Федеральное государственное бюджетное

образовательное учреждение высшего образования

«Финансовый университет при Правительстве Российской Федерации»

Лицей Финуниверситета

Проектная работа

Разработка образовательного приложения «HistoryZen: повторяем даты к ЕГЭ по истории» под ОС Android

**Работу выполнили:** ученики 10 Б класса

технологического профиля

Орусский В. Р. и Ляшенко Я. И

**Руководитель проекта:**

Алиев С. А. – учитель информатики и ИКТ

Урмайкин В. М. – учитель обществознания и истории

Москва

2020

**Оглавление**

[Введение 3](#_Toc38280307)

[Глава 1. Концепция, инструментарий, описание продукта 5](#_Toc38280308)

[1.1. Пельманизм и «Memory Game» 5](#_Toc38280309)

[1.2. «Memory Game» как основная механика приложения 5](#_Toc38280310)

[1.3. Выбор платформы, программного обеспечения и языка программирования 6](#_Toc38280311)

[1.4. Структура и функционал приложения 12](#_Toc38280312)

[1.4.1. Сцены, объекты, скрипты, ресурсы 12](#_Toc38280313)

[1.4.2. Производительность, системные требования 14](#_Toc38280314)

[1.4.3. Cценарий использования конечным пользователем 14](#_Toc38280315)

[Глава 2. Программные коды скриптов 18](#_Toc38280316)

[2.1. DateNumber 18](#_Toc38280317)

[2.2. Limits\_by\_Period 18](#_Toc38280318)

[2.3. Record\_printing 21](#_Toc38280319)

[2.4. AddButtons 21](#_Toc38280320)

[2.5. GameController 22](#_Toc38280321)

[2.6. BackGround 24](#_Toc38280322)

[2.7. Counter 27](#_Toc38280323)

[2.8. Writer 28](#_Toc38280324)

[2.9. ScreenChange 28](#_Toc38280325)

[Заключение 29](#_Toc38280326)

[Список литературы 32](#_Toc38280327)

# Введение

На данный момент самым распространенным способом поступления в ВУЗы в России является сдача Единого Государственного Экзамена. Выпускники, сдающие историю, в свою очередь сталкиваются с некоторыми трудностями при запоминании учебного материала, т.к. изучение этого предмета предполагает заучивание большого количества информации (исторические деятели, даты, события и связи между оными). Сейчас существует множество способов облегчить данный процесс: мнемонические техники, специальные приложения, структурированные курсы.

Актуальность работы состоит в том, что разные люди воспринимают материал по-разному, следовательно, они нуждаются в разных методах обучения, каждый из которых может быть эффективен для определенной группы. Мы создали приложение, которое может стать полезным помощником в изучении истории, применяя известную методику, хорошо проявившую себя в сфере образования и развлечений.

Целью данного проекта является создание приложения под ОС Android, помогающего эффективно повторять ранее изученные даты и события, используя метод, распространённое название которого – «Memory Game». В ходе работы были проведены поиск и систематизация сведений об истории «Memory Game». Также было изучено большое количество технической информации, приобретены практические навыки программирования.

Для реализации проекта были определены следующие задачи:

1. Определить актуальность продукта.
2. Определить сроки работы.
3. Отобрать материал для приложения.
4. Разработать дизайн.
5. Обучиться основам разработки на движке Unity.
6. Разработать и протестировать бета - версию приложения.
7. Опубликовать приложение в магазине приложений «Google Play».

Практическая значимость проекта заключается в том, что будущие выпускники смогут использовать наше приложения для закрепления дат и событий в удобном игровом формате. Все необходимые для сдачи экзамена даты собраны в одном месте и сгруппированы по историческим периодам, таким образом пользователи получают возможность повторять даты только того периода, который им необходим, и избавляются от необходимости искать их в других источниках. Платформа Android делает приложение доступным для широкой аудитории учащихся, а формат игры обеспечивает более высокий интерес к процессу, чем перечитывание учебников и конспектов.

# Глава 1. Концепция, инструментарий, описание продукта

## Пельманизм и «Memory Game»

Пельманизм – система умственного развития, разработанная в 1890-х годах Уильямом Джозефом Энневером. Он рекламировался как система обучения, которая укрепляет и развивает разум так же, как физическая подготовка укрепляет тело. Энневер обещал вылечить целый ряд проблем, таких как забывчивость, депрессия, фобия, прокрастинация, повысить обучаемость, увеличить объем кратковременной памяти, развить аналитические способности.

Пельманизм практиковался и пропагандировался бывшим британским премьер-министром Гербертом Асквитом, сэром Робертом Баден-Пауэллом (основателем движения бойскаутов), романистом сэром Райдером Хаггардом, драматургом Джеромом К. Джеромом и композитором Дамой Этель Смит, а также тысячами менее известных британцев.

В современном мире к Пельманизму относятся скептически, однако некоторые элементы системы Уильяма Энневера все же нашли свое место. Один из таких элементов – игра «Memory Game»

Memory Game – это игра, призванная развивать краткосрочную память. Она реализуется с помощью карточек, которые по умолчанию лежат лицевой стороной вниз. В ходе игры участники переворачивают карточки, просматривают их содержимое и пытаются найти две такие, которые совпадают по определённым признакам (это могут быть как картинки, так и надписи).

* 1. **«Memory Game» как основная механика приложения**

Игра «Memory game» обычно используется для развития кратковременной памяти у детей, а также частично для их обучения. Взрослые же люди используют такие игры ради развлечения, однако мы решили внедрить данный метод в образовательное приложение для старшеклассников.

Главный плюс этого метода в том, что он сочетает в себе сразу несколько аспектов: развлечение, обучение и развитие. Предполагается, что процесс закрепления изученного материала во время игры будет проходить легче, чем при использовании традиционных методов. Также полезен эффект развития кратковременной памяти: для успешного завершения игры необходимо постоянно запоминать текст на карточках. Механика «Memory Game» идеально подходит и с практической точки зрения: человек должен знать все элементы, находящиеся с обратных сторон карточек, иначе он не сможет понять, правильную ли пару он собрал, и игра теряет смысл. Таким образом, приложение действительно помогает именно повторять уже изученное, а не учить с нуля, что является изначальной идеей.

* 1. **Выбор платформы, программного обеспечения и языка программирования**

На данный момент на рынке существуют 2 платформы, которые используются в абсолютном большинстве смартфонов: Android и iOS. Нам необходимо было добиться максимального охвата аудитории, поэтому платформа выбиралась только исходя из ее популярности. По данным источника Bloomberg за 2019 год в России смартфонами на OS Android пользовались 73% людей, что фактически составляет больше двух третьих от общего количества. Таким образом было принято решение делать приложение под Android.

Следующим вопросом является выбор игрового движка. Игровой движок – это фундамент любой игры, которая хотя бы чуть-чуть сложнее классической «Змейки». Разработка даже простой игры требует большого количества сил и времени на реализацию функционала, который не относится напрямую к геймплею. Все игровые проекты используют примерно одинаковые шаблоны и компоненты, поэтому сторонние разработчики совместно разрабатывают различные библиотеки и движки, в частности, для создания игр под OS Android. Другими словами, такое программное обеспечение позволяет разработчикам не тратить время на создание одного и того же (не нужно «изобретать велосипед»), а также значительно снижает порог входа, потому что, например, передать необходимые параметры функциям отрисовки 3D-графики намного проще, чем самостоятельно написать эти функции, используя серьезный математический аппарат и большой опыт в программировании.

Игровые движки серьезно облегчают процесс разработки, позволяя сконцентрироваться на самом главном – реализации игровой механики. Использование движков также уменьшает объем тестирования готовой игры и предоставляет некоторые дополнительные функции, такие как, например, оптимизация под конкретную аппаратную платформу.

Движки отличаются друг от друга по предоставляемым разработчику возможностям: одни подходят для разработки 2D игр, другие – для 3D, одни представляют из себя некий конструктор с готовым набором функций и объектов, которым может воспользоваться даже ребенок, другие, ориентированные на создание более сложных проектов, предоставляют разработчикам широкий функционал и большую свободу действий.

Также движки отличаются по охвату доступных платформ. Есть движки под одну платформу, но также существует много кроссплатформенных движков.

Выбор движка является ключевым вопросом. Игровые движки с обширным функционалом обычно массивны и тяжелы в использовании, по сравнению с более «легкими». Но так как большинство разработчиков игр – профессионалы, многофункциональные движки более востребованы, поэтому они имеют крупные сообщества. Крупные сообщества, в свою очередь – это большие базы знаний и готовых решений, а также возможность быстро получать ответы на неизбежно возникающие в процессе разработки вопросы.

В дальнейшем рассмотрении будут участвовать только популярные игровые движки, позволяющие разрабатывать игры под OS Android.

Unity – кроссплатформенный игровой движок от компании Unity Technologies. Распространяется на платной и бесплатной основе в зависимости от версии: Student – версия для иностранных студентов. Бесплатна, Personal – бесплатная версия для тех, чей продукт имеет годовой оборот средств менее 100 000$/год, Plus и Pro – для организаций с оборотом от 100 000$/год до 200 000$/год и более 200 000$/год соответственно. Стоимость – 399$/год и 1800$/год соответственно.

На данный момент является самым популярным игровым движком.

Юнити использует компонентно-ориентированный подход: любая игра разделена на сцены, внутри которых присутствуют объекты, которым присвоены различные компоненты. Например, если нам нужно создать куб, то сначала создается пустой GameObject, к которому прикрепляется компонент, задающий его геометрию.

В подавляющем большинстве случаев необходимо создавать свои компоненты. Такие компоненты называются скриптами и пишутся на одном из двух языков: С# или UnityScript, который является модификацией JavaScript.

Такой подход хорошо себя показывает при разработке малых и средних проектов, но в играх ААА-класса появляется проблема излишней запутанности структуры и нарушения связей между скриптами (она, впрочем, решается грамотным проектированием). Маленькие проекты, созданные в Unity, весят больше и работают медленнее, чем могли бы, будь они созданы на другом движке.

Присутствует магазин с большим выбором ассетов и иногда даже готовых проектов. Магазин ассетов в Unity – самый крупный среди аналогичных. Существует великое множество обучающих материалов: книги, видео, курсы, статьи (Unity Learn – официальный обучающий сайт компании).

Игры, созданные на Unity: Cuphead, Hearthstone, Pillars of Eternity, Subnautica.

Unity подходит для разработки проектов любой сложности для любой платформы, прост в использовании, удобен, но не очень хорошо оптимизирован и уступает в 3D графике таким движкам, как, например, Unreal Engine.

Unreal Engine – кроссплатформенный игровой движок от компании Epic Games. Есть два варианта лицензирования: Creators License и Publishing License. Creators License – для разработчиков вне игровой индустрии, учеников и учителей, создателей бесплатных игр или приложений личного пользования. Полностью бесплатна. Publishing License – для разработчиков, собирающихся монетизировать свои игры. Необходимо платить 5% роялти от проекта, но только если доход составляет более 3000$ за квартал.

Unreal Engine – мощный игровой движок для создания современных AAA-игр. Язык программирования – С++. Редактор UnrealEd (Unreal Editor) позволяет редактировать уровни и ресурсы, манипулировать 3D-моделями, изменяя их свойства в реальном времени.

Все элементы игрового движка представлены в виде объектов, имеющих набор характеристик, и класса, который определяет доступные характеристики. В свою очередь, любой класс является «дочерним» классом object. Среди основных классов и объектов можно выделить следующие:

1. Актёр (actor) (жаргонное выражение, принятое в среде 3D-моделлеров, разработчиков, заменяющее выражение «действующий объект» или «субъект» — являющееся переводом [англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) Actor — тот, кто действует — [англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) action) — родительский класс, содержащий все объекты, которые имеют отношение к игровому процессу и имеют пространственные координаты.
2. Пешка (pawn) — физическая модель игрока или объекта, управляемого искусственным интеллектом. Название происходит от англ. pawn — тот, кем манипулируют (или пешка, поэтому такой объект без какой-либо модели выглядит как пешка). Метод управления описан специальным объектом, такой объект называется контроллером. Контроллер искусственного интеллекта описывает лишь общее поведение пешки во время игрового процесса, а такие параметры как «здоровье» (количество повреждений, после которых пешка перестаёт функционировать) или, например, расстояние, на котором пешка обращает внимание на звуки. задаются для каждого объекта отдельно.
3. Мир, уровень (world, game level) — объект, характеризующий общие свойства «пространства», например, силу тяжести и туман, в котором располагаются все «актёры». Также может содержать в себе параметры игрового процесса, как, например, игровой режим, для которого предназначен уровень.

Присутствует встроенный инструмент визуального программирования, позволяющий написать игру без единой строчки кода – BluePrint.

Последнее поколение движка содержит современные средства обработки 3D графики: система расчета освещения Unreal Lightmass, система пост-обработки, реалистичная симуляция физики, создание больших карт ландшафтов, инструменты анимации персонажей, создание кат-сцен и многое другое. Таким образом движок позволяет создавать проекты с высококлассной 3D графикой, но для их запуска потребуются значительные мощности.

Игры, созданные на Unreal Engine: We Happy Few, Gears of War 4, Borderlands 3, Unreal Tournament.

Этот движок послужит прекрасной платформой для создания крупных ААА-игр с современной 3D графикой и спецэффектами.

libGDX – open-source фреймворк от разработчика BadLogicGames. Написан на Java и C++. Разработка игр в нем ведется так же на Java, однако подходит любой язык, который можно запустить на JVM. Поддерживает как 2D, так и 3D. Имеется встроенная библиотека быстрой математики. Поддерживает большое количество сторонних инструментов.

Фреймворк работает с редактором карт Tiled, физическим движком box2d, шрифтами .ttf. Неплохая встроенная библиотека UI. Инструменты визуального программирования отсутствуют. Есть сторонние расширения для работы с соц. сетями и внутриигровыми покупками.

libGDX – высокопроизводительный игровой движок для игр начального уровня. Популярных проектов на нем создано0 не было: фреймворк используется для создания простых коммерческих или ученических проектов.

Cocos2d-x - кросс-платформенный open-source фреймворк, используемый для разработки интерактивных приложений и игр (преимущественно для мобильных устройств). Написан на С++, Lua, JavaScript. Разработка проектов в нем ведется тоже на С++. Так же, как и Unity, использует компонентно-ориентированный подход: игра состоит из сцен и размещенных на них спрайтах, которые могут восприниматься фреймворком в качестве текстуры или физического объекта. В основе Cocos2d-x лежат такие объекты: Scene, Node, Sprite, Menu и Action. Перемещающиеся объекты – это спрайты, статичные – ноды. Menu – объекты системы UI (label, button и т.д.).

Игры, разработанные на Cocos2d-x: Geometry Dash, Hill Climb Racing, Badland.

Cocos2d-x –хороший выбор для создания 2D проектов низкой и средней сложности.

Выбор движка основывался на нескольких факторах: популярность (наличие крупного сообщества), лицензирование, удобство, наличие широкого инструментария для работы с UI, перспективность и востребованность, используемый язык программирования.

Вопрос лицензирования никак не повлиял на выбор движка: два рассмотренных фреймворка являются open-source проектами, Unity требует покупки платной версии только по достижению годового оборота проекта в 100 000$, а Unreal Engine требует перечисления 5% роялти, но только при заработке свыше 3000$ за квартал, что абсолютно недостижимо для нашего проекта. Таким образом, любой из этих движков в наших условиях является бесплатным.

В проекте нет 3D графики, поэтому Unreal Engine вышел из рассмотрения, как чересчур громоздкий для выполнения столь маленького проекта. libGDX не подходит по причине некой устарелости и высокого порога вхождения.

Выбор стоял между Unity и Cocos2d-x, но в конечном итоге Unity был выбран по следующим причинам: крупнейшее сообщество разработчиков, широкое коммерческое использование (осле завершения проекта есть возможность дальше развиваться в направлении game-delelopment’а), возможность глубокой настройки элементов UI (из которых преимущественно стоит игра), которая определяет итоговый внешний вид проекта и использование C# - мощного современного языка программирования, использующегося во многих сферах (от веб-разработки и геймдева до крупных enterprise проектов),

В качестве среды разработки был выбран Microsoft Visual Studio, как официальное решение от крупной компании. Он очень удобен в использовании, имеет приятный графический интерфейс, а также широкий инструментарий для отладки.

## Структура и функционал приложения

### Сцены, объекты, скрипты, ресурсы

В параграфе о выборе движков было описано строение Unity и его компонентно-ориентированный подход. Таким образом наш проект представляет из себя следующий набор сцен, объектов и скриптов (см. табл.1).

Таблица 1



Всего в проекте 9 скриптов:

1. DateNumber. Управляет кнопками «плюс» и «минус» на StartScreen, а также количеством дат.
2. Limits\_by\_Period. Позволяет выбрать исторический период для деления дат на группы.
3. Record\_printing. Ведет счет найденных пар за сутки и за все время.
4. AddButtons. Отвечает за размещение кнопок и содержимое их дочерних объектов «Text» в сцене Game.
5. GameController. Реализует логику «Memory Game» в сцене Game.
6. BackGround. Случайно устанавливает задний фон из имеющегося списка спрайтов.
7. Counter. «Хранилище» глобальных переменных: выбранный период, счет за день и за все время, случайный номер для выбора фонового спрайта, выбранное количество дат.
8. Writer. Выводит на FinishScreen пары дат и событий, собранные на сцене Game.
9. ScreenChange. Содержит в себе функции перехода со сцены на сцену.

Также папка Resources (ресурсы) содержит в себе 13 спрайтов для заднего фона, один шрифт .ttf, 5 спрайтов для кнопок, текстовый файл «Dates» с датами, текстовый файл «Events» с событиями, текстовый файл «EventsDates» с датами и событиями.

* + 1. **Производительность, системные требования**

Приложение работает на смартфонах с версией OS Android не ниже Android 6.0 Marshmallow, занимает 54 Мб ПЗУ.

Тестирование приложения производилось на устройстве Xiaomi Redmi Note 5, Android 9, процессор Qualcomm Snapdragon 636 14 нм. 1.8 ГГц, 4 ГБ ОЗУ, размер дисплея 5,99 дюймов, соотношение сторон 18:9, разрешение 2160\*1080.

Холодный старт приложения занимает около 3 секунд, все события внутри игры обрабатываются мгновенно, приложение потребляет 100 +-10 Мб оперативной памяти. Графических артефактов за время тестирования не выявлено.

Планируется оптимизация потребления ОЗУ.

* + 1. **Cценарий использования конечным пользователем**

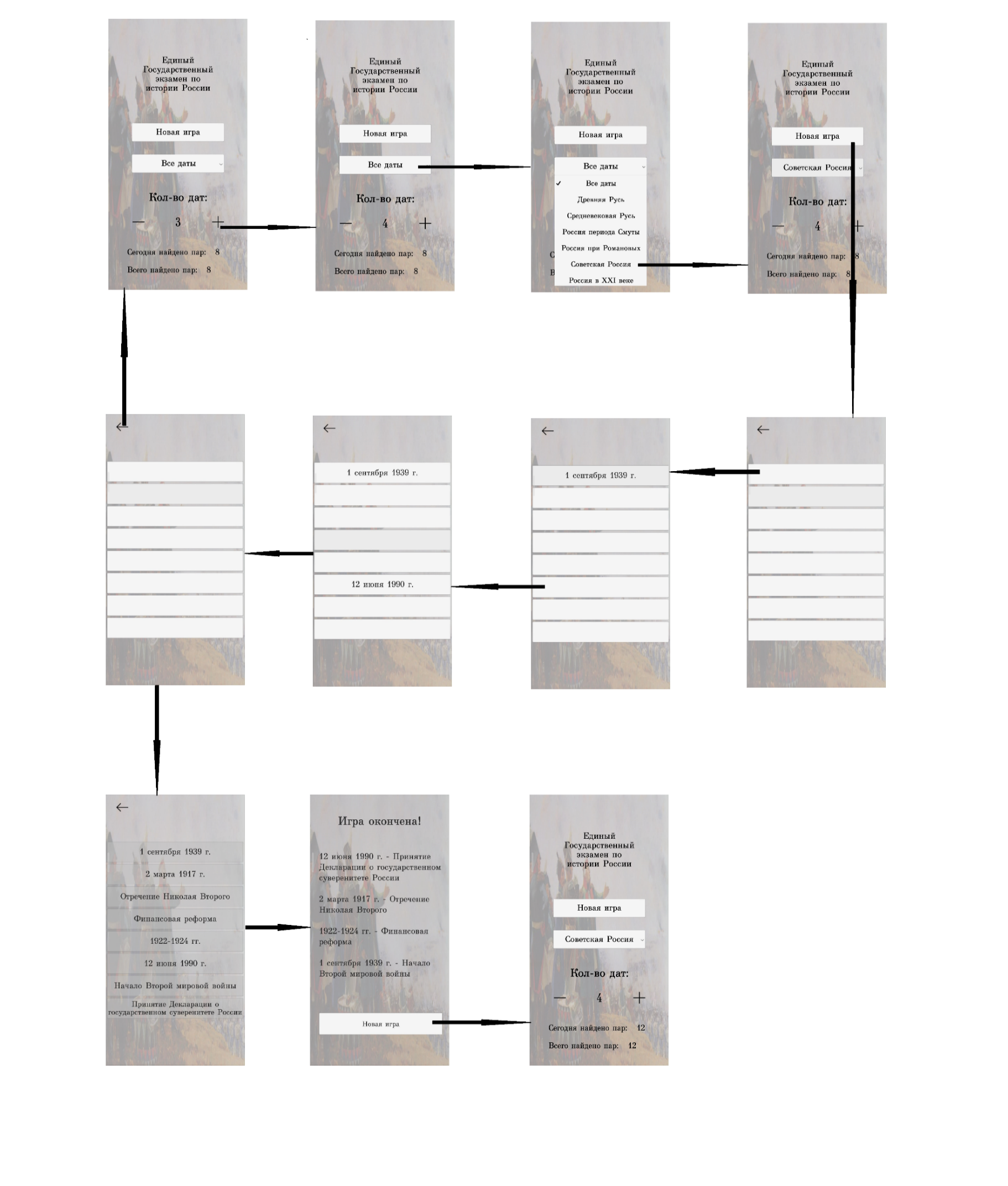
Сценарий использования приложения выглядит следующим образом (см. рис.1):

Рисунок 1

После запуска приложения пользователь попадает на начальный экран (сцена «StartScreen»). Здесь находится счетчик пар, собранных в этот день и за все время. Пользователю доступно 4 действия:

1. Увеличить количество дат (макс. 5)
2. Уменьшить количество дат (мин. 2)
3. Выбрать исторический период
4. Начать игру

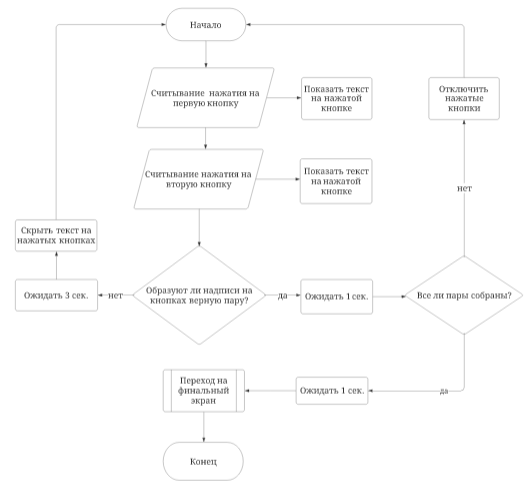
После нажатия на кнопку «Новая игра» пользователь попадает на игровой экран (сцена «Game»). На нем присутствуют пустые белые клавиши в количестве выбранного числа дат, умноженного на два. Игровая сцена работает следующему алгоритму (см. рис. 2):

Рисунок 2

Когда все пары собраны, пользователь попадает на конечный экран (сцена «FinishScreen»). На нем он имеет возможность прочесть пары дат и событий, которые были расположены на предыдущем экране, и нажать кнопку «Новая игра», которая возвращает его на начальный экран (сцена «StartScreen»)

**Глава 2. Программные коды скриптов**

* 1. **DateNumber**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

public class DateNumber : MonoBehaviour

{

public Text Num; // поле, отображающее кол-во дат

public Button Pl; // кнопка " + "

public Button Mi; // кнопка " - "

// выставляем ограничения при запуске скрипта (при каждом открытии начального экрана)

void Start()

{ if (Counter.counter == 5)

Pl.interactable = false; // отключаем кнопку " + ", если кол-во равно 5

if (Counter.counter == 2)

Mi.interactable = false; // отключаем кнопку " - ", если кол-во равно 2

Num.text = Counter.counter.ToString(); // прописываем кол-во из глобальной переменной

}

void Update()

{

if (Counter.counter <5 && Counter.counter > 2) // ограничитель для работы + и -

{

Pl.interactable = true;

Mi.interactable = true;

}

}

public void Minus () // работа нажатия на кнопку " - "

{

if (Counter.counter > 2)

Counter.counter--;

if (Counter.counter == 2)

Mi.interactable = false;

Num.text = Counter.counter.ToString(); // вывод значения в кол-во

}

public void Plus () // работа нажатия на кнопку " + "

{

if (Counter.counter < 5)

Counter.counter++;

if (Counter.counter == 5)

Pl.interactable = false;

Num.text = Counter.counter.ToString(); // вывод значения в кол-во

}

}

* 1. **Limits\_by\_Period**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

public class Limits\_by\_Period : MonoBehaviour

{

public static Limits\_by\_Period Instance { get; private set; } // объявление синглтона (разрешение на доступ к данным скрипта из любого другого)

public int low\_lim = 0; // нижняя граница номера события

public int high\_lim = 377; // верхняя граница номера события

public Dropdown period; // переменная типа DropDown (выпадающий список)

public void Awake() // функция, выполняющаяся при инициализации скрипта (при его первой работе)

{

Instance = this; // относится к объявлению синглтона, на что будет ссылаться синглтон (this - указатель на данный скрипт)

period.value = Counter.per\_change; // значение, принимаемое Dropdown по умолчанию

Period\_Changed(Counter.per\_change); // выставление границ по умолчанию

}

// Функция, отвечающая за смену границ, в зависимости от выбранного периода

public void Period\_Changed(int index) // в index передаётся значение DropDown, при его изменении

{

switch (index)

{ // случай по умолчанию (Все даты)

case 0:

low\_lim = 0;

high\_lim = 377;

Counter.per\_change = 0;

break;

// период "Древняя Русь"

case 1:

low\_lim = 5;

high\_lim = 26;

Counter.per\_change = 1;

break;

// период "Средневековая Русь"

case 2:

low\_lim = 27;

high\_lim = 66;

Counter.per\_change = 2;

break;

// период "Россия периода Смуты"

case 3:

low\_lim = 67;

high\_lim = 76;

Counter.per\_change = 3;

break;

// период "Россия при Романовых"

case 4:

low\_lim = 77;

high\_lim = 186;

Counter.per\_change = 4;

break;

// период "Советская Россия"

case 5:

low\_lim = 187;

high\_lim = 348;

Counter.per\_change = 5;

break;

// период "Россия в XXI веке"

case 6:

low\_lim = 349;

high\_lim = 377;

Counter.per\_change = 6;

break;

}

}

}

* 1. **Record\_printing**

using System.Collections;

using System;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

public class Record\_printitng : MonoBehaviour

{

public Text this\_day; // Text отвечающий за даты на сегодня

public Text all\_time; // Text отвечающий за всё время

string timequit; // время последнего выхода из игры

// Start is called before the first frame update

void Start()

{

this\_day.text = this\_day.text + " " + PlayerPrefs.GetInt("today"); // запись счёта в кол-во дат совпавших сегодня

all\_time.text = all\_time.text + " " + PlayerPrefs.GetInt("record"); // запись кол-во дат совпавших за всё время

timequit = PlayerPrefs.GetString("Time").Substring(0,2); // получение числа из даты, когда был совершён выход

// сравнение числа нынешнего дня с числом последнего выхода

// если числа разные, то есть начался другой день, мы обнуляем кол-во дат, отвеченных за этот день

if (DateTime.Now.ToShortDateString().Substring(0, 2) != timequit)

{

Counter.score = 0;

PlayerPrefs.SetInt("today", 0); // изменяем кол-во дат отвеченных за день на 0

}

}

private void OnApplicationQuit()

{

PlayerPrefs.SetString("Time", DateTime.Now.ToShortDateString()); // при выходе из игры сохраняем дату выхода

}

}

* 1. **AddButtons**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

using System.IO;

using System;

public class AddButtons : MonoBehaviour

{

[SerializeField]

public Transform puzzleField;

[SerializeField]

public GameObject btn;

public static int DateK; // кол-во кнопок, которые будут на экране игры

public int Changer = 0; //рандомно генерируемое число для обмена значений массива ButtonsContent

public string Buffer; //строка для обмена значений массива ButtonsContent

public static int[] RandomNums = new int[5]; //массив для рандомных значений для забивания строками массива BUttonContent

public static string[] DatesMain;

public static string[] EventsMain;

public Font krasivo;

void Start()

{

DateK = Counter.counter\*2;

System.Random TheBestRandomizer = new System.Random(); // объявляем переменную класса Random для случайных чисел

string[] ButtonsContent = new string[DateK];

TextAsset Dates = (TextAsset)Resources.Load("Dates", typeof(TextAsset)); //помещение файла в текст ассет, затем в строку, затем деление и помещение в массив строк

DatesMain = Dates.text.Split(new char[] { '\n' });

TextAsset Events = (TextAsset)Resources.Load("Events", typeof(TextAsset)); //то же самое

EventsMain = Events.text.Split(new char[] { '\n' });

for (int o = 0; o < 377; o++) //убираем символы перевода каретки

{

DatesMain[o]=DatesMain[o].Replace("\r", "");

EventsMain[o]=EventsMain[o].Replace("\r", "");

}

for (int x=0; x<Counter.counter; x++) // Заполняем RandomNums, исключая совпадения

{ //выбираем случайное число в диапазоне двух значений, в зависимости от периода

RandomNums[x] = TheBestRandomizer.Next(Limits\_by\_Period.Instance.low\_lim, Limits\_by\_Period.Instance.high\_lim);

//исключаем совпадения при работе random

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

{

if (RandomNums[x] == RandomNums[i])

{

while (RandomNums[x] == RandomNums[i])

RandomNums[x] = TheBestRandomizer.Next(Limits\_by\_Period.Instance.low\_lim, Limits\_by\_Period.Instance.high\_lim);

}

}

}

for (int NumbersEvents = 0; NumbersEvents < Counter.counter; NumbersEvents++) //этот цикл забивает датами и событиями массив зачений кнопок

{

ButtonsContent[NumbersEvents] = EventsMain[RandomNums[NumbersEvents]];

ButtonsContent[NumbersEvents + Counter.counter] = DatesMain[RandomNums[NumbersEvents]];

}

for (int i = DateK - 1; i > 1; i--) //этот цикл случайным образом перемешивает значения кнопок

{

Changer = TheBestRandomizer.Next(i); //генерируем случайное число больше или равно нуля меньше или равно i и назовем его Changer

Buffer = ButtonsContent[i];

ButtonsContent[i] = ButtonsContent[Changer]; //меняем местами ButtonsContens[i] и ButtonsContens[Changer];

ButtonsContent[Changer] = Buffer;

}

for (int x = 0; x < DateK; x++)

{

GameObject button = Instantiate(btn);

button.name = Convert.ToString(x);

button.transform.SetParent(puzzleField, false);

button.GetComponentInChildren<Text>().text = ButtonsContent[x]; // заполнение тектом кнопок

button.GetComponentInChildren<Text>().resizeTextForBestFit = true;

button.GetComponentInChildren<Text>().resizeTextMaxSize = 48;

button.GetComponentInChildren<Text>().resizeTextMinSize = 32; // размер шрифта на экране игры

button.GetComponentInChildren<Text>().fontStyle = FontStyle.Bold; // стиль текста жирный

button.GetComponentInChildren<Text>().font = krasivo; // меняем шрифт

button.transform.Find("Text").gameObject.SetActive(false);

}

}

}

* 1. **GameController**

using System.Collections;

using System.Collections.Generic;

using UnityEngine.UI;

using UnityEngine;

using System.IO;

using System;

public class GameController : MonoBehaviour

{

public List<Button> btns = new List<Button>();

private bool firstGuess, secondGuess; //логические переменные для отслеживания нажатий

public int CountGuesses, CountCorrectGuesses, gameGuesses; //различные счетчики

private int firstGuessIndex, secondGuessIndex; //храним номера нажатых кнопок для дальнейшего доступа

public string firstGuessText; //текст первой кнопки

public string secondGuessText; //текст второй кнопки

public string[] EventsDatesMain; //массив строк дата-событие

public TextAsset EventsDates;

public string sum, sumReverse; //сумма текстов кнопок с перестановкой слагаемых для дайльнейшего сравнения

public int Equality = 0; //ключ для выполнения корутины

void Awake ()

{

TextAsset EventsDates = (TextAsset)Resources.Load("EventsDates", typeof(TextAsset)); //помещаем текст из ресурсов в текстассет

EventsDatesMain = EventsDates.text.Split(new char[] { '\n' }); //делим строку по переносам строки

}

void Start()

{

for (int o = 0; o < 377 ; o++) //убираем переводы каретки из строк

{

EventsDatesMain[o]=EventsDatesMain[o].Replace("\r", "");

}

GetButtons();

AddListeners();

gameGuesses = Counter.counter;

}

void GetButtons()

{

GameObject[] objects = GameObject.FindGameObjectsWithTag("PuzzleButton"); //размещаем кнопки на экране

for(int i=0; i<objects.Length;i++)

{

btns.Add(objects[i].GetComponent<Button>());

}

}

void AddListeners()

{

foreach (Button btn in btns)

{ btn.onClick.AddListener(()=>PickAPuzzle()); } //добавляем лисенеры к кнопкам

}

public void PickAPuzzle()

{

if (!firstGuess) //при нажатии на первую кнопку получаем ее текст, номер, показываем текст

{

firstGuess = true;

firstGuessIndex = int.Parse(UnityEngine.EventSystems.EventSystem.current.currentSelectedGameObject.name);

firstGuessText = btns[firstGuessIndex].transform.Find("Text").gameObject.GetComponent<Text>().text;

btns[firstGuessIndex].transform.Find("Text").gameObject.SetActive(true);

}

else if(!secondGuess) //то же самое для второй кнопки, но происходит еще и сравнение суммы строк со строками дата-событие, то есть проверяется, верна ли пара

{

secondGuess = true;

secondGuessIndex = int.Parse(UnityEngine.EventSystems.EventSystem.current.currentSelectedGameObject.name);

secondGuessText = btns[secondGuessIndex].transform.Find("Text").gameObject.GetComponent<Text>().text;

btns[secondGuessIndex].transform.Find("Text").gameObject.SetActive(true);

sum = firstGuessText + secondGuessText;

sumReverse = secondGuessText + firstGuessText;

CountGuesses++;

for (int i = 0; i < 377; i++)

{

if (sumReverse.Equals(EventsDatesMain[i]) | sum.Equals(EventsDatesMain[i])) //если пара верная, то прибавляем к счетчикам по единице

{

Equality = 1;

CountCorrectGuesses++;

Counter.score++;

Counter.record++;

}

}

StartCoroutine(CheckIfThePuzzleMatch()); //вызываем корутину

}

}

IEnumerator CheckIfThePuzzleMatch()

{

if (Equality == 1) //если пара верна, то ждем секунду и отключаем кнопки

{

yield return new WaitForSeconds(1.0f);

btns[firstGuessIndex].interactable = false;

btns[secondGuessIndex].interactable = false;

Equality = 0;

}

else //если пара неверная, то ждем три секунды и скрываем текст

{

yield return new WaitForSeconds(3.0f);

btns[firstGuessIndex].transform.Find("Text").gameObject.SetActive(false);

btns[secondGuessIndex].transform.Find("Text").gameObject.SetActive(false);

}

firstGuess = false;

secondGuess = false;

if (CountCorrectGuesses == gameGuesses) //каждый раз проверяем, все ли пары собраны.Если да, то сохраняем счетчики очков и переходим на конечный экран

{

yield return new WaitForSeconds(1.0f);

UnityEngine.SceneManagement.SceneManager.LoadScene("FinishScreen");

PlayerPrefs.SetInt("record", Counter.record);

PlayerPrefs.SetInt("today", Counter.score);

}

}

}

* 1. **BackGround**

using System.Collections;

using System;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

public class BackGround : MonoBehaviour

{

public Sprite[] IMG\_list; // это массив изображений

public Image backGround; // это панель, которая будет использоваться в качестве BCGD

void Start()

{

backGround.sprite = IMG\_list[Counter.num\_img]; // передаём глобальную переменную, которой присвоили рандомное значение в диапазоне кол-ва картинок

}

}

* 1. **Counter**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public static class Counter // класс для хранения глобальных переменных, которые видны всем скриптам в любой сцене игры

{

public static int counter = 3; // глобальная переменная для кол-ва дат

public static int num\_img = Random.Range(0, 13); // глобальная переменная для номера картинки фона

public static int score = PlayerPrefs.GetInt("today"); // счёт текущего дня

public static int record = PlayerPrefs.GetInt("record"); // рекорд за всё время

public static int per\_change = 0; // переменная для сохранения значения DropDown(выбор периода)

}

* 1. **Writer**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

public class Writer : MonoBehaviour

{

public Text Teext; // объявляем переменную типа Text, куда выведем итоги игры

void Start()

{

string Events; // строка, которая будет содержать в себе событие с датой

for (int i = 0; i<Counter.counter; i++)

{ // по очереди передаём строке значения даты и события

Events = AddButtons.DatesMain[AddButtons.RandomNums[i]] + " - " + AddButtons.EventsMain[AddButtons.RandomNums[i]] + '\n' + '\n';

//выводим эту строку на экран

Teext.text+= Events;

}

}

}

* 1. **ScreenChange**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class ScreenChange : MonoBehaviour

{

public void ScreenChanger()

{

UnityEngine.SceneManagement.SceneManager.LoadScene("Game"); // загружаем основную сцену игры

}

public void StartScreen()

{

UnityEngine.SceneManagement.SceneManager.LoadScene("StartScreen"); // загружаем стартовый экран

PlayerPrefs.SetInt("record", Counter.record); // сохраняем в файлах игры кол-во дат за всё время

PlayerPrefs.SetInt("today", Counter.score); // сохраняем в файлах игры кол-во дат за сегодня

System.GC.Collect(); // принудительный запуск сбора накопившегося мусора для его очистки

}

# Заключение

В ходе работы над проектом была изучена история Пельманизма и «Memory Game», общая информация о четырех популярных игровых движках, базовый синтаксис языка С#, устройство редактора Unity, некоторые функции библиотек Unity.UI и UnityEngine.

Было разработано и протестировано мобильное приложение под ОС Android, отвечающее почти всем изначальным требованиям. Некоторые задачи не были выполнены: дизайн приложения уже отличается от тестового варианта, но не является финальным. Также приложение не было опубликовано в магазине Google Play, так как сложность разработки изначально была оценена неверно, соответственно, неверно были определены и сроки работы. Публикация в онлайн-магазине планируется после некоторых доработок.

Таким образом, цель достигнута, выполнено 5/7 задач.

**Список литературы**

Cocos2d. URL: https://ru.wikipedia.org/wiki/Cocos2d (дата обращения - 18.04.2020)

Cocos2d-x – Основные понятия. URL: https://habr.com/ru/post/339564/ (дата обращения - 18.04.2020)

How Android paved the way for the smartphone revolution. URL: https://www.bloomberg.com/graphics/2019-android-global-smartphone-growth/ (дата обращения – 10.04.20)

LibGDX. URL: https://ru.wikipedia.org/wiki/LibGDX (дата обращения - 18.04.2020)

Online Diagram Software & Visual Solution. URL: https://www.lucidchart.com/pages/ (дата обращения - 18.04.2020)

The Pelman school of memory. The Pelman Institute and Pelmanism. URL: http://www.ennever.com/histories/history386p.php/ (дата обращения - 17.04.20)

Unreal Engine. URL: <https://ru.wikipedia.org/wiki/Unreal_Engine/> (дата обращения - 18.04.20)

Обзор игрового движка Unity3d. URL: https://gamedevmania.ru/engines/unity3d/ (дата обращения - 18.04.20)

Обзор и анализ возможностей современного игрового движка Unreal Engine / Крупин А С., Жилин И. А. // «Экономика и социум» №12(31) 2016. С. 1951-1954.