source T3D зберіг і навіть збільшив свої можливості, а T2D, навпаки, багато втратив. Наприклад, він втратив абсолютно всі вбудовані редактори, які, очевидно, були вилучені за певних юридичних угод. Проте на ньому можна розробляти ігри для трьох платформ: Windows, OS X і, що найцікавіше, iOS (і продавати ігри в App Store, не відраховуючи ні копійки авторам движка). Даний движок - це одна кодова база на C ++ без додаткових експортерів. Як видно, фундаментальні відмінності 2D і 3D - версій полягають в графічної підсистемі: T2D для візуалізації використовує OpenGL, а T3D - Direct. В якості скриптової мови в T2D, як і в T3D, використовується Torque Script. Разом з тим в T2D для опису ігрових елементів служить XML-подібна мова TAML. Вона дозволяє визначити властивості об'єктів на стадії ініціалізації рівня гри. Для відтворення звуків T2D використовує бібліотеку OpenAL. Симуляція фізики здійснюється за допомогою движка Box2D, що став стандартом в двовимірних фізичних обчисленнях. Незважаючи на те що в двовимірному TORK ще поки немає конструктора GUI,

за допомогою засобів движка (в скриптовому коді) можна створювати призначений для користувача інтерфейс звичними компонентами, а не простими спрайтами. Однак, якщо необхідний компонент відсутній, його можна створити на основі спрайтів. Маючи аналогічну з 3D-версією мережеву систему, на T2D можна розробляти мультиплеєрні ігри, які набирають популярність, - наприклад P2P з планшетів. Рис. 1.4. Основний інтерфейс Torque 3D Таблиця 1.1. Переваги і недоліки Torque 3D Недоліки Переваги Немає якісного редактора рівнів Якісний Повільна компіляція Багатоплатформовий Високі вимоги до заліза Легкий в освоєнні Немає вихідного коду Безкоштовний 1.2.3. CryEngine 3 CryEngine 3 бере початок своєї історії в 2001 році, коли була анонсована перша розроблена на ньому гра Far Cry. З тих пір минуло багато часу, і поточна на даний момент п'ята - остання версія була випущена в жовтні 2016-го. Розробники цього движка з самого початку мали на меті не самим створювати на ньому ігри, а продавати його як технологію. Отже, всі розробленні Crytek'ом ігрові програми - це з метою зробити додаткову рекламу своєму головному продукту. Хоча для вивчення він доступний безкоштовно, щоб розробляти на ньому комерційні проєкти, необхідно заплатити, при чому ціна публічно не оголошується. В результаті ліцензіат отримує движок, документацію (навчальні матеріали), вихідний код, а також оперативну підтримку. Крім того, процес ліцензування движка таїть в собі безліч підводних каменів - хоча б те, що ліцензувати його може тільки юридична особа, яка має надати дані про розроблені продукти і в окремих випадках про всіх своїх співробітників. Рис. 1.5. Основний інтерфейс CryEngine 3 Таблиця 1.2. Переваги і недоліки CryEngine 3 Недоліки Переваги Не безкоштовний Якісний Повільна компіляція Багато документації та уроків Високі вимоги до заліза Легкий в освоєнні Не багатоплатформовий Неможливо продавати свої ассети та скрипти На відміну від попередніх движків лінійки (які були виключно PC- орієнтованими), CryEngine 3 орієнтований на створення крос-платформних ігор, призначених для PC і консолей. В даний час підтримуються платформи Xbox 360, Xbox One, PlayStation 3-4, WiiU, а також технології візуалізації настільної Windows - DirectX 9-11. Як можна помітити, підтримки мобільних платформ немає. У ньому спочатку присутня підтримка глобальних мультіплеєрних (MMO) ігор. CryEngine 3 володіє приголомшливим списком технологій візуалізації, ось деякі з них: динамічне освітлення і затінення в реальному часі, затуманення, карти нормалей і паралакс-маппінг, підповерхневе розсіювання, світлові промені і хвилі, управління рівнем деталізації ландшафту, а також багато іншого. Фізичний компонент движка CryPhysics також працює незалежно від фізичних API, таких як PhysX. Вбудована система анімації пропонує кілька відмінних підсистем: індивідуалізація персонажів, параметрична скелетна анімація, процедурне деформування руху. Також заслуговує на окрему увагу вбудована система AA, яка дозволяє обробляти поведінку не тільки персонажів, але і транспортних засобів. Вона складається з трьох модулів: розумні об'єкти, алгоритми динамічного виявлення шляху, а також система, керована сценаріями. 1.2.4. UDK UDK - це безкоштовна версія движка Unreal Engine 3, що володіє всім успадкованим інструментарієм останнього для створення ігрових світів. Список підтримуваних платформ не такий широкий, як у Unity, але цього цілком вистачає, щоб окупити розробку: Windows PC, Windows Store, OS X, iOS, Android і консолі передостаннього покоління. Для скриптинга в движку використовується власна мова - UnrealScript. На сайті розробників представлено багато навчальних матеріалів, як текстових, так і відео, як по редактору, так і по скриптингу. UE3 отримав безліч нагород на індустріальних заходах, а також в кінематографі і не раз ставав кращим ігровим / графічним движком року. Можна сказати, що UDK відрізняється від UE3 тільки відсутністю вихідного коду. Рис. 1.6. Основний інтерфейс UDK Таблиця 1.3. Переваги і недоліки UDK Недоліки Переваги Тільки для професіоналів Безкоштовне розповсюдження Повільна компіляція Простий, зручний інтерфейс; Не самі передові технології Великий набір інструментів для створення ігор; Не багатоплатформовий Багатоплатформовий Що до функціоналу, гнучка система анімації дозволяє контролювати кожну деталь анімованого об'єкта. Анімаційна модель контролюється системою AnimTree, яка включає наступні механізми: контролер змішання (Blend), контролер керований даними, фізичні, процедурно-скелетні контролери. Для імпортування об'єктів використовується формат FBX, що став стандартом для експорту моделей між редакторами. Для візуалізації UE3 використовує 64-бітний кольоровий HDR графічний конвеєр, який здійснює гамма-корекцію, розмиття рухомих об'єктів, зовнішню окклюзію і інші ефекти постобробки. Движком підтримуються всі сучасні ефекти освітлення і технології візуалізації: нормалізовані карти, параметризоване освітлення по Фонгу, різні анізотропні ефекти та інше. UE3 відомий своєю високо оптимізованою мережевою архітектурою, що включає підтримку онлайнових баталій для ігор різних жанрів. 1.2.5. Frostbite Engine Frostbite Engine - ігровий движок,

розроблений компанією EA Digital Illusions CE; застосовується як у власних розробках, так і проєктах інших філіалів Electronic Arts (EA). Першою грою, створеною з використанням цього движка стала Battlefield: Bad Company 2008 року. Движок був розроблений для заміни технічно застарілої технології Refractor Engine, яка використовувалася в попередніх іграх фірми. Рис. 1.7. Основний інтерфейс Frostbite Engine Таблиця 1.4. Переваги і недоліки Frostbite Engine Недоліки Переваги Не безкоштовний Ідеальне руйнування ландшафту Не багатоплатформовий Динамічне освітлення Високі вимоги до заліза Ігровий редактор FrostED Неможливо продавати свої ассети та скрипти Власний звуковий движок Движок відноситься до типу підпрограмного забезпечення (англ. Middleware) і являє собою зв'язку декількох компонентів, таких як графічний движок, звуковий движок і т.д. В операційній системі Microsoft Windows ігровий движок підтримує відображення графіки за допомогою Mantle починаючи з версії 3, DirectX 9, DirectX 10, DirectX 10.1, а починаючи з версії 1.5 - і DirectX 11. Однією із заявлених особливостей є оптимізація для роботи на багатоядерних процесорах. Технологія здатна обробляти знищення ландшафту і оточення (наприклад, будівель, дерев, автомобілів). Підтримується динамічне освітлення і затінення з функцією HBAO, процедурний шейдинг, різні пост- ефекти (наприклад, HDR і depth of field), система частинок і техніки текстурирования, такі, як бамп-маппінг. Максимальний розмір локації становить обмеження в 32 × 32 кілометри відображаємої площі і 4 × 4 кілометри ігрового простору. Крім цього, за твердженням творців, максимальна дистанція промальовування дозволяє побачити рівень аж до горизонту. Також вбудований власний звуковий движок, що не вимагає використання спеціалізованих засобів, подібних EAX. Згадана лише мізерна частина доступних для використання ігрових движків. Таблиця 1.5. Допоміжна порівняльна характеристика усіх движків Характеристика Вимоги Unity 3D Torque 2D/3D CryEngine 3 UDK Frostbite Engine Режим рендеринга 3D + - + + + Звуковий движок - - - + + Фізичний движок + + + + + Ігровий II + - - + + Цільовий жанр RPG + + + - + Мова розробки C# или JavaScript + - + - + Інтеграція з IDE Visual Studio + + + + - ОС для розробки Windows + + + + + Цільові платформи Windows, OSX + + + + + Ціна безкоштовно + + - + - Цільова аудиторія Для професіоналів Так Так Так Ні Так Якість документації 3 2 2 1 3 Інтерфейс користувача 3 2 3 3 2 Результуючий рейтинг 6 4 5 4 5 1.3. Функціональні можливості Unity 3D Unity - це інструмент для розробки двох-і тривимірних додатків та ігор, що працює під операційними системами Windows, Linux і OS X. Створені за допомогою Unity програми працюють під операційними системами Windows, OS X, Windows Phone, Android, Apple iOS, Linux, а також на ігрових приставках Wii, PlayStation 3, PlayStation 4, Xbox 360, Xbox One і MotionPar- allax3D дисплеях (пристрої для відтворення віртуальних голограм), наприклад, Nettlebox. Є можливість створювати додатки для запуску в браузерах за допомогою спеціального модуля Unity (Unity Web Player), а також за допомогою реалізації технології WebGL. Раніше була експериментальна підтримка реалізації проєктів в рамках модуля Adobe Flash Player, але пізніше команда розробників Unity прийняла складне рішення щодо відмови від цього. Рис. 1.8. Браузерна гра запущена за допомогою Unity Web Player Рис. 1.9. Технологія WebGL для запуску браузерних ігор Додатки, створені за допомогою Unity, підтримують DirectX і OpenGL. Активно движок використовується як великими розробниками (Blizzard, EA, QuartSoft, Ubisoft), так і розробниками Indie-ігор (наприклад, Pathologic, Ker- bal, Space Program, Slender: The Eight Pages, Slender : The Arrival, Surgeon Simulator 2013, Baeklyse Apps: Guess the actor та інші) в силу наявності безкоштовної версії, зручного інтерфейсу і простоти роботи з движком. Редактор Unity має простий Drag & Drop інтерфейс, який легко налаштовувати,

Обнаружен Плагиат: **0,08%** https://uk.wikipedia.org/wiki/Unity_(ir… id: **28**

що складається з різних вікон, завдяки чому можна проводити налагодження гри

прямо в редакторі. Движок підтримує три сценарних мови: C #, JavaScript (модифікація), Boo (діалект Python). Boo прибраний в 5-ій версії. Розрахунки фізики виробляє фізичний движок від NVIDIA. Особливості графіки: Система частинок. Спосіб представлення об'єктів, що не мають чітких геометричних кордонів (хмари, туманності, дим, пар, рідини і т.д.); Ефект рендеру в текстуру, тобто можливість приміщення зображення з камери нема на екран, а в текстуру. Потрібно для створення деяких ефектів; Автоматична нарізка і анімація спрайтів. вбудований редактор спрайтів здатний автоматично нарізати растр з спрайтами на кадри і створювати з них анімацію. Прив'язувати події до певних ключовим кадрам анімації; Пакувальник спрайтів, використання різноманітних видів стиснення текстур і упаковка використовуваних кадрів спрайтів в один ресурс; 2D фізика. Вбудована бібліотека Box2D, і підтримка NVIDIA PhysX (фізика реалізована за рахунок використання в розрахунках GPU); Альфавирез, обрізання частини зображення по масці, яку розміщують в

альфакомпоненте текстури; Dynamic Batching, алгоритм оптимізації рендеринга дозволяє домогтися збільшення продуктивності. Особливості роботи з анімацією: State machines. Візуальний редактор станів, які можна прив'язати до ігрових об'єктів і використовувати їх для створення і поліпшення анімації; Інтегрований редактор анімації; Загальні можливості програмного забезпечення: Створення скриптів за допомогою c #, JavaScript або Boo; Нативная інтеграція з Visual Studio; Функції II з передовою системою пошуку шляху; Реалістична анімація; Розміщення одним клацанням; Оптимізована графіка; Конвеєр імпорту; 64-розрядний розширюваний редактор. Unity 5 володіє величезною кількістю переваг перед іншими ігровими движками. Ком'юніті Unity 5 на сьогоднішній момент є найбільшим в світі. На офіційному сайті Unity є спеціальний розділ, в якому можна знайти статистику по ігровим движкам. За цими даними Unity 5 використовує понад 50% розробників відеоігор, 20% належать Unreal Engine, а решта ігрових движків - 30%. Для розробки 2D або 3D інді-ігор Unity 5 підходить за всіма параметрами. Розробка AAA-проєктів в Unity - найскладніший процес. По-перше, будь-який скрипт відразу тягне за собою купу помилок, які в майбутньому необхідно виправити, або переписати скрипт заново. По-друге, Unity 5 все ще має погану оптимізацію. Весь контент який стоїть у вікні Project, але не стоїть в сцені, буде запечений, а значить що гра буде важити в рази більше, ніж передбачалося. Unity в найближчому оновленні виправить цю проблему. У движку є ряд проблем зі скроллингом. При приближенні до об'єкта в певний момент камера наближається повільніше. Якщо потрібно максимально близько наблизиться до землі, то іноді це буває дуже складно зробити. Швидше за все в найближчих оновленнях скролінг так само виправлять. 1.4. Установка Unity і короткий опис можливостей Для того щоб почати установку Unity на свій комп'ютер слід зайти на веб-сайт www.unity3d.com в розділ Store і вибрати потрібну підписку, в нашому випадку, це буде Personal Edition, це єдина представлена безкоштовна версія. Після установки у нас є можливість відразу зайти в налаштування і вказати яку IDE ми будемо використовувати (рис. 1.10). із запропонованих варіантів можна вибрати Mono Developer, Microsoft Visual Studio, або будь-який інший IDE (наприклад: IntelliJ IDEA, для цього є спеціальна версія для Unity з підтримкою c# і javascript), який вам підходить. Нашим вибором буде Mono Developer. Рис. 1.10. Unity preferences Після установки у нас є можливість відразу зайти в налаштування і вказати яку IDE ми будемо використовувати (рис. 1.10). із запропонованих варіантів можна вибрати Mono Developer, Microsoft Visual Studio, або будь-який інший IDE (наприклад: IntelliJ IDEA, для цього є спеціальна версія для Unity з підтримкою c# і javascript), який вам підходить. Так як збирати проєкт ми будемо не тільки під Windows, але і під Android, то нам необхідно буде завантажити Android SDK. Зробити це можна зайшовши на вебсайт developer.android.com в розділ downloads і вказавши шлях до нього після установки в Unity preferences. При першому запуску Unity ми потрапляємо у вікно створення проєктів, так як таких у нас поки що немає, Unity відразу пропонує нам створити новий проєкт (рис 1.11). Рис.1.11. Створення проєкту 1. Ім'я за замовчуванням New Unity Project, ви можете поміняти його на що завгодно (рис. 1.12). Розташування вашого майбутнього проєкту вказує на домашню папку вашого профілю в Windows. Розташування можна змінити тільки зараз, при створенні проєкту. Рис. 1.12 Вибір назви і розташування проєкту 2. Вибрати тип проєкту, 3D або 2D. Це вплине на налаштування візуального редактора Unity. Їх завжди можна поміняти вже після створення проєкту, так що можна залишити їх за замовчуванням. 3. Вибрати додаткові Ассет (рис. 1.13), які будуть включені в наш проєкт. Ассет представляють собою заздалегідь створені текстури, стилі, світлові ефекти, корисні додаткові інструменти та інше. Рис. 1.13. Вибір Ассет пакетів за замовчуванням Після чого можемо створити проєкт і тепер завжди при старті Unity ми будемо потрапляти в список створених нами проєктів, вибравши один з них ми потрапляємо в візуальний редактор Unity. 1.5. Інтерфейс редактора Unity Редактор складається з декількох основних частин, які можна перетягувати, групувати і розташовувати, так як нам зручно (рис. 1.14). Рис. 1.14. Візуальний редактор Project window (вікно проєкту) відображає список встановлених Ассет, текстур, файлів анімацій, скриптів, матеріалів, спрайтів, сцен, в загальному, все того що ви додали і будете використовувати в своєму проєкті. Додавання файлів відбувається простим перетягуванням файлів або їх груп в це вікно. У разі, якщо ми хочемо створити новий файл, то встановивши курсор на цьому вікні правою кнопкою миші, ми можемо викликати контекстне меню де вибираємо тип ресурсу, який хочемо створити (рис. 1.15). Рис. 1.15. Вікно Project Unity автоматично імпортує додані файли в потрібний для неї формат. Надалі ми можемо експортувати всі додані сюди файли одним пакетом і використовувати їх в іншому проєкті. Вікно Scene, дозволяє нам управляти, редагувати сцену або кілька сцен проєкту, тобто вікно, де будуть відображатися об'єкти нашої гри

(рис.1.16). Рис. 1.16. Вікно Scene Вікно має можливість перемикатися в 2D або 3D режими, в залежності від типу нашого проєкту, вмикати і вимикати освітлення. Дозволяє вибирати, переміщати і змінювати позицію об'єктів знаходяться на сцені (наприклад: джерела світла, персонажа, камери та інші), тимчасово вимикати об'єкти. Вікно Hierarchy відображає ієрархію всіх об'єктів знаходяться на сцені. Кожен елемент сцени має запис в цьому вікні. вікно Hierarchy відображає структуру сцени (рис.1.17) і як об'єкти прикріплені один до одного, воно нерозривно пов'язане з вікном Scene. Вікно Inspector дає нам можливість переглядати і змінювати всі можливі параметри обраних об'єктів. Так як параметри різних об'єктів відрізняються, то розташування і вміст вікна інспектора може змінюватися в залежності від обраного об'єкта. Рис. 1.17 - Вікно Hierarchy Інспектор (рис. 1.18) ми будемо використовувати для перегляду та зміни параметрів і налаштувань майже все, що є в редакторі Unity, включаючи ігрові об'єкти, матеріали, спрайт, Ассет і так далі. Якщо до об'єкта доданий скрипт, то Інспектор (рис. 1.19) буде відображати всі публічні змінні і посилання на інші об'єкти, присутні в ньому. Вміст цих змінних можна змінювати прямо в цьому вікні, не залазячи в скрипт і не змінюючи код безпосередньо. Рис. 1.18 Вікно Inspector Рис. 1.19. Вікно Inspector – Camera Toolbar надає доступ до найбільш часто використовуваних функцій редактора (рис. 1.20). Справа знаходяться кнопки доступу до хмарних сервісів і аккаунту Unity. Layers відповідає за видимість шарів на сцені, дозволяє включати і вимикати їх у процесі налагодження. У середній частині панелі знаходяться кнопки управління роботою додатка, що дозволяють запускати, ставити на паузу і покроково його виконувати. Ліва частина містить стандартні інструменти управління сценою і об'єктами на ній. Рис. 1.20. Toolbar 1.6. Висновки до розділу В розділі проведено огляд та аналіз популярних сучасних засобів для розробки ігор та їх представників, розбиття засобів на класи за їх базовими заявленими функціональними можливостями. Визначено оптимальні сфери використання для інструментальних засобів спираючись на аналіз процесу розробки прототипів. Приводиться порівняльна характеристика ігрових движків (Unity, Torque 2D/3D, CryEngine 3, UDK, Frostbite Engine). Проведено аналіз функціональних можливостей Unity 3D. РОЗДІЛ 2.
РОЗРОБКА АРХІТЕКТУРИ ІГРОВОГО ПРОЄКТУ Створення ігрового додатку дуже трудомісткий і складний процес. В ході цього процесу величезна кількість елементів об'єднується в програму призначену для того, щоб розважати величезну масу людей. 2.1. Основні модулі ігрового додатку Наш ігрове додаток буде мати будову класичної гри, яка складається з декількох основних частин або модулів, кожен з яких виконує свою унікальну функцію[8]: Модуль управління. Відповідає за опитування зовнішніх пристроїв управління, наприклад, таких як миша, клавіатура; Модуль, який відповідає за рендеринг об'єктів сцени. Цей модуль займається тим, що відображає на екрані гравця ігрові об'єкти, елементи меню, текст, вже з урахуванням використовуваних текстур, матеріалів, джерел світла, шейдерів. Саме цей модуль по засобом API (наприклад: DirectX, OpenGL, в залежності від обраної платформи) спілкується з апаратною частиною комп'ютера, і зокрема GPU: Модуль, який відповідає за звук; Модуль штучного інтелекту. В нашій майбутній грі буде використовуватися тільки частина з цих модулів, це модуль управління, звуку і рендеринга. Всі ці модулі вже реалізовані в ігровому движку, тому нам немає необхідності самим займатися їх розробкою. Для гри нам залишається тільки реалізувати свій модуль, який відповідає за ігрову логіку і фізику, який буде управляти взаємодією ігрових об'єктів на сцені, підраховувати очки, перемикати сцени в залежності від обраного пункту меню, завантажувати карти рівнів і так далі. Розгорнутий опис функціональних вимог здійснюється на етапі проєктування програми. Для того щоб деталізувати вимоги, необхідно виділити процеси, що відбуваються в заданій предметній області. Опис таких процесів на UML виконується у вигляді прецедентів (рис. 2.1). Прецеденти є сценарієм або варіантом використання ігрового програми при взаємодії із зовнішнім середовищем. За допомогою прецедентів описується функціонування гри з точки зору користувача, який називається в UML актором (actor). Актор є будь- яку зовнішню

Обнаружен Плагиат: **0,14%** https://pns.hneu.edu.ua/pluginfile.ph... id: **29**

по відношенню до модельованої системі сутність (людина, програмна система, пристрій), яка взаємодіє з системою і використовує її функціональні можливості

для досягнення певної мети або вирішення приватних завдань. Рис. 2.1. Діаграма прецедентів 2.2. Ігровий цикл Серцем гри є ігровий цикл (рис. 2.2), який здійснює послідовний виклик всіх цих модулів в правильній послідовності і синхронізацію їх між собою. Наш ігровий цикл буде реалізований в скрипті і підключений до камери сцени. Рис. 2.2. Діаграма найпростішого ігрового циклу Ігровий цикл нашої гри буде складатися з:

опитування стану зовнішніх пристроїв управління; зміна положень ігрових об'єктів; підрахунок очок; лічильник часу відведеного на проходження рівня; оновлення координат фізичних об'єктів сцени; оновлення елементів меню і інтерфейсу. Рис. 2.3. Реалізація ігрового циклу в нашій грі Використання класичного ігрового циклу в Unity неможливо через особливості архітектури движка. Кожен об'єкт в Unity має в скрипті два обов'язкових методу. Перший метод Start() для ініціалізації об'єкта і використовується один раз при додаванні об'єкта на сцену. Другий метод називається Update(), Unity викликає його кожен раз і у кожного об'єкта, перед тим як оновити всю сцену (рис. 2.2)[9]. 2.3. Система меню Для додатка потрібно реалізувати систему меню, яка дозволить гравцеві запускати гру, перезапускати рівень, виходити з програми, включати і вимикати звук. Для її проєктування зручно використовувати діаграму станів Головне меню буде включати наступні пункти: Старт гри, вибір цього пункту викликає завантаження сцени з останнім пройденим рівнем (рис. 2.4); вихід з програми (рис. 2.5); кнопка Звук, повністю виключає або включає звукові ефекти в ігровому додатку (рис. 2.6); інформація про творця гри, показує вікно з короткою інформацією про розробника (рис. 2.7). Рис. 2.4. Діаграма станів - Старт гри Рис. 2.5. Діаграма станів - Вихід з гри Рис. 2.6. Діаграма станів – Звук Рис. 2.7. Діаграма станів – Інформація Додаткове меню, яке можна буде використовувати в процесі гри: Рестарт рівня, перезапускає сцену з рівнем, скидає лічильник часу; Вихід в головне меню. На стадії розробки гри вкрай необхідно приділити увагу графічної, візуальної складової гри, від її якості залежить популярність гри, а значить і кількість її продажів. Рис. 2.8. Діаграма станів - Рестарт рівня На стадії розробки гри вкрай необхідно приділити увагу графічної, візуальної складової гри, від її якості залежить популярність гри, а значить і кількість її продажів. Але й не варто сильно захоплюватися цим процесом, так як це може позначитися на підсумковій продуктивності ігрового програми і відповідно погіршити такі показники як фреймрейт або fps (frames per second), відгук управління і плавність ігрового процесу, обсяг займаного іграми платформи простору і часом завантажити програму з мережі. 2.4. Висновки до розділу У другому розділі в результаті узагальнення всієї наявної інформації по роботі з Unity 3D, була розроблена архітектура ігрового проєкту. Описано основні модулі ігрового додатку та ігровий цикл. РОЗДІЛ 3.
ОСНОВНІ ЕТАПИ РЕАЛІЗАЦІЇ ІГРОВОГО ПРОЄКТУ 3.1. Проєктування інфологічної моделі бази даних Інфологія ІС – складене слово від двох коренів слів

Цитирования: **0,01%** id: **30**

«інформація»

і

Цитирования: **0,01%** id: **31**

«логіка»,

тобто інформаційна логіка (функціонування) інформаційної системи. Відповідно, інфологічне (інформаційно-логічне) моделювання та проєктування представляють процеси, пов'язані з поданням семантики у вигляді моделі, якій властиві морфізм і заходи. Основними конструктивними елементами інфологічних моделей є сутності, зв'язки між ними і їх властивості (атрибути). Інфологічна модель бази даних являє собою опис об'єктів (сутностей), з набором атрибутів і зв'язків між ними та призначена для структурного утворення предметної області. Вона повинна бути стабільної та незмінною. Загальновідомо, що при побудові інфологічних моделей баз даних, зокрема, часто користуються мовою ER-діаграм (від англ. entity-relationship, тобто

Цитирования: **0,01%** id: **32**

«сутність-зв'язок»

). Її компонентами є сутності і зв'язки. Сутністю може бути будь-який інформаційний об’єкт, що необхідно зберігати у базі даних. Це може бути формалізований опис конкретного галузі предметної області інформаційної системи, що розроблюється. Дуже важливо розрізняти поняття сутності та екземпляру сутності, адже екземпляром сутності є лише конкретний прояв дії або об’єкту галузі, що описується [9]. Атрибутом сутності виступає та чи інша характеристика присутня сутності. Вона може бути як кількісною, так і якісною. Назва атрибута повинна унікальною у межах однієї сутності, але атрибути різних сутностей можуть співпадати. Атрибут вказує на ту інформацію, яку необхідно зібрати про сутність. Атрибути також має екземпляри, але в даному випадку кожному атрибуту привласнюється тільки одне значення до відповідної сутності Ключем сутності називають набір атрибутів або один атрибут, що однозначно ідентифікують сутність у межах моделі.

Обнаружен Плагиат: **0,63%** https://kivra.kpi.ua/wp-content/uploa... id: **33**

Зв'язок – асоціювання двох або більше сутностей. Якби призначенням бази даних було тільки збереження окремих, не пов'язаних між собою даних, то її структура могла б бути дуже простою. Проте одна з основних вимог до організації бази даних – це забезпечення можливості відшукання одних сутностей за значеннями інших, для чого необхідно встановити між ними певні зв'язки. А так як в реальних базах даних нерідко містяться сотні або навіть тисячі сутностей, то теоретично між ними може бути встановлено більше мільйона зв'язків. Наявність такої безлічі зв'язків і визначає складність

інфологічних моделей. При побудові інфологічних моделей можна використовувати мову ER- діаграм (від англ. Entity-Relationship, тобто сутність-зв'язок). У них сутності зображуються позначеними прямокутниками, асоціації – поміченими ромбами або шестикутниками, атрибути – поміченими овалами, а

Обнаружен Плагиат: **0,13%** https://dystosvita.org.ua/mod/page/v... id: **34**

зв'язки між ними – ненаправленими ребрами, над якими може проставлятися ступінь зв'язку (1 або буква, що заміняє слово

Цитирования: **0,01%** id: **35**

"багато"

)

Обнаружен Плагиат: **0,08%** https://dystosvita.org.ua/mod/page/v... + 3 ресурсов! id: **36**

і необхідне пояснення. Між двома сутностями, можливі чотири види простих зв'язків:

Цитирования: **0,02%** id: **37**

«один до одного»,

«один до багатьох»,

Цитирования: **0,02%** id: **38**

«багато до одного»,

«багато до багатьох» і більш складні зв'язки, наприклад, семантичні зв'язки, що визначають якість функціонування семантичних інформаційних систем (інформаційних порталів, бібліотек, навчальних систем і т.д.). Перший тип – зв'язок

Обнаружен Плагиат: **0,11%** https://dspace.chmnu.edu.ua/jspui/b... id: **39**

ОДИН-ДО-ОДНОГО (1: 1): в кожен момент часу кожному представнику (екземпляру) сутності А відповідає 1

чи 0 представників сутності В. Так як між двома сутностями можливі зв'язки в обох напрямках, то існує ще два типи зв'язку БАГАТО-ДО-ОДНОГО (М: 1) і БАГАТО-ДО-БАГАТЬОХ (М: N). Крістофер Дейт – один з найбільших фахівців в області баз даних – визначає три основні класи

Обнаружен Плагиат: **0,19%** https://yak.koshachek.com/articles/2... id: **40**

сутностей: стрижневі, асоціативні і характеристичні, а також підклас асоціативних сутностей – позначення. Стрижнева сутність (стрижень) – це незалежна сутність. Асоціативна сутність (асоціація) – це зв'язок виду

Цитирования: **0,01%** id: **41**

"багато-до-багатьох"

Обнаружен Плагиат: **0,42%** https://yak.koshachek.com/articles/2... id: **42**

між двома або більше сутностями або екземплярами сутності. Асоціації розглядаються як повноправні суті: вони можуть брати участь в інших асоціаціях і позначеннях точно так же, як стрижневі сутності; можуть мати властивості, тобто мати не тільки набір ключових атрибутів, необхідних для вказівки зв'язків, а й будь-яке число інших атрибутів, що характеризують зв'язок. Характеристична сутність (характеристика) – це зв'язок виду

Цитирования: **0,01%** id: **43**

"багато-до- одного"

або

Цитирования: **0,01%** id: **44**

"один-до-одного"

Обнаружен Плагиат: **0,34%** https://yak.koshachek.com/articles/2… id: **45**

між двома сутностями (окремий випадок асоціації). Єдина мета характеристики в рамках розглянутої предметної області складається в описі чи уточнення деякої іншої сутності. Необхідність в них виникає в зв'язку з тим, що сутності реального світу мають іноді багатозначні властивості. Позначаюча сутність або позначення – це зв'язок виду

Цитирования: **0,01%** id: **46**

"багато-до-одного"

або

Цитирования: **0,01%** id: **47**

"один-до-одного"

Обнаружен Плагиат: **0,09%** https://yak.koshachek.com/articles/2… id: **48**

між двома сутностями і відрізняється від характеристики тим, що не залежить від

позначаємої сутності.

Обнаружен Плагиат: **0,24%** https://yak.koshachek.com/articles/2… id: **49**

Як правило, позначення не розглядаються як повноправні сутності, хоча це не привело б до будь-якої помилки Позначення і характеристики не є повністю незалежними сутностями, оскільки вони припускають наявність деякої іншої сутності, яка буде

Цитирования: **0,01%** id: **50**

"позначатися"

або

Цитирования: **0,01%** id: **51**

"характеризуватися".

Обнаружен Плагиат: **0,37%** https://yak.koshachek.com/articles/2… id: **52**

Однак вони все ж є окремі випадки сутності і можуть, звичайно, мати властивості, можуть брати участь в асоціаціях, позначеннях і мати свої власні (нижчого рівня) характеристики. Підкреслимо також, що всі екземпляри характеристики повинні бути обов'язково пов'язані з будь-яким екземпляром характеризуємої сутності. Однак допускається, щоб деякі екземпляри характеризуємої сутності не мали зв'язків.

Обнаружен Плагиат: **0,54%** https://yak.koshachek.com/articles/2… + 3 ресурсов! id: **53**

Ключ або можливий ключ – це мінімальний набір атрибутів, за значеннями яких можна однозначно знайти необхідний екземпляр сутності. Мінімальність означає, що виключення з набору будь-якого атрибута не дозволяє ідентифікувати сутність по тим атрибутам, що залишилися. Кожна сутність має хоча б один можливий ключ. Один з них приймається за первинний ключ. При виборі первинного ключа слід віддавати перевагу простим ключам або ключам, складеним з мінімального числа атрибутів. Недоцільно також використовувати ключі з довгими текстовими значеннями.

Обнаружен Плагиат: **0,61%** https://yak.koshachek.com/articles/2… id: **54**

Не допускається, щоб первинний ключ стрижневий сутності (будь-який атрибут, який бере участь в первинному ключі) брав невизначений значення. Інакше виникне суперечлива ситуація: з'явиться екземпляр стрижневий сутності, який не володіє індивідуальністю, і, отже не існуючий. З тих же причин необхідно забезпечити унікальність первинного ключа. Тепер про зовнішні ключі: –якщо сутність С пов'язує сутності А і В, то вона повинна включати зовнішні ключі, відповідні первинним ключам сутностей А і В. –якщо сутність У позначає сутність А, то вона повинна включати зовнішній ключ, відповідний первинному ключу сутності

А. Цілісність (від англ. Integrity – недоторканність, збереженість,

Обнаружен Плагиат: **0,28%** https://yak.koshachek.com/articles/2… + 3 ресурсов! id: **55**

цілісність) розуміється як правильність даних в будь-який момент часу. Але ця мета може бути досягнута лише в певних межах: СУБД не може контролювати правильність кожного окремого значення, що вводиться в базу даних (хоча кожне значення можна перевірити

на правдоподібність

).

Обнаружен Плагиат: **0,71%** https://yak.koshachek.com/articles/2... + 2 ресурсов! id: **56**

Підтримка цілісності бази даних може розглядатися як захист даних від невірних змін або руйнувань (не плутати з незаконними змінами і руйнуваннями, які є проблемою безпеки). Сучасні СУБД мають ряд засобів для забезпечення підтримки цілісності (так само, як і засобів забезпечення підтримки безпеки). Виділяють три групи правил цілісності: цілісність по сутностей цілісність по посиланнях цілісність, що визначається користувачем Не допускається, щоб будь-який атрибут, який бере участь в первинному ключі, приймав невизначене значення. Значення зовнішнього ключа повинно або: бути рівним значенню первинного ключа мети; бути повністю невизначеним, тобто кожне значення атрибуту, який бере участь в зовнішньому ключі має бути невизначеним.

[13] Інфологічна модель бази даних цього проєкту зображена на додатку А. 3.2. Проєктування даталогічної моделі бази даних Даталогічне

Обнаружен Плагиат: **0,23%** https://ela.kpi.ua/bitstream/12345678... + 2 ресурсов! id: **57**

проєктування – створення схеми бази даних на основі конкретної моделі даних, наприклад, реляційної моделі даних. Для реляційної моделі даних даталогічна модель – набір схем відносин, зазвичай із зазначенням первинних ключів, а також

Цитирования: **0,01%** id: **58**

«зв'язків»

між відносинами, що представляють собою зовнішні ключі. Будь-яка СУБД оперує з допустимими для неї логічними одиницями даних, а також допускає використання певних правил композиції логічних структур більш високого рівня зі складових інформаційних одиниць нижчого рівня. Крім того, багато СУБД накладають кількісні та інші обмеження на структуру бази даних. Тому перш ніж приступити до побудови даталогічної моделі, необхідно детально вивчити особливості СУБД, визначити чинники, що впливають на вибір проєктного рішення, ознайомитися з існуючими методиками проєктування, а також провести аналіз наявних засобів автоматизації проєктування, можливості і доцільності їх використання. Хоча даталогічне проєктування є проєктуванням логічної структури бази даних, на нього впливають можливості фізичної організації даних, що надаються конкретною СУБД. Тому знання особливостей фізичної організації даних є корисним при проєктуванні логічної структури. Логічна структура бази даних, а також сама заповнена даними база даних є відображенням реальної предметної області. Тому на вибір проєктних рішень найбезпосередніший вплив надає специфіка відображається предметної області, відображена в інфологічній моделі. Кінцевим результатом даталогічного проєктування є опис логічної структури бази даних на мові опису даних. Однак якщо проєктування виконується

Цитирования: **0,01%** id: **59**

«вручну»,

то для більшої наочності спочатку будується схематичне графічне зображення структури бази даних. При цьому повинно бути забезпечено однозначна відповідність між конструкціями мови опису даних і графічними позначеннями інформаційних одиниць і зв'язків між ними. Графічне представлення використовується і при автоматизованому проєктуванні структури бази даних як частиною інтерфейсу засіб спілкування з проєктувальником, і при документуванні проєкту. Спроєктувати логічну структуру бази даних означає визначити всі інформаційні одиниці і зв'язку між ними, задати їх імена; якщо для інформаційних одиниць можливе використання різних типів, то необхідно визначити їх тип. Слід також задати деякі кількісні характеристики, наприклад довжину поля. Кожному типу моделі даних і кожного різновиду моделі, яку підтримує конкретної СУБД, притаманні свої специфічні особливості. Разом з тим є багато спільного у всіх структурованих моделях даних і принципах проєктування БД в їх середовищі. Все це дає можливість використовувати єдиний методологічний підхід до проєктування структури бази даних. В БД відображається певна предметна область. Тому процес проєктування БД передбачає попередню класифікацію об'єктів предметної області, систематизоване представлення інформації про об'єкти і зв'язки між ними. На проєктні рішення впливають особливості необхідної обробки даних. Тому відповідна інформація повинна бути певним чином представлена і проаналізована на початкових етапах проєктування БД. Дані про

предметну область і особливості обробки інформації в ній фіксуються в інфологічної моделі. У такій моделі відображена вся інформація, що циркулює в інформаційній системі, але це зовсім не означає, що вся вона повинна зберігатися в базі даних. У зв'язку з цим одним з перших кроків проєктування є визначення складу БД, тобто переліку тих показників, які доцільно зберігати в БД. При проєктуванні логічної структури БД здійснюються перетворення вихідної інфологічної моделі в модель даних, підтримувану конкретної СУБД, і перевірка адекватності отриманої даталогічної моделі відображається предметної області. Для будь-якої предметної області існує безліч варіантів проєктних рішень її відображення в даталогічної моделі. Методика проєктування повинна забезпечувати вибір найбільш підходящого проєктного рішення. Мінімальна логічна одиниця даних (незважаючи на їх різні назви) семантично для всіх СУБД однакова і відповідає або ідентифікатором об'єкта, або властивості об'єкта або процесу. Зв'язки між сутностями предметної області, відображені в інфологічної моделі, можуть відображатися в даталогічній моделі або за допомогою спільного розташування відповідних їм інформаційних елементів, або шляхом оголошення зв'язку між ними. Не всі види зв'язків, що існують в предметній області, можуть бути безпосередньо відображені в конкретній даталогічній моделі. Так, багато СУБД не підтримують безпосередньо відношення М: М між елементами. У цьому випадку в даталогічну модель вводиться додатковий допоміжний елемент, що відображає цей зв'язок (таким чином, ставлення М: М хіба що розбивається на два відносини 1: М між цим знову введеним елементом і вихідними елементами). Слід звернути увагу на те, що відносини, які мають місце в предметній області, можуть бути передані не тільки за допомогою структури бази даних, але і програмним шляхом (тобто завжди існує альтернатива між декларативним і процедурним способом опису явища). Наприклад, при відображенні узагальнених об'єктів можна не виділяти підкласи на рівні логічної структури бази даних. В цьому випадку підкласи будуть виділятися програмним шляхом при обробці даних, що зберігаються. Рішення про те, який із способів відображення (структурний / декларативний або програмний / процедурний) слід використовувати в кожному конкретному випадку, буде залежати від багатьох факторів, таких, як стабільність сутності, що відображається обсяг номенклатури, особливості СУБД, характер обробки даних та ін. Так, якщо в предметній області використовується класифікація співробітників по статі, в базі даних не слід створювати класифікатор статей, оскільки він буде містити всього дві позиції і ніколи не змінюється. Як правило, не слід виділяти за цією ознакою і відповідні підкласи для об'єктів предметної області, тому що в більшості випадків вони обробляються спільно і в основному мають однаковий набір властивостей, що характеризує їх. Однак в деяких предметних областях поділ на такі підкласи може бути доцільним. Якщо ж відображається сутність не стабільна, то її краще передавати за допомогою даних, так як в противному випадку буде часто турбуватися перетворення програми, що зазвичай забезпечити важче, ніж зміна даних. При відображенні узагальнених об'єктів в БД можливі різні варіанти: зберігати всю інформацію про всі узагальнені об'єкті в одному файлі / таблиці, кожному підкласу об'єктів нижчого рівня виділяти окремі самостійні файли / таблиці. І перший, і другий варіанти широко використовуються у різноманітних СУБД. У першому випадку підкреслюється спільність об'єктів різних підкласів, що входять в узагальнений об'єкт. У другому випадку, навпаки, узагальнений об'єкт як єдине ціле не відображається в структурі бази даних. Інші способи відображення пов'язані з явним або неявним виділенням підкласів в логічній структурі БД. Неявне виділення підкласу полягає в тому, що в запису відводяться поля для фіксації значень властивостей, загальних для об'єктів різних підкласів, і значення ознаки підкласу, а замість полів, наявність яких залежить від підкласу, використовується одне поле зі змінним складом, зміст якого буде залежати від того, до якого підкласу відноситься описуваний об'єкт. Реалізація принципу явного виділення підкласів в структурі БД істотно залежить від специфіки СУБД. При проєктуванні логічної структури БД основне значення має специфіка предметної області, що відображається. Однак, як зазначалося вище, і характер обробки інформації впливає на прийняте проєктне рішення. Наприклад, рекомендується зберігати разом інформацію, часто оброблювану спільно, і, навпаки, розділяти по різних файлах інформацію, не що використовується одночасно. Інформацію, яку використовують часто, і інформацію, частота звернення до якої мала, також слід зберігати в різних файлах, причому останню може виявитися вигідним винести в архівні файли, а не підтримувати в складі БД. Як зазначалося вище, описується не окремий об'єкт, а клас об'єктів. Але в окремих випадках буває, що клас включає в себе тільки один екземпляр. Наприклад, якщо предметною областю є якийсь конкретний інститут, то клас об'єктів буде містити тільки один екземпляр об'єкта. Таким

Цитирования: **0,01%** id: **60**

«виродженим»

класами об'єктів зазвичай не ставиться у відповідність окремий файл бази даних[12]1). На основі інфологічної моделі з попереднього пункту була спроєктована даталогічна модель. У якості системи керування базою даних була обрана SQLite – компактна вбудовувана реляційна база даних, що поставляється з вихідними кодами. Вперше випущена в 2000 році, призначена для надання звичних можливостей реляційних баз даних без властивих їм накладних витрат. За час експлуатації встигла заслужити репутацію як переносима, легка у використанні, продуктивна і надійна база даних. Слово

Цитирования: **0,01%** id: **61**

«вбудована»

(embedded) означає, що база даних існує не як процес, окремий від обслуговується процесу, а є його частиною – частиною деякого прикладного застосування. SQLite

Обнаружен Плагиат: **1,42%** https://kievoit.ippo.kubg.edu.ua/kievo... + 4 ресурсов! id: **62**

не використовує парадигму клієнт-сервер, вона являє собою бібліотеку, з якої програма компонується, і SQLite стає складовою частиною програми. Таким чином, в якості протоколу обміну використовуються виклики функцій (API) бібліотеки SQLite. І клієнт, і сервер працюють в одному процесі – це позбавляє від проблем конфігурації. Все, чого потребує програміст, вже скомпільовано в його додатку. Такий підхід зменшує накладні витрати, час відгуку і спрощує програму. SQLite зберігає всю базу даних (включаючи визначення, таблиці, індекси і дані) в єдиному стандартному файлі на тому пристрої, на якому виповнюється програма. Простота реалізації досягається за рахунок того, що перед початком виконання транзакції запису весь файл, який зберігає базу даних, блокується; ACID-функції досягаються в тому числі за рахунок створення файлу журналу. Кілька процесів або потоків можуть одночасно без будь-яких проблем читати дані з однієї бази. Запис в базу можна здійснити тільки в тому випадку, якщо ніякі інші запити в даний момент не обслуговуються; в іншому випадку спроба запису закінчується невдачею, і в програму повертається код помилки. Іншим варіантом розвитку подій є автоматичне повторення спроб запису протягом заданого інтервалу часу. У комплекті поставки йде також функціональна клієнтська частина у вигляді виконуваного файлу sqlite3, за допомогою якого демонструється реалізація функцій основної бібліотеки. Клієнтська частина

є кросплатформною утилітою командного рядка. Завдяки архітектурі SQLite її можливо використовувати як на вбудовуваних системах, так і на виділених машинах з великими масивами даних. SQLite підтримується динамічна типізація даних. Підтримуються такі типи даних: NULL – NULL-значення; INTEGER – цілочисельне значення зі знаком; REAL: число з плаваючою комою; TEXT: текстовий рядок з кодуванням UTF-8, UTF-16BE або UTF- 16LE; BLOB: тип даних, що зберігається точно в такому ж вигляді, в якому і був отриманий. На поточному ринку вбудовуваних баз даних представлено багато продуктів від різних виробників, але тільки один з них поставляється з відкритим кодом, не вимагає ліцензійних зборів і спроєктований виключно як вбудовувана БД – це SQLite. SQLite має модульну архітектуру, яка відображає унікальні підходи до управління реляційними базами даних. Вісім окремих модулів згруповані в три головних підсистеми (рис. 3.1). Вони поділяють обробку запиту на окремі завдання, які працюють подібно конвеєру. Верхні модулі компілюють запити, середні виконують їх, а нижні працюють з диском і взаємодіють з операційною системою. Рис. 3.1. Архітектура SQLite Незважаючи на маленький розмір, SQLite надає великий спектр особливостей і можливостей. Він підтримує вельми повний набір стандарту ANSI SQL92 для особливостей мови SQL, а також такі особливості як тригери, індекси, стовпці з автоінкрементом та LIMIT / OFFSET особливості. Так само підтримуються такі рідкісні властивості, як динамічна типізація і вирішення конфліктів. Даталогічна модель наведна у додатку Б. Використовуючи цю систему керування базами даних, була створена база даних, яка має таблиці, наведені на рис. 3.2 – 3.9. Таблиця

Цитирования: **0,01%** id: **63**

«Hero»

містить у собі інформацію про головного героя. Структура даної таблиці наведена на рис. 3.2. Рис. 3.2. Таблиця

Цитирования: **0,01%** id: **64**

«Hero»

Таблиця

Цитирования: **0,01%** id: **65**

«Enemy»

містить у собі інформацію про ворогів. Структура даної таблиці наведена на рис. 3.3. Рис. 3.3. Таблиця

Цитирования: **0,01%** id: **66**

«Enemy»

Таблиця

Цитирования: **0,01%** id: **67**

«Level»

містить у собі інформацію про рівень, на якому знаходиться герой (рис. 3.4). Рис. 3.4. Таблиця

Цитирования: **0,01%** id: **68**

«Level»

Таблиця

Цитирования: **0,01%** id: **69**

«Weapon»

містить у собі інформацію про зброю (рис. 3.5). Рис. 3.5. Таблиця

Цитирования: **0,01%** id: **70**

«Weapon»

Таблиця

Цитирования: **0,01%** id: **71**

«Armor»

містить у собі інформацію про броню. Структура даної таблиці наведена на рис. 3.6. Рис. 3.6. Таблиця

Цитирования: **0,01%** id: **72**

«Weapon»

Таблиця

Цитирования: **0,01%** id: **73**

«Potions»

містить у собі інформацію про зілля. Структура даної таблиці наведена на рис. 3.7. Рис. 3.7. Таблиця

Цитирования: **0,01%** id: **74**

«Weapon»

Таблиця

Цитирования: **0,01%** id: **75**

«Hero_Level»

містить у собі інформацію про рівень, який на даний момент є у героя. Структура даної таблиці наведена на рис. 3.8. Рис. 3.8. Таблиця

Цитирования: **0,01%** id: **76**

«Hero_Level»

Таблиця

Цитирования: **0,01%** id: **77**

«Loot»

містить у собі інформацію про речі, знайдені героєм. Структура даної таблиці наведена на рис. 3.9. Рис. 3.9. Таблиця

Цитирования: **0,01%** id: **78**

«Loot»

3.3. Процес створення ігрового додатку 3.3.1. Створення проєкту Для створення проєкту необхідно скачати та встановити Unity Game Engine. Після інсталяції та запуску Unity буде представлено інтерфейсом, що зображено на рис 3.10. Рис 3.10. Вікно створення нового проєкту Натиснувши

Цитирования: **0,01%** id: **79**

“New”

треба обрати ім’я проєкту. Після введення ім’я необхідно обрати вимірність розробляємого оточення та шлях розташування проєкту на рис 3.11.-3.12. Рис 3.11. Вікно обрання ім’я проєкту Рис 3.12. Вікно редактора проєкту 3.3.2. Налаштування редактора - роздільна здатність ігри, зміна платформи Роздільна здатність ігри. По-перше необхідно встановити роздільну здатність ігри, для цього необхідно обрати вкладку

Цитирования: **0,01%** id: **80**

“Free Aspect”

під вкладкою

Цитирования: **0,01%** id: **81**

“Game”.

Додати та ввести значення кожного параметру. Для ширини (

Цитирования: **0,01%** id: **82**

“width”

) – 1280 та для висоти (

Цитирования: **0,01%** id: **83**

“height”

) – 720. Зміна платформи. В меню

Цитирования: **0,01%** id: **84**

“File”,

натиснути на

Цитирования: **0,01%** id: **85**

“Build Setting”

та обрати Android (рис. 3.13). Рис 3.13. Обрання платформи Категоризація активів. Необхідно категоризувати потрібні активи щоб уникнути складності при навігації по файлам. Animations – місце знаходження анімаційних контролерів та анімації Fonts – місце знаходження шрифтів Scenes – місце знаходження об’єктів сцени Scripts - місце знаходження скриптів Sounds – директорія аудіофайлів Sprites – місце знаходження всіх зображень та спрайтів 3.3.3. Створення головного меню ігрової програми Для того, щоб створити меню в Unity не буде потрібно багато часу. Для початку створимо ще одну сцену і назвемо її Menu. Для створення елементів інтерфейсу в Unity будемо використовувати такий компонент як Canvas, він є головним на сцені елементом і всі інші елементи, такі як кнопки, панелі і так далі створюються як дочірні до нього. Canvas відображається на сцені як великий прямокутник, він дозволяє легко керувати позиціонуванням елементів на сцені і має свою Event System. Елементи призначеного для користувача інтерфейсу відображаються на сцені в тій же послідовності, в якій вони представлені у вікні ієрархії.У головному меню на даному етапі розробки нам потрібно тільки два елементи, це кнопка Play, яка дозволить нам запускати тестовий рівень ігрового програми і кнопка Quit для виходу. Додати кнопку можна натиснувши правою кнопкою миші на сцені і вибравши в контекстному меню розділ UI Button (рис. 3.14). Рис. 3.14. Створення елементів інтерфейсу Після того як кнопки додані, ми можемо вибрати одну з них і подивитися які елементи управління вона має у вікні інспектора. У першій верхній частині вікна знаходяться елементи керування положенням кнопки, управління її розмірами, зумом, так само є можливість змінювати шар в якому ця кнопка буде відображатися (в нашому випадку всі елементи меню будуть лежати в шарі UI) і кут повороту. Далі йдуть елементи управління кольором кнопки, її текстурою і матеріалом якщо такі є і способом їх відображення[10]. Натиснувши на опцію Source Image, виберемо текстуру кнопки, яка буде відображатися на ній за замовчуванням. Ця текстура буде відповідати стану кнопки, коли над нею немає

курсору (тобто вона не в фокусі) і вона не в натиснутому стані. Рис. 3.15. Кнопка – Інспектор Після того як ми вибрали основну для кнопки текстуру, необхідно перейти до наступної частини вікна інспектора (рис. 3.15). У ній нас цікавить спосіб зміни текстур кнопки в залежності від її стану. Ця опція в Unity називається Transition і вона має кілька станів: None означає, що ніякої візуальної реакції на зміну станів кнопки не буде;Color Tint - в залежності від стану буде змінюватися колір кнопки; Sprite swa - кожномустаном буде відповідати своя текстура; Animation - стану кнопки будуть анімованими. У нашому випадку найбільше підходить варіант Sprite swap, так як у нас вже є заздалегідь заготовлені текстури для всіх станів кнопки. Після того як ми виберемо цей режим з'являться додаткові опції з яких нас цікавлять (рис. 3.16): Highlighted Sprite - текстура, яка буде завантажуватися кнопкою, в разі якщо вона у фокусі, тобто курсор знаходиться над нею; Pressed Sprite - текстура відповідна кнопці; Disabled Sprite – якщо кнопка має статус

Цитирования: **0,01%** id: **86**

"виключена",

то буде завантажена ця текстура. Рис. 3.16. Властивості кнопки в інспектора Так само в цьому вікні можна вказати метод, в який буде передаватися управління, в разі якщо кнопку натиснули. У об'єкта Button автоматично створюється нащадок Text, який ми будемо використовувати для створення написів на поверхні текстур кнопок. Для того щоб скористатися широкими можливостями цього об'єкта, ми повинні вибрати його в вікні ієрархії, у кожної кнопки буде свій індивідуальний об'єкт Text. Перейшовши у вікно інспектора, ми зможемо вказати використовуваний на наших кнопках шрифт (рис. 3.17) (ми будемо використовувати Carbon block), відрегулювати його розмір, налаштувати вирівнювання та колір[11]). Рис.3.17. Текст на кнопці Після того як ми створили першу кнопку, необов'язково проробляти всю процедуру створення і налаштування для інших кнопок. можна просто скопіювати вже створений об'єкт, задати йому нове ім'я, інший текст і текстуру (рис. 3.18). Використання копіювання, у разі коли потрібно створити кілька схожих за властивостями об'єктів, сильно спрощує процес створення програми та скорочує тимчасові витрати на розробку. Рис. 3.18. Сцена за меню Отже, ми створили меню. Тепер до кнопок цього меню потрібно прив'язати дії. Для це ми створимо новий скрипт і назвемо його NewGame. Додамо туди функції для старту гри і виходу. Для завантаження сцен в Unity служить Scene Manager, його ми будемо використовувати для завантаження головної сцени гри. На початковому етапі у нас буде всього один рівень, тому більше нічого нам не знадобиться. public static void NewGame() { Application.LoadLevel(

Цитирования: **0,01%** id: **87**

“GameScene”

) } Далі ми створюємо порожній ігровий об'єкт, назвемо його mainMenuObj і призначаємо йому наш скрипт (рис. 3.19). Рис. 3.19. Порожній об'єкт У властивостях створеної нами кнопки в інспектора можна задати дію, яке буде відбуватися при її натисканні, додаємо туди наш порожній об'єкт і функцію, яка буде викликатися при натисканні, в даному випадку, це playButton (рис. 3.20). Рис.3.20. Створення події Тепер при натисканні кнопки Start буде завантажуватися сцена з першим рівнем гри. Дії для інших кнопок меню аналогічні цим, створюємо функцію в скрипті Menu, відповідну кнопці, і призначаємо її в інспектора у властивостях цієї кнопки. 3.3.4. Створення контролерів Маючи створене головне меню потрібно створити контролер до гри. Контролер – це ігровий об’єкт який містить методи обробки та опис взаємодії компонентів. Для створення контролеру у вкладці

Цитирования: **0,01%** id: **88**

“GameObject”

необхідно обрати

Цитирования: **0,01%** id: **89**

“Create Empty”

– GameObject (

Цитирования: **0,01%** id: **90**

“ігровий об’єкт”

) з’явиться в області

Цитирования: **0,01%** id: **91**

“Hierarchy”.

(рис.3.21). Рис 3.21. Створення ігрового об’єкту Перш ніж працювати над GameObject, обов'язково потрібно скасувати усі трансформації, щоб уникнути конфліктів в майбутньому, див. рис.3.22. Рис 3.22. Скасування трансформації Новостворений ігровий об’єкт слід перейменувати у

Цитирования: **0,01%** id: **92**

“Game Controller”.

Після цього у директорії

Цитирования: **0,01%** id: **93**

“Scripts”

створимо нову папку

Цитирования: **0,01%** id: **94**

“Game Controllers”.

Створимо скрипт C# (рис.3.23), натиснувши праву кнопку миші та обравши

Цитирования: **0,03%** id: **95**

“Create – C# Script”.

Збережемо його як

Цитирования: **0,01%** id: **96**

“GameController”

– він буде обробляти всі збережені дані гри. Щоб уникнути помилки сценарію, необхідно переконатися, що ім'я сценарію не має пробілів, тому що C # - чутливий до пробілів. Також слід зберегти всі контролери всередині папки ігрових контролерів, для уникнення неправильного розташування. Рис 3.23. Створення скрипта C# Відкривши новостворений скрипт можна помітити готовий блок коду. По замовчуванню Unity створює частину коду для полегшення розробки. using System.Collections; using System.Collections.Generic; using UnityEngine; public class GameController : MonoBehaviour { // Use this for initialization void Start () { } // Update is called once per frame void Update () { } } Для початку роботи необхідно імпортувати важливі модулі: using System; using System.IO; using System.Runtime.Serialization.Formatters.Binary; public static GameController instance; public bool isGameStartedFirstTime; public bool isMusicOn; public bool isGameStartedInMainMenu; public int highScore; public int currentScore; public int currentLives; public int currentLevel = 1; private GameData data; void Awake(){ MakeInstance (); InitializeGameVariables (); } // Use this for initialization void Start () { } // Update is called once per frame void Update () { } void MakeInstance(){ if (instance != null) { Destroy (gameObject); } else { instance = this; DontDestroyOnLoad (gameObject); } } void InitializeGameVariables(){ Load (); if (data != null) { isGameStartedFirstTime = data.GetIsGameStartedFirstTime (); } else { isGameStartedFirstTime = true; } if (isGameStartedFirstTime) { isMusicOn = true; highScore = 500; isGameStartedFirstTime = false; data = new GameData (); data.SetIsGameStartedFirstTIme (isGameStartedFirstTime); data.SetHighScore (highScore); data.SetIsMusicOn (isMusicOn); Save (); Load (); } else { isMusicOn = data.GetIsMusicOn(); highScore = data.GetHighScore (); isGameStartedFirstTime = data.GetIsGameStartedFirstTime (); } } public void Save(){ FileStream file = null; try{ BinaryFormatter bf = new BinaryFormatter(); file = File.Create(Application.persistentDataPath +

Цитирования: **0,01%** id: **97**

"/data.dat"

); if(data != null){ data.SetIsGameStartedFirstTIme(isGameStartedFirstTime); data.SetIsMusicOn(isMusicOn); data.SetHighScore(highScore); bf.Serialize(file, data); } }catch(Exception e){ Debug.LogException (e, this); }finally{ if(file != null){ file.Close (); } } } public void Load(){ FileStream file = null; try{ BinaryFormatter bf = new BinaryFormatter(); file = File.Open(Application.persistentDataPath +

Цитирования: **0,01%** id: **98**

"/data.dat",

FileMode.Open); data = bf.Deserialize(file) as GameData; }catch(Exception e){ Debug.LogException (e, this); }finally{ if(file != null){ file.Close (); } } } [Serializable] class

GameData{ private bool isGameStartedFirstTime; private bool isMusicOn; private int highScore; public void SetIsGameStartedFirstTIme(bool isGameStartedFirstTime){ this.isGameStartedFirstTime = isGameStartedFirstTime; } public bool GetIsGameStartedFirstTime(){ return this.isGameStartedFirstTime; } public void SetIsMusicOn(bool isMusicOn){ this.isMusicOn = isMusicOn; } public bool GetIsMusicOn(){ return this.isMusicOn; } public void SetHighScore(int highScore){ this.highScore = highScore; } public int GetHighScore(){ return this.highScore; } Для того щоб скрипт запрацював необхідно долучити його до об'єкту Game Controller як компонент, перетягнувши його в Інспектор ігрового контролера або натиснувши

Цитирования: **0,01%** id: **99**

“Add component”

та указати скрипт GameController. 3.3.5. Створення музичного контролера Музичний контролер обробляє фонові звуки гри, саме тут ви завантажуєте ігрову звукову доріжку як масив. Так само, як і в ігровому контролі, потрібно створити новий GameObject, а потім назвати його як

Цитирования: **0,01%** id: **100**

"Music Controller".

Після цього слід створити новий скрипт C #, назвавши його як

Цитирования: **0,01%** id: **101**

"MusicController".

Примітка. Всі необхідні звуки вже знаходяться в папці Звуки. Всередині сценарію Music Controller створіть певну змінну, яку ми будемо використовувати для наших звуків.public static MusicController instance; public AudioClip backgroud; public AudioClip gameplay; public AudioClip coinRush; public AudioClip death; public AudioClip start; public AudioClip bite; [HideInInspector] public AudioSource audioSource; У методі

Цитирования: **0,01%** id: **102**

“start”

призначається змінна

Цитирования: **0,01%** id: **103**

“audioSource”

до компоненту AudioSource; audioSource = GetComponent AudioSource (); Після цього потрібно створити екземпляр сценарію; це перешкоджатиме сценарію припинення роботи при переході до інших сцен. void CreateInstance(){ if(instance != null){ Destroy (gameObject); }else{ instance = this; DontDestroyOnLoad (gameObject); } } Створимо метод Awake (

Цитирования: **0,01%** id: **104**

“пробудження”

), вбудований метод Unity, який при старті гри викликає будь-який сценарій. Викличемо CreateInstance () всередині методу пробудження. void Awake(){ CreateInstance (); } Після напишемо методи обробки для опрацювання звуку public void PlayMainMenuBg(){ if(backgroud){ audioSource.clip = backgroud; audioSource.loop = true; audioSource.Play (); } } public void PlayGameplaySound(){ if(gameplay){ audioSource.clip = gameplay; audioSource.loop = true; audioSource.Play (); } } public void PlayCoinRush(){ if(coinRush){ if(audioSource.isPlaying){ audioSource.clip = coinRush; audioSource.loop = true; audioSource.Play (); } } } public void PlayDeathSound(){ if(death){ if(!audioSource.isPlaying){ audioSource.clip = death; audioSource.loop = false; audioSource.Play (); } } } public voidPlayStartSound(){ if(death){ audioSource.clip = start; audioSource.loop = false; audioSource.Play (); } } public void StopAllSounds(){ if(audioSource.isPlaying){ audioSource.Stop (); } } Далі потрібно долучити сценарій до контролера музики як компонент; так само, як у контролера гри. Після цього додати новий компонент під назвою AudioSource, він керує звуками гри. Слід зняти прапорець Play On Wake, тому що це буде зроблено автоматично за допомогою сценарію. Потім з каталогу звуків перетягніть весь звук в масив AudioClips. Рис 3.24. Додавання звукового компоненту 3.4. Створення карти Ландшафт буде виконаний у вигляді карти, тобто реєстрового зображення з розставленими на ньому коллайдера. У ролі коллайдеров будем використовувати вбудований компонент 2D box collider. Для карти будемо використовувати asset, який можна взяти на вебсайті вільного ігрового арту[12]). Він представляється собою набір тайлів або елементів карти (рис. 3.25).

Рис. 3.25. Asset для створення карти Саму карту можна підготувати в будь-якому доступному графічному редакторі (наприклад: paint.net) з довільно виконаним ландшафту (рис. 3.26). Рис. 3.26. Підготовлена карта ландшафту Після створення карти поміщаємо її на сцену. Тепер за допомогою коллайдеров покажемо які місця в лабіринті будуть стінами і підлогою. Для цього додамо компонент 2D box collider, так як таких непрохідних поверхонь в лабіринті буде багато, то, відповідно, і компонентів теж треба буде додати стільки ж (рис. 3.27). Рис. 3.27. Налаштування коллайдера Після того як вони додані, потрібно зайти в кожен компонент, натиснути кнопку Edit bounding volume і виділити області, які будуть займати наші поверхні. Колайдери, використовуючи виділені інформуватимуть Unity про попадання в них об'єкта. Тепер при завантаженні рівня ми можемо запросити у Unity список всіх коллайдеров і використовувати їх у своїй функції перевірки зіткнень. 3.5. Висновки до розділу В даному розділі проаналізовано та обрані інструменти для розробки ігрового проєкту розроблено головні концепції та програмно реалізовано гру за допомогою мови програмування C# і платформи Unity3D. ВИСНОВКИ У кваліфікаційній роботі роботі було розглянуто аналіз та розробку ігрового проєкту на основі Unity3D. В роботі проведено огляд та аналіз популярних сучасних засобів для розробки ігор та їх представників, розбиття засобів на класи за їх базовими заявленими функціональними можливостями. Визначено оптимальні сфери використання для інструментальних засобів спираючись на аналіз процесу розробки прототипів. Визначено, що серед ігрових рушіїв у колі незалежних розробників, останнім часом, найбільшою популярністю користується Unity. Unity – ігровий рушій, зібравши в собі фізичний, графічний рушій, великий набір інструментів та широкий функціонал. Гарний баланс написаного коду та використання вбудованого інструментарію для налаштування елементів гри. Зручна візуалізація продукту та чудові можливості відладки як коду так і поведінки гри в цілому. Великі можливості для повторного використання коду та вбудована оптимізація графіки, фізики. Unity є найпотужнішим засобом для розробки гри серед розглянутих і надто легким для використання представленого функціоналу. Він повністю задовольняє нас за своїми базовими характеристиками до того має потужну підтримку товариства, що значно знижує поріг входження та номінально обіцяє широкий спектр функціоналу та кросплатформеність. Проаналізовано та обрані інструменти для розробки ігрового проєкту розроблено головні концепції та програмно реалізовано гру за допомогою мови програмування C# і платформи Unity3D. Цей додаток не є закінченим продуктом, так як вимагає наповнення професійно виконаним графічним контентом, але при відповідному доопрацюванні може зайняти свою нішу на ринку ігор. Крім того були розробленні інфологічна та даталогічна моделі бази даних. СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ Abel Gomes. Geometric Computing. Lecture 4 – Interpolation methods [Электронный ресурс] / Abel Gomes. – 2010. – Режим доступа: http://www.di.ubi.pt/~agomes/tcg/lectures/04-lecture.pdf. About CMake [Электронный ресурс]. Режим доступа: https://cmake.org/overview. Andrew M. Understanding Open Source and Free Software Licensing / Andrew M. St. Laurent. – O'Reilly Media, 2004. – 207 с. Aung Sithu Kyaw. Unity 4.x Game AI Programming / Aung Sithu Kyaw, Clifford Peters, Thet Naing Swe. – Packt Publishing, 2013. – 232 с. Bevilacqua Fernando. Understanding Steering Behaviors [Электронный ресурс] / Fernando Bevilacqua. – 2013. – Режим доступа: https://gamedevelopment.tutsplus.com/tutorials/understanding-steering-behaviors-movement-manager-gamedev-4278. Bourg David M. Physics for Game Developers, 2nd Edition / David M Bourg, Bryan Bywalec. – O'Reilly Media, 2013. – 578 с. Corless Robert. A Graduate Introduction to Numerical Methods – From the Viewpoint of Backward Error Analysis / Robert Corless, Nicolas Fillion. – Springer Science & Business Media, 2013. – 869 с. Despain. 100 Principles of Game Design / Despain, Wendy. – New Riders, 2012. – 240 с. DevGam сравнение игровых движков URL: http://devgam.com/sravnenie-igrovyx-dvizhkov-kakoj-vybrat . Engelbert Roger. Cocos2d-x by Example Beginner's Guide / Roger Engelbert. – Packt Publishing Ltd, 2013. – 246 с. Erin Catto. World Class. Simulation [Электронный ресурс] / Erin Catto // Box2D v2.3.0 User Manual. – 2013. – Режим доступа: http://box2d.org/manual.pdf. Fiedler Glenn. Fix Your Timestep! [Электронный ресурс] / Glenn Fiedler. – 2016. – Режим доступа: http://gafferongames.com/game-physics/fix-your-timestep. Global mobile OS market share in sales to end users from 1st quarter 2009 to 1st quarter 2016 [Электронный ресурс] / 2016. – Режим доступа: https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems Heaton Jeff. Artificial Intelligence for Humans, Volume 1: Fundamental Algorithms / Jeff Heaton. – Heaton Research, 2013. – 147 с. ISO/IEC 14882:2014. Information technology – Programming languages – C++ / ISO, 2014. – Режим доступа: https://www.iso.org/standard/64029.html. Kehoe Donald. Designing Artificial Intelligence for Games [Электронный ресурс / Donald Kehoe. –

2015. – Режим доступа: https://software.intel.com/en-us/articles/designing-artificial-intelligence-for-games-part-1. Korth Andy. Chipmunk 7 released – Pro tools open sourced [Электронный ресурс] / Andy Korth. – Howling Moon Software, 2015. – Режим доступа: http://howlingmoonsoftware.com/wordpress/chipmunk-7-released-pro-tools-open-sourced. List of game engines and frameworks URL: https://en.wikipedia.org/wiki/List_of_game_engines (дата звернення 21.03.2019). List of game engines URL: https://80.lv/articles/the-most-comprehensive-list- of- game-engines/ (дата звернення 27.03.2019). Matsuura Akihiro. Cocos2d-x Cookbook / Akihiro Matsuura. – UK: Packt Publishing, 2015. – 250 с. New mobile game statistics every game publisher should know in 2016 [Электронный ресурс] / 2016. – Режим доступа: https://www.surveymonkey.com/business/intelligence/mobile-game-statistics. O'Haver T. Smoothing [Электронный ресурс] / O'Haver T. – 2012. – Режим доступа: http://terpconnect.umd.edu/~toh/spectrum/Smoothing.html. OMG. OMG Unified Modeling Language (OMG UML), Superstructure Version 2.5 / OMG, 2015. – Режим доступа: http://www.omg.org/spec/UML/2.5/PDF. Parberry I. Introduction to Game Physics with Box2D / Ian Parberry. – CRC Press, 2013. – 275 с. Rabin Steven. Game AI Pro: Collected Wisdom of Game AI Professionals / Steven Rabin. – A K Peters/CRC Press, 2013. – 626 с. Scott Chacon. Pro Git [Электронный ресурс] / Scott Chacon, Ben Straub. – 2014. – Режим доступа: https://git-scm.com/book/en/v2. Scott Meyers. Effective Modern C++ / Scott Meyers. – 2014. – 334 с. Scott Robertson. How to design: Concept Design Process, Styling, Inspiration, and Methodology. – Titan Books, 2014. – 176 с. Shah Ryan. Blueprints Master the Art of Unreal Engine 4 – Blueprints / Ryan Shah. – CreateSpace Independent Publishing Platform, 2014. – 122 с. Unity store URL: https://store.unity.com/ru?_ga=2.200322542.1765686190.1560285797-130111505.1547668569. Видео уроки по Unity3D URL: https://www.youtube.com/watch?v=m0sKo81DZME . Галёнкин Сергей. Маркетинг игр / Сергей Галёнкин. – 2014. – 78 с. Гамма Эрих. Приемы объектно-ориентированного проектирования. Паттерны проектирования / [Эрих Гамма, Ричард Хелм, Ральф Джонсон, Джон Влиссидес]. – Питер, 2016. – 366 с. Интегрированная среда разработки Visual Studio [Электронный ресурс] / Microsoft, 2017. – Режим доступа: https://www.visualstudio.com/ru/vs/. Макконнелл Стив. Совершенный код. Мастер-класс / Стив Макконнелл. – Русская Редакция, 2017. – 896 с. Руководство Unity URL: https://docs.unity3d.com/ru/current/Manual/index.html (дата звернення 18.02.2019). Руководство по программированию на C# URL: https://docs.microsoft.com/ru- ru/dotnet/articles/csharp/programming- guide/index. Русскоязычный форум Unity3d URL: http://unity3d.ru . Смит Джейсон Мак-Колм. Элементарные шаблоны проектирования / Джейсон Мак-Колм Смит. – М.:

Цитирования: **0,01%** id: **105**

«Вильямс»,

2012. – 304 с. Фримен Элизабет. Паттерны проектирования / [Элизабет Фримен, Эрик Фримен, Кэти Сиерра, Берт Бейтс]. – Питер, 2013. – 656 с. Хокинг Д. Unity in Action: Multiplatform Game Development in C# with Unity 5 : Manning Publications Company. 2015 – 352с. Чакон Скотт. Git для профессионального программиста / Скотт Чакон, Бен Штрауб. – Питер, 2016. – 496 с. ДОДАТКИ Додаток 1. Даталогічна модель бази даних Додаток 2. Лістинг програми using UnityEngine; using System.Collections; public class CubeJump: MonoBehaviour { public static bool jump, nextBlock; public GameObject mainCube, buttons, lose_buttons; private bool animate, lose; private float scratch_speed = 0.5f, startTime, yPosCube; public static int count_blocks; void Start () { StartCoroutine (CanJump ()); } void FixedUpdate () { if (animate && mainCube.transform.localScale.y 0.4f) PressCube (-scratch_speed); else if (!animate && mainCube != null) { if (mainCube.transform.localScale.y 1f) PressCube (scratch_speed * 3f); else if (mainCube.transform.localScale.y != 1f) mainCube.transform.localScale = new Vector3 (1f, 1f, 1f); } if (mainCube != null) { if (mainCube.transform.position.y -7.5f) { Destroy (mainCube, 0.5f); print (

Цитирования: **0,01%** id: **106**

"Player Lose"

); lose = true; } } if (lose) PlayerLose (); } void PlayerLose () { buttons.GetComponent Scrollobjectg ().speed = 5f; buttons.GetComponent Scrollobjectg ().checkPos = 0; if (!lose_buttons.activeSelf) lose_buttons.SetActive (true); lose_buttons.GetComponent Scrollobjectg ().checkPos = 70; } void OnMouseDown () { if (nextBlock && mainCube.GetComponent Rigidbody ()) { animate = true; startTime = Time.time; yPosCube =

mainCube.transform.localPosition.y; } } void OnMouseUp () { if (nextBlock && mainCube.GetComponent Rigidbody ()) { animate = false; //Jump jump = true; float force, diff; diff = Time.time - startTime; if (diff 3f) force = 190 * diff; else force = 280f; if (force 60) force = 60; mainCube.GetComponent Rigidbody ().AddRelativeForce (mainCube.transform.up * force); mainCube.GetComponent Rigidbody ().AddRelativeForce (mainCube.transform.right * -force); StartCoroutine (checkCubePos ()); nextBlock = false; } } void PressCube (float force) { mainCube.transform.localPosition += new Vector3 (0f, force * Time.deltaTime, 0f); mainCube.transform.localScale += new Vector3 (0f, force * Time.deltaTime, 0f); } IEnumerator checkCubePos () { yield return new WaitForSeconds (1.5f); if (yPosCube == mainCube.transform.localPosition.y) { print (

Цитирования: **0,01%** id: **107**

"Player Lose "

); lose = true; } else { while (!mainCube.GetComponent Rigidbody ().IsSleeping ()) { yield return new WaitForSeconds (0.05f); if (mainCube == null) break; } if (!lose) { nextBlock = true; count_blocks++; print (

Цитирования: **0,01%** id: **108**

"Next One"

); mainCube.transform.localPosition = new Vector3 (1.8f, mainCube.transform.localPosition.y, mainCube.transform.localPosition.z); mainCube.transform.eulerAngles = new Vector3 (0f, mainCube.transform.eulerAngles.y, 0f); } } } IEnumerator CanJump () { while (!mainCube.GetComponent Rigidbody ()) yield return new WaitForSeconds (0.05f); yield return new WaitForSeconds (1f); nextBlock = true; } } DetectClicks using UnityEngine; using UnityEngine.UI; using System.Collections; public class Detectclicks: MonoBehaviour { public GameObject [] cubes; public GameObject buttons, m_cube, spawn_blocks; public Text playTxt, gameName, record; public Animation cubes_anim, block; public Light dirLight, dirLight_2; private bool clicked; void Update () { if (clicked && dirLight.intensity != 0) dirLight.intensity -= 0.03f; if (clicked && dirLight_2.intensity = 1.05f) dirLight.intensity -= 0.025f; } void OnMouseDown () { if (!clicked) { StartCoroutine (Delcubes ()); clicked = true; // Works only ones playTxt.gameObject.SetActive (false); record.gameObject.SetActive (true); gameName.text =

Цитирования: **0,01%** id: **109**

"0"

; buttons.GetComponent Scrollobjectg ().speed = -10f; buttons.GetComponent Scrollobjectg ().checkPos = -130f; m_cube.GetComponent Animation ().Play (

Цитирования: **0,01%** id: **110**

"StartGameCube"

); StartCoroutine (CubeToBlock ()); m_cube.transform.localScale = new Vector3 (1f, 1f, 1f); cubes_anim.Play (); } } IEnumerator Delcubes () { for (int i = 0; i 3; i++) { yield return new WaitForSeconds (0.3f); cubes [i].GetComponent FallCube ().enabled = true; } spawn_blocks.GetComponent SpawnBlocks ().enabled = true; } IEnumerator CubeToBlock () { yield return new WaitForSeconds (m_cube.GetComponent Animation ().clip.length + 0.5f); block.Play (); //add rigid component m_cube.AddComponent Rigidbody (); } } Score using UnityEngine; using UnityEngine.UI; using System.Collections; public class Score: MonoBehaviour { public Text record; private Text txt; private bool gameStart; void Start () { record.text =

Цитирования: **0,01%** id: **111**

"TOP: "

+ PlayerPrefs.GetInt (

Цитирования: **0,01%** id: **112**

"Record"

).ToString (); txt = GetComponent Text (); CubeJump.count_blocks = 0; } void Update () { if (txt.text ==

Цитирования: **0,01%** id: **113**

"0"

) gameStart = true; if (gameStart) { txt.text = CubeJump.count_blocks.ToString (); if (PlayerPrefs.GetInt (

Цитирования: **0,01%** id: **114**

"Record"

) CubeJump.count_blocks) { PlayerPrefs.SetInt (

Цитирования: **0,01%** id: **115**

"Record",

CubeJump.count_blocks); record.text =

Цитирования: **0,01%** id: **116**

"TOP: "

+ PlayerPrefs.GetInt (

Цитирования: **0,01%** id: **117**

"Record"

).ToString (); } } } } SpawnBlock using UnityEngine; using System.Collections; public class SpawnBlocks: MonoBehaviour { public GameObject block, allCubes; private GameObject blockInst; private Vector3 blockPos; private float speed = 7f; private bool onPlace; void Start () { spawn (); } void Update () { if (blockInst.transform.position != blockPos && !onPlace) blockInst.transform.position = Vector3.MoveTowards (blockInst.transform.position, blockPos, Time.deltaTime * speed); else if (blockInst.transform.position == blockPos) onPlace = true; if (CubeJump.jump && CubeJump.nextBlock) { spawn (); onPlace = false; } } float RandScale () { float rand; if (Random.Range (0, 100) 80) rand = Random.Range (1.2f, 2f); else rand = Random.Range (1.2f, 1.5f); return rand; } void spawn () { blockPos = new Vector3 (Random.Range (0.7f, 1.7f), -Random.Range (0.6f, 3.2f), -5.8f); blockInst = Instantiate (block, new Vector3 (9f, -5f, -5.8f), Quaternion.identity) as GameObject; blockInst.transform.localScale = new Vector3 (RandScale (), blockInst.transform.localScale.y, blockInst.transform.localScale.z); blockInst.transform.parent = allCubes.transform; } } MovesCube using UnityEngine; using System.Collections; public class MovesCube: MonoBehaviour { private bool moved = true; private Vector3 target; void Start () { target = new Vector3 (-4.66f, 4.02f, 0.16f); } void Update () { if (CubeJump.nextBlock) { if (transform.position != target) transform.position = Vector3.MoveTowards (transform.position, target, Time.deltaTime * 5f); else if (transform.position == target && !moved) { target = new Vector3 (transform.position.x - 5f, transform.position.y + 5f, transform.position.z); CubeJump.jump = false; moved = true; } if (CubeJump.jump) moved = false; } } } DeleteBlocks using UnityEngine; using System.Collections; public class DeleteBlocks: MonoBehaviour { void OnTriggerEnter (Collider other) { if (other.tag ==

Цитирования: **0,01%** id: **118**

"Cube"

) Destroy (other.gameObject); } } Buttons using UnityEngine; using UnityEngine.SceneManagement; using System.Collections; public class BUTTONS: MonoBehaviour { void OnMouseDown() { transform.localScale = new Vector3(1.1f, 1.1f, 1.1f); } void OnMouseUp() { transform.localScale = new Vector3 (1f, 1f, 1f); } void OnMouseUpAsButton () { switch (gameObject.name) { case

Цитирования: **0,01%** id: **119**

"Restart":

SceneManager.LoadScene (

Цитирования: **0,01%** id: **120**

"main"

); break; case

Цитирования: **0,01%** id: **121**

"FaceBook":

Application.OpenURL (

Цитирования: **0,01%** id: **122**

"https://facebook.com"

); break; } } } FallCube using UnityEngine; using System.Collections; public class FallCube: MonoBehaviour { void Start () { Destroy (gameObject, 1f); } void Update () { transform.position

+= new Vector3 (0, 0.2f, 0); } } ScrollObject using UnityEngine; using System.Collections; public class Scrollobjectg: MonoBehaviour { public float speed = 5f, checkPos = 0f; private RectTransform rec; void Start() { rec = GetComponent RectTransform (); } void Update() { if (rec.offsetMin.y != checkPos) { rec.offsetMin += new Vector2 (rec.offsetMin.x, speed); rec.offsetMax += new Vector2 (rec.offsetMax.x, speed); } } } SpawnStar using UnityEngine; using System.Collections; public class SpawnStar: MonoBehaviour { public GameObject star; void Start () { StartCoroutine (spawn ()); } IEnumerator spawn () { while (true) { Vector3 pos = Camera.main.ScreenToWorldPoint (new Vector3 (Random.Range (0, Screen.width), Random.Range (0, Screen.height), Camera.main.farClipPlane / 2)); Instantiate (star, pos, Quaternion.identity); yield return new WaitForSeconds (5.01f); } } } StarAnimation using UnityEngine; using System.Collections; public class StarAnimation: MonoBehaviour { private SpriteRenderer star; void Start () { star = GetComponent SpriteRenderer (); Destroy (gameObject, 5f); } void Update () { star.color = new Color (star.color.r, star.color.g, star.color.b, Mathf.PingPong (Time.time / 2.5f, 1.0f)); } } TextFade using UnityEngine; using UnityEngine.UI; using System.Collections; public class textFade: MonoBehaviour { private Text txt; private Outline oLine; void Start () { txt = GetComponent Text (); oLine = GetComponent Outline (); } void Update () { txt.color = new Color (txt.color.r, txt.color.g, txt.color.b, Mathf.PingPong (Time.time / 2.5f, 1.0f)); oLine.effectColor = new Color (oLine.effectColor.r, oLine.effectColor.g, oLine.effectColor.b, txt.color.a - 0.3f); } }

Заявление об ограничении ответственности:

Этот отчет должен быть правильно истолкован и проанализирован квалифицированным специалистом, который несет ответственность за оценку!

Любая информация, представленная в этом отчете, не является окончательной и подлежит ручному просмотру и анализу.
Пожалуйста, следуйте инструкциям: Рекомендации по оценке

Детектор Плагиата - Ваше право на оригинальность! □ SkyLine LLC