ГЛАВНОЕ УПРАВЛЕНИЕ ОБРОЗОВАНИЯ ГОМЕЛЬСКОГО ОБЛИСПОЛКОМА

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

«ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ МАШИНОСТРОИТЕЛЬНЫЙ КОЛЛЕДЖ»

**ПОЯНИТЕЛЬНАЯ ЗАПИСКА**

**к курсовому проекту**

КП.КПяП.0429961ПЗ

Разработал Лосев А.С.

Руководитель

проекта В.М. Гаврилова

Гомель 2020

Оглавление

[**ВВЕДЕНИЕ** 3](#_Toc42549754)

[**1 ПОСТАНОВКА ЗАДАЧИ** 4](#_Toc42549755)

[**1.1 Описание задачи** 4](#_Toc42549756)

[**1.2 Требования к программе** 4](#_Toc42549757)

[**1.3 Входные и выходные данные** 5](#_Toc42549758)

[**1.4 Описание состава технических и программных средств** 5](#_Toc42549759)

[**2 ОПИСАНИЕ ПРОГРАММЫ** 6](#_Toc42549760)

[**2.1 Общие сведения** 6](#_Toc42549761)

[**2.2 Структура программы** 6](#_Toc42549762)

[**2.3 Входные и выходные данные** 7](#_Toc42549763)

[**2.4 Вызов и загрузка** 8](#_Toc42549764)

[**3 ОПИСАНИЕ ПРИМЕНЕНИЯ** 9](#_Toc42549765)

[**3.1 Требования к аппаратным и программным средствам** 9](#_Toc42549766)

[**3.2 Требования техники безопасности и охраны труда при эксплуатации программы** 9](#_Toc42549767)

[**3.3 Энерго и ресурсосбережение** 12](#_Toc42549768)

[**4 РУКОВОДСТВО ПРОГРАММИСТА** 13](#_Toc42549769)

[**4.1 Назначение программного обеспечения** 13](#_Toc42549770)

[**4.2 Характеристики программы** 13](#_Toc42549771)

[**4.3 Установка программы** 13](#_Toc42549772)

[**4.4 Сообщения программисту** 15](#_Toc42549773)

[**5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ** 16](#_Toc42549774)

[**5.1 Наименование программы** 16](#_Toc42549775)

[**5.2 Условия выполнения программы** 16](#_Toc42549776)

[**5.3 Выполнение программы** 16](#_Toc42549777)

[**5.4 Сообщение оператору** 17](#_Toc42549778)

[**6 ПРОГРАММА И ДЕТОДИКА ИСПЫТАНИЙ** 18](#_Toc42549779)

[**6.1 Объект и цель испытаний** 18](#_Toc42549780)

[**6.2 Порядок испытаний** 18](#_Toc42549781)

[**6.3 Методы испытаний** 19](#_Toc42549782)

[**ЗАКЛЮЧЕНИЕ** 20](#_Toc42549783)

[**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ** 21](#_Toc42549784)

[**ПРИЛОЖЕНИЕ А** 22](#_Toc42549785)

[**ПРИЛОЖЕНИЕ Б** 41](#_Toc42549786)

[**ПРИЛОЖЕНИЕ B** 51](#_Toc42549787)

**ВВЕДЕНИЕ**

Игра - тип осмысленной непродуктивной деятельности, где мотив лежит не в её результате, а в самом процессе. Также термин «игра» используют для обозначения набора предметов или программ, предназначенных для подобной деятельности.

Современное общество основано на коммуникации. От межличностных связей зависит карьера, досуг, семья – все самые важные сферы человека. Игры в интерактивном формате – это деятельность которая направлена на познание окружающего мира и решение поставленных задач с помощью коллективной работы.

Решение этой задачи в значительной степени зависит от внедрения компьютерных технологий в учебный процесс. Современные технические и программные средства, сетевые и мультимедийные технологии уже позволяют создавать электронные учебники, электронные лекционные курсы и проводить дистанционное обучение и развитие навыков в режиме реального времени.

Необычайно эффективным и действенным методом обучения является интерактивная игра. Она позволяет в легкой форме рассказать про окружающий их мир, познакомить с механиками игры и при этом учит их взаимодействовать друг с другом для достижения конечной цели.

В интерактивной игре происходит взаимодействие между игроками и компьютером или реальным игроком, игроки учатся не только новому, но и общению, пониманию друг к другу, они во время игры приобретают собственный опыт, который будут применять в дальнейшем.

Основы алгоритмизации и программирование являются фундаментальными основами теоретической информатики, дисциплина имеет весьма ограниченное время для изучения. Поэтому некоторые важные вопросы не рассматриваются на лекциях. Учебным средством, предусматривают ориентацию на самостоятельную работу и самообразование.

# **1 ПОСТАНОВКА ЗАДАЧИ**

## **1.1 Описание задачи**

## 

В качестве задания на курсовой проект по дисциплине «Конструирование программ и языки программирования» требуется разработать электронную интерактивную игру «Billionaire». Программа разрабатывается для учреждения образования «Гомельский государственный машиностроительный колледж».

Постановщикам задачи является преподаватель колледжа. Пользователями данной программы являются любые заинтересованные люди.

Игра – тип осмысленной и структурированной непродуктивной деятельности, где мотив лежит не в ее результате, а в самом процессе. Настольная игра, основанная на манипуляции относительно небольшим набором предметов, которые могут целиком разместиться на столе или в руках играющих.

Главной целью реализации является развитие у игрока экономических навыков. Это достигается за счет правил игры. Каждый ход игроку предоставляется возможность походить на определенное количество клеток, число которых равно числу, выпавшему на кубике. Игрок выбирает купить предприятие или нет, если сделка не состоялась, то игрок может идти дальше, или попытаться еще раз совершить сделку. Но в такие случаи он рискует отстать от других игроков и в будущем остаться в долгах. Так же продажа или взятие кредита в банке может как помочь поднятию репутации, а значит и доходу. Так и к банкротству.

В зависимости от предпочтений игрока в стратегии игры, а также за счет налаживания логических и экономических навыков, и будет зависеть продвижения игрока. В игре может быть, как прогресс, так и ступор вплоть до завершения игры.

**1.2 Требования к программе**

К разрабатываемой программе предъявляются следующие требования:

* ввод данных;
* выбор вариантов действий;
* вывод данных на экран;
* наличие справочной системы;
* сохранение данных;
* предусмотреть возможность одновременной игры от одного до четырех игроков;
* генерация реакции программы случайным образом.

К интерфейсу программы предъявляются следующие требования:

* кнопки для работы с меню игры;
* кнопки для выбора или подтверждения действия;
* наличие основного, и вспомогательных форм;
* наличие полей для ввода и вывода данных.

**1.3 Входные и выходные данные**

Входными данными в программе являются:

* имя игрока;
* выбранное пользователем действие.

Выходными данными в программе являются:

* реакции программы на действия пользователя;
* вывод статистики;

**1.4 Описание состава технических и программных средств**

Состав и параметры технических средств:

* AMD Sempron (tm) Processor 2800+ 1.60 Ггц.;
* 1024МБ ОЗУ;
* GeForse 7600, 1024mb.

Требования к информационной и программной совместимости:

* ОС Windows не ниже версии 7;

Для реализации проекта, полученного как задание на практику, была выбран игровой движок «Unity5» так как он представляет собой бесплатный мощный игровой движок для разработки мульти платформенных игр с поддержкой всего необходимого функционала. Совместно со средой разработки «Microsoft Visual Studio 2019». При решении поставленной задачи оптимально использовать для представления информационных материалов язык C#, который является языком высокого уровня и позволяет быстро и эффективно создавать приложения.

# **2 ОПИСАНИЕ ПРОГРАММЫ**

## **2.1 Общие сведения**

Наименование программы: электрона интерактивная игра «Billionaire».

Обозначение программы: «Setup.exe»;

Автор: Учащийся УО ГГМК, группы ПК-32, Лосев Алесей Сергеевич.

Дата создания: 23.06.2020 г.

Версия программы: 0.2.1.

Программная среда и программное обеспечение, необходимое для функционирования программы:

* среда исполнения: операционная система от Windows 7 и выше (32-bit / 64-bit);
* система управления базами данных: любой программный редактор текста, поддерживающий открытие файлов формата «.txt» (необходим только для редактирования текстового материала).

Рекомендованная версия программы: 0.2.1 без модификаций.

Программа разработана на языке программирования C# в среде Microsoft Visual Studio 2019.

## **2.2 Структура программы**

Логическая структура взаимодействия клиентского приложения с базой данных представлена на рисунке 2.1.

Клиентское приложение

Обращение к специальной процедуре, для чтения файла

База данных

Рисунок 2.1 - Логическая структура программы

Схема работы выбора действия представлена на рисунке 2.2.

Предоставление выбора действия пользователю

Вариант 1

Вариант 2

Вариант 3

Запрос к процедуре

Запрос к процедуре

Запрос к процедуре

Рисунок 2.2 – Схема работы выбора действия

Структура таблицы базы данных представлена в таблице 2.1.

Таблица 2.1 – Структура таблиц

|  |  |  |  |
| --- | --- | --- | --- |
| **Имя файла** | **Тип дынных** | **Размер** | **Описание** |
| RecordFile | Числовой | Целое | Хранит количество очков каждого игрока за лучшую игру, а также кол-во очков за всю игру. |
| GameRulesRu | Текстовый | 50 | Содержит текстовое описание правил игры на русском языке. |
| GameRulesEn | Текстовый | 50 | Содержит текстовое описание правил игры на английский языке. |

Содержимое каждого файла будет содержаться в приложении А таблица 1.

Используемый набор протоколов данных для обмена данными логического типа, между участками, классами и программами. Это задает единообразный способ передачи данных или сообщений. Классы, процедуры и функции будут содержатся в приложении А таблица 2.

## **2.3 Входные и выходные данные**

Организация входных данных представлена в рисунке 2.3.

Запрос на ввод данных

Обработка входных данных, присваивание необходимых параметров, продолжение работы

Рисунок 2.3 – Схема входных данных

Организация выходных данных представлена в рисунке 2.4.

Запрос на вывод данных

Обработка запроса, считывание необходимых параметров, вывод данных, продолжение работы

Рисунок 2.4 – Схема выходных данных

Предоставляется возможность ввода данных, с передачей этих данных в заранее созданной переменной. Форматами входных и выходных данных являются: числовые, символьные, текстовые, логические и вещественные.

## **2.4 Вызов и загрузка**

Вызов программы осуществляется с помощью исполняемого файла «Setup.exe», находящегося на жестком диске, или на переносном носителе. Вход в программу осуществляется только с исполняемого файла «Setup.exe».

# **3 ОПИСАНИЕ ПРИМЕНЕНИЯ**

Наименование программы: электронная интерактивная игра «Billionaire».

Назначение программы: развитие экономических навыков у пользователя.

Входные данные: имя игрока, выбранное пользователем действие.

Выходные данные: реакция программы на действие пользователя, вывод статистики.

## **3.1 Требования к аппаратным и программным средствам**

Для корректной работы программы и выполнения всех ее функций необходима следующая конфигурация аппаратного обеспечения:

* AMD Sempron (tm) Processor 2800+ 1.60 Ггц.;
* 1024МБ ОЗУ;
* GeForse 7600, 1024mb.
* дисковый накопитель не менее 100 МБ свободного места;
* разрешение экрана должно быть не менее 1280х720.

## **3.2 Требования техники безопасности и охраны труда при эксплуатации программы**

При эксплуатации программы необходимо соблюдать правила техники безопасности и охраны труда при работе на ПК:

- продолжительность непрерывной работы на ПК не должна превышать 40 минут с 10 – перерывом.

- общая продолжительность работы за компьютером не должна превышать в день более 4-х часов;

- не допускать попадания внутрь компьютера металлических предметов, воды;

- не допускать чрезмерное накопление пыли внутри компьютера, своевременно проводя работы по техническому обслуживанию;

- не подключать и не отключать интерфейсные кабели при включённом электропитании.

Основной опасностью при работе на ПК является поражение электрическим током, которое может произойти при неисправности ПК или не соблюдении требований техники безопасности. Во избежание опасности поражения электрическим током, обязательными для выполнения являются следующие правила техники безопасности:

- компьютер можно подключить только к электрической розетке;

- не снимать крышку корпуса компьютера при включенном электропитании;

- не работать с компьютером, если повреждён сетевой кабель;

- не касаться частями тела экрана или тыльной стороны монитора стороны монитора, а также питающих проводов;

- не работать на ПК мокрыми руками, в сырой одежде и обуви;

- не работать на ПК во время грозы;

- не пользоваться открытым огнём при работе на ПК;

- вынимать вилку, держаться за вилку, а не за кабель. Не касаться вилки влажными руками;

- полное обесточивание компьютера возможно только при отключении шнура питания от электрической розетки. Поэтому доступ к ней должен быть всегда открыт.

Немедленно отключите компьютер от электрической сети, если:

- появится огонь, дым, необычный звук или запах;

- повреждён сетевой кабель, разъём или розетка;

- внутрь компьютера попала вода или металлический предмет.

Жизненно важным для повышения работоспособности пользователя и сохранения его здоровья, является правильная организация рабочего места пользователя. Кроме того, выбор места важен для надёжной работы компьютера и сопряжённого с ним оборудования. Требования, предъявляемые к организации и оборудованию автоматизированных рабочих мест:

- компьютер предназначен для эксплуатации в закрытых отапливаемых помещений с низкой запыленностью, в котором не должно возникнуть условий для конденсации влаги:

- размеры (площадь, объём) помещения должны соответствовать количеству работающих и размещаемому в них комплексу технических средств;

- планировка рабочего места должна удовлетворять требованиям удобства выполнения работ, экономии энергии и времени, рационального использования производительных площадей и удобства обслуживания устройств ПК, соблюдения охраны и труда;

- при небольшом количестве рабочих предпочтений следует отдавать расстановке ПК у стены, противоположной оконным проёмам;

- конструкция рабочей мебели (столы, кресла и стулья) должна обеспечивать возможность индивидуальной регулировки соответственно росту работающего и создать условия для удобства при работе. Рабочий стол должен регулироваться по высоте в пределах 0.68-0.76 м. Наиболее оптимальным размером рабочей поверхности стола является -1.6×0.9 м. Под ней должно быть свободное пространство для ног с размерами по высоте – не менее 0.6 м от пола, по ширине 0.5 м, по глубине -0.65м;

- в помещении температура воздуха должна содержать 19-21 градусов Цельсия, относительная влажность воздуха 55-56%, скорость движения воздуха не более 0.2м/с;

- уровни звука не должны превышать 50 ДБ. Учитывая, что основным источником шума является принтер и принимая во внимание повсеместное использование практических бесшумных струйных принтеров, этого требования достичь не трудно.

Требования безопасности в аварийных ситуациях.

При перерывах и других нарушениях в подаче электроэнергии компьютер следует выключать. Выключение компьютера обязательно даже в случае отсутствия напряжения в сети питания.

Признаками отключения от нормальной работы ПК являются:

- возникновения необычного шума системного блока или монитора;

- появление запаха гари или озона, возникновения дыма или пламени;

- искажение или исчезновение изображения на мониторе;

- непонятное или исчезновения изображения на мониторе;

- непонятное пользователю неожиданное изменение в работе ПК.

Если при выполнении работы пользователь заменил неисправность, он обязан прекратить работу, сообщить об этом лицу, ответственному за безопасность выполнение работы.

При обнаружении повреждений проводов питания, неисправности заземления и других повреждений электрооборудования, необходимо прекратить работу и сообщить об аварийной ситуации руководителю. Обо всех сбоях в работе оборудования или программного обеспечения необходимо сообщать руководителю.

При появлении рези в глазах, ухудшению видимости, появлении боли в пальцах и кистях рук, усилении сердцебиения необходимо покинуть рабочее место и сообщить о происшедшем руководителю работ. В случае поражения электрическим током работающего рядом необходимо немедленно освободить пострадавшего от действия тока и начать оказание первой помощи, используя соответствующие его состоянию способы: при нарушении дыхания – искусственное дыхание, при нарушении сердечной деятельности – непрямой массаж сердца. Для обесточивания оборудования – выключить автомат питания в помещении. Доложить о несчастном случае руководителю. При необходимости – воспользоваться аптечкой, и вызвать скорую помощь по телефону 103.

Пользователь ПК должен быть обучен правилам тушения пожара, информирован о плане эвакуации людей и имущества из помещения.

В случае возникновения пожара, немедленно вызвать пожарную команду, сообщить о пожаре руководителю и приступить к ликвидации очагов возгорания имеющимися средствами, предварительно обесточив помещение выключением автомата питания или рубильника в коридоре. Лицам, не участвующим в тушении пожара, следует покинуть помещение. При сильном задымлении помещения необходимо закрыть дыхательные пути влажной тканью и выйти из помещения по эвакуационным путям.

## **3.3 Энерго и ресурсосбережение**

Отдельное внимание необходимо уделить энерго и ресурсосбережению. Снизить уровень энерго и ресурсопотребления возможно за счёт наукоёмких энерго- и ресурсосберегающих технологий.

Одним из направлений снижения энергопотребления является использование жидкокристаллических мониторов.

Также по окончанию работы на ПК необходимо полностью выключать и отключать от сети компьютер, монитор, оргтехнику, такую как принтеры, сканеры, ксероксы. В режиме ожидания, по мнению специалистов, компьютер оставлять не следует, так как он потребляет электроэнергию.

Есть ещё много путей, с помощью которых можно снизить энергопотребление. К ним можно отнести такие пути, как использование последних моделей процессоров, чипов и т.д.

# **4 РУКОВОДСТВО ПРОГРАММИСТА**

## **4.1 Назначение программного обеспечения**

Наименование программы: электронная интерактивная игра «Billionaire».

Назначение программы: развитие экономических навыков у пользователя.

## **4.2 Характеристики программы**

Минимальная конфигурация технического оборудования:

* AMD Sempron (tm) Processor 2800+ 1.60 Ггц.;
* 1024МБ ОЗУ;
* GeForse 7600, 1024mb.
* дисковый накопитель не менее 100 МБ свободного места;
* разрешение экрана должно быть не менее 1280х720.

Необходимое программное обеспечение:

* среда исполнения: ОС от Microsoft Windows 7 или выше (32 – bit / 64 - bit);
* любой доступный базовый редактор текста, поддерживающий формат «.txt».

Контроль правильности выполнения программы осуществляется методом исключения. Программа предотвращает ввод данных неприемлемого формата, путем их отслеживания и исправления до вывода ошибок.

В случае сбоя управления или параметров данных, в программе предусмотрена возможность сохранения.

## **4.3 Установка программы**

За установку программы отвечает инсталляционный пакет «setup.exe», при загрузке которого сначала будет предложено выбрать язык мастера установки представлено на рисунке 4.1.

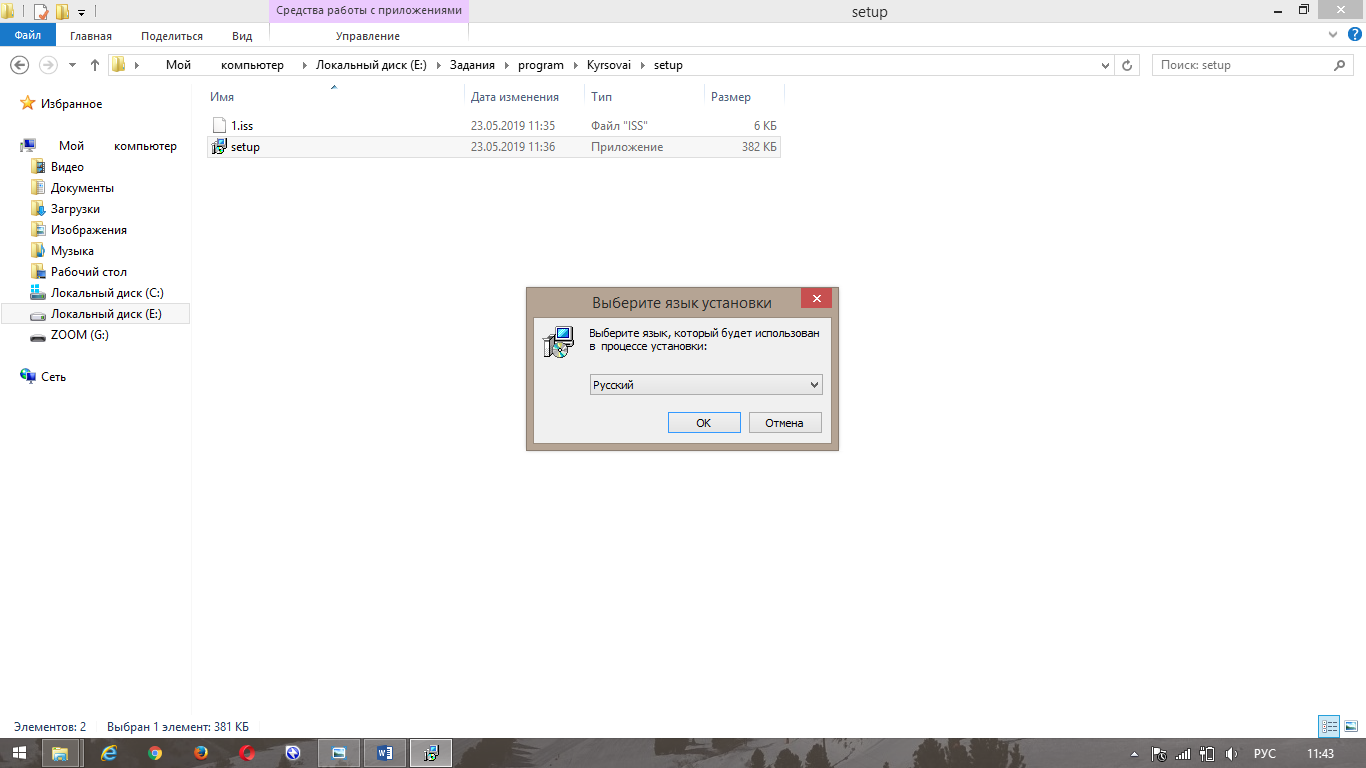


Рисунок 4.1 – Выбор языка мастера установки

После выбора языка откроется мастер, сообщающий о том, какая программа (название, версия) будет установлена на компьютер, как на рисунке 4.2 и рекомендация закрыть прочие приложения перед установкой.

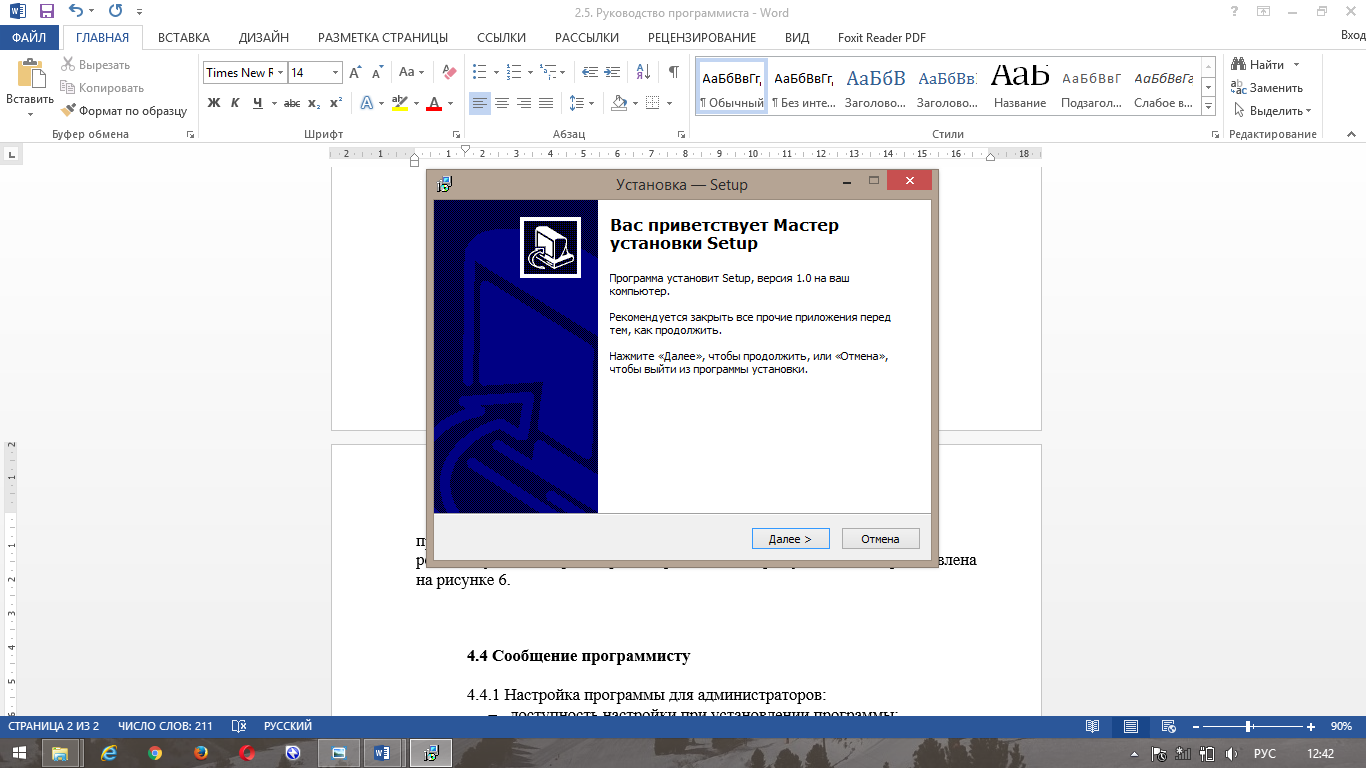


Рисунок 4.2 – Открытие мастера установки

После ознакомления названия и версии программы представляется выбор принятия лицензионного соглашения, как на рисунке 4.3.

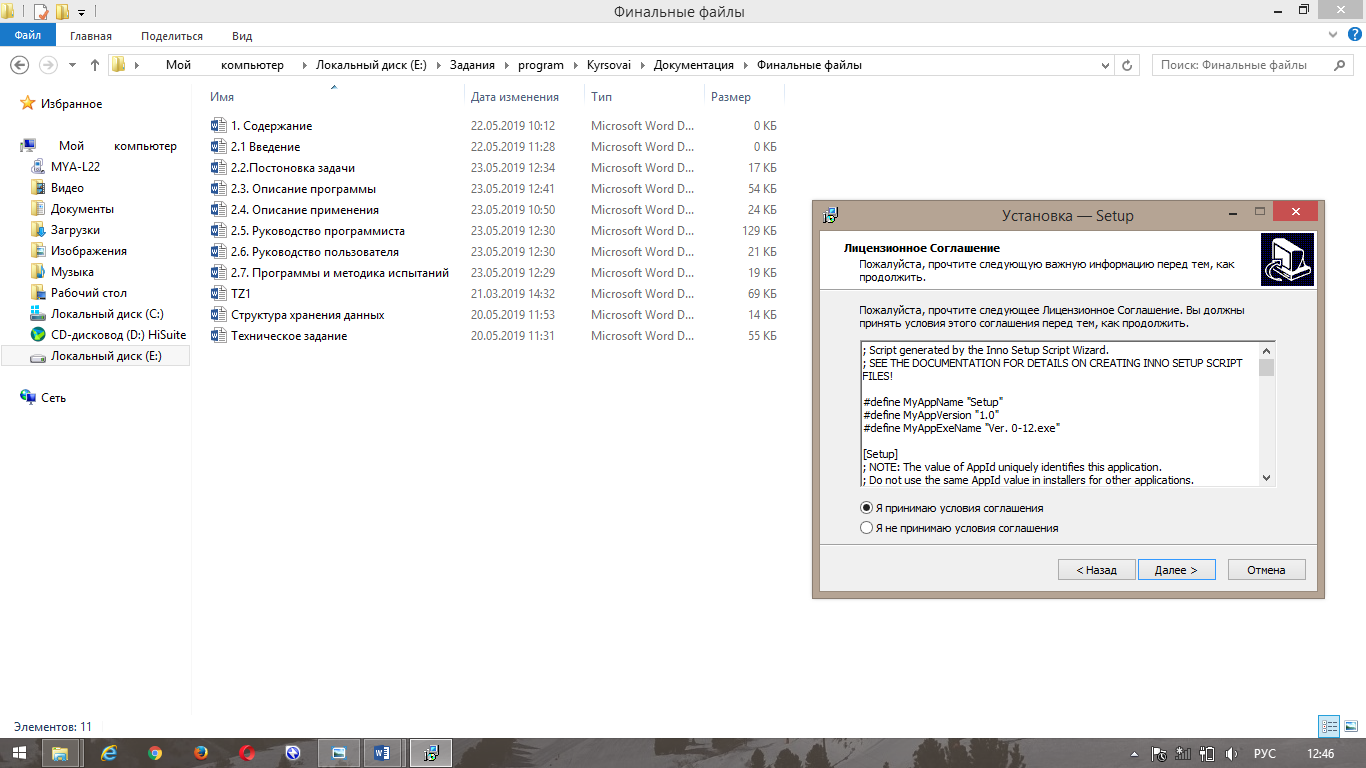


Рисунок 4.3 – Принятие лицензионного соглашения

После принятия лицензионного соглашения требуется указать путь, по которому мы будет устанавливать программу как на рисунке 4.4. По умолчанию будет предложен путь «C:\Program Files (x86)\Setup».

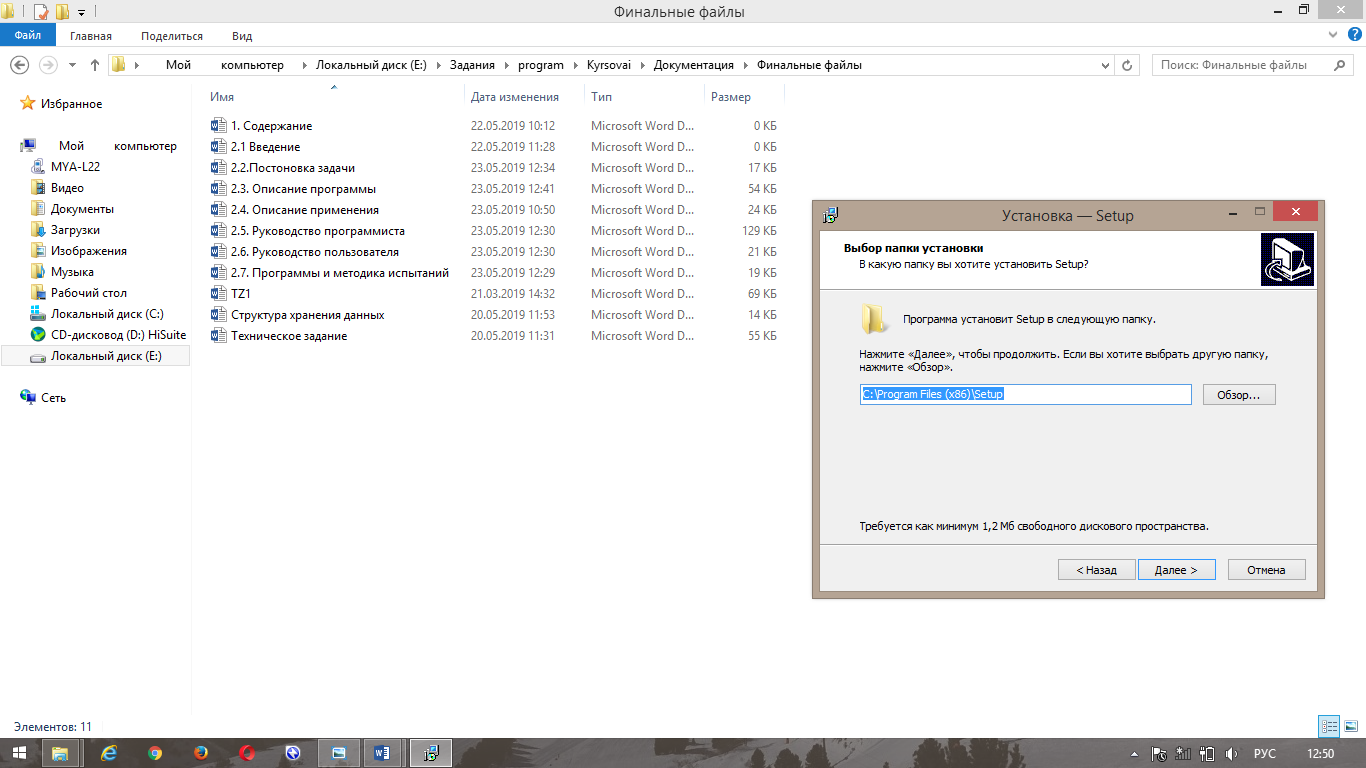


Рисунок 4.4 – Выбор пути установки программы

После демонстрируют принятые настройки. Дальше предлагается установить программу, или вернуться и изменить принятые настройки как на рисунке 4.5.

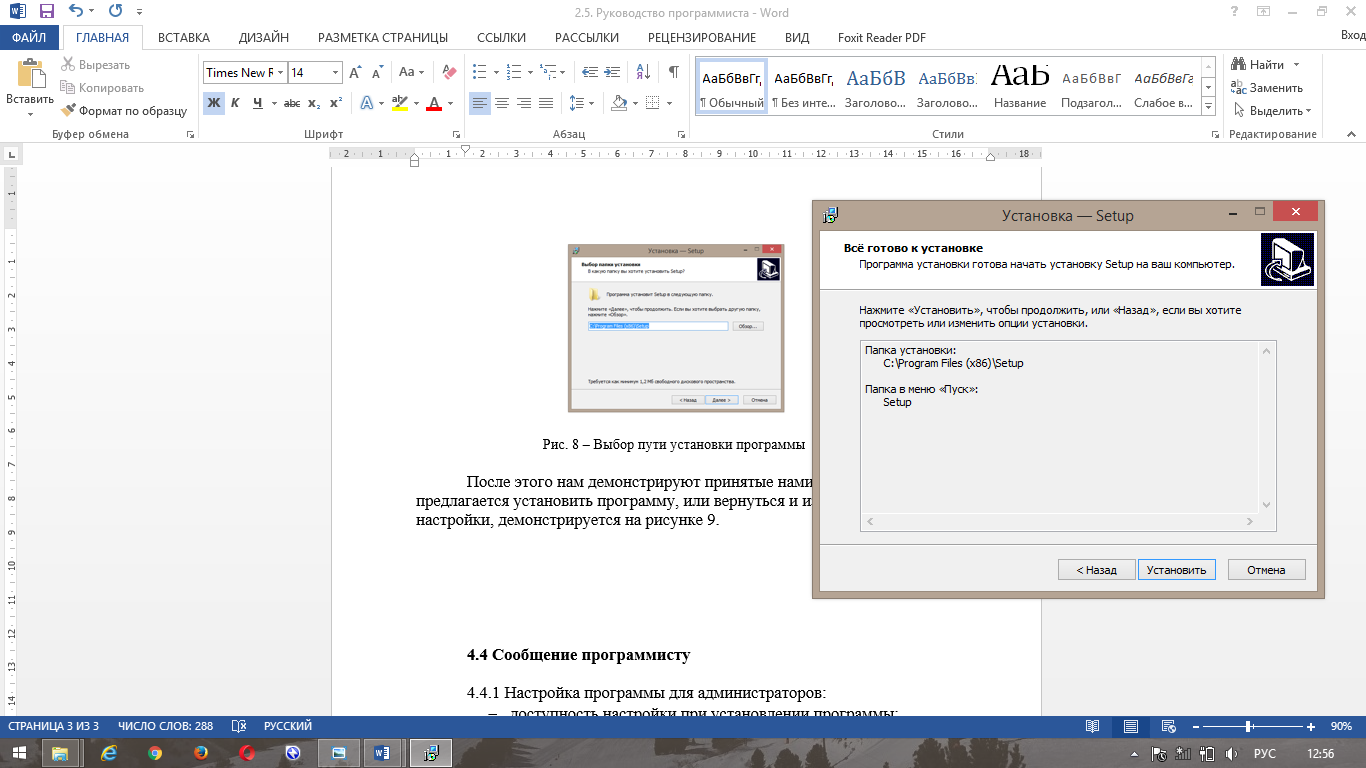


Рисунок 4.5 –Установки программы

После установки на ваш компьютер программа будет доступна для запуска, и дальнейшей работы с ней.

## **4.4 Сообщения программисту**

Настройка программы для администраторов:

* доступность настройки при установлении программы;

Средства самовосстановления программы не предусмотрены. При удалении одного из комплектующих необходимо переустановить программу.

# **5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ**

## **5.1 Наименование программы**

Наименование программы: электронная интерактивная игра «Billionaire».

Назначение программы: развитие экономических навыков у пользователя.

Входные данные: имя игрока, выбранное пользователем действие.

Выходные данные: реакция программы на действие пользователя, вывод статистики.

## **5.2 Условия выполнения программы**

Минимальная конфигурация технического обеспечения:

* AMD Sempron (tm) Processor 2800+ 1.60 Ггц.;
* 1024МБ ОЗУ;
* GeForse 7600, 1024mb.
* дисковый накопитель не менее 100 МБ свободного места;
* разрешение экрана должно быть не менее 1280х720.

Необходимое программное обеспечение:

* среда исполнения: ОС от Microsoft Windows 7 или выше (32 – bit / 64 - bit);
* любой доступный базовый редактор текста, поддерживающий формат «.txt».

## **5.3 Выполнение программы**

Загрузка программы производится по ярлыку на рабочем столе, или путем запуска файла «Setup.exe» в каталоге с установленной программой. Программа по умолчанию устанавливается по пути: «C:\ProgramFiles (x86)\Setup.exe». Для ознакомления с правилами игры, пользователю необходимо ознакомиться с правилами, которые расположены в файле «GameRulesRu». Стартовое меню программы представлено на рисунке 5.1.



Рисунок 5.1 – Стартовое меню программы

Управление программой осуществляется путем нажатия соответствующих кнопок в меню, отвечающей за определенное действие. Значение кнопок представлено на рисунке 5.2.



Рисунок 5.2 – Меню игры

## **5.4 Сообщение оператору**

В ходе выполнения работы, программа всегда сопровождает действия пользователя пояснительным текстом или сообщением. В случаи сбоя программы без возможности сохранения, пользователю рекомендуется выключить программу и перезапустить ее. Подробное описание механик и правил игры представлено в приложении А.

# **6 ПРОГРАММА И ДЕТОДИКА ИСПЫТАНИЙ**

## **6.1 Объект и цель испытаний**

Объектом испытаний является электронная интерактивная игра «Billionaire». Данная программа предназначена для развития у пользователя экономических навыков.

Целью испытаний является проверка полной работоспособности программы и соответствия её функций требованиям технического задания.

## **6.2 Порядок испытаний**

Проверка корректности запуска программы

В стандартном запуске программы необходимо подождать загрузки сцены меню, если все настройки были успешно пройдены то игрок видит сцену меню игры как на рисунке 6.1.



Рисунок 6.1 – Меню игры

Проверка соответствий программно-аппаратной среды, осуществляется за счет продолжения работы программы. За счет сгенерированных чисел, игрок может сгенерировать название страны и название города принадлежащего этой стране, название стран и городов представлены в приложении А, результат работы программы представлен на рисунке 6.2.



Рисунок 6.2 – Результат работы программы

## **6.3 Методы испытаний**

Один из наиболее часто используемых методов является тесты. Программа многократно была подвергнута прогонам (тестам), цель которых была: нахождение ошибок, неисправностей, нахождение критических ошибок и настройка сбалансированной игры.

# **ЗАКЛЮЧЕНИЕ**

В ходе выполнения курсового проекта была разработана электронная интерактивная игра «Billionaire».

Все функции программы реализованы в соответствие с требованиями технического задания и проведены их успешные испытания.

Основные возможности программы:

* защита от шаблонного прохождения, благодаря генерируемым значениям, которые невозможно предугадать.
* разработана система экономических взаимоотношений, пользователю предоставляется свобода выбора;
* возможность сбора и предоставлении статистики, сохранение и загрузки своих результатов;
* разработана система экономического развития, с предоставлением пользователю своду действий;
* разработана система генерации случайных событий, подразделенных на несколько подпунктов;

Разработана справочная и эксплуатационная документация:

* руководство программиста;
* руководство пользователя.

Программа разработана с учетом работы непрофессиональных пользователей ПЭВМ. Интерфейс выполнен простым и интуитивным понятным.

В результате создания программы были применены знания по дисциплине «Конструирование программ и языки программирования», закреплен опыт работы в среде Microsoft Visual Studio 2019.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Виды программ и программных документов: ГОСТ 19.101-77 ЕСПД. – М: Стандартинформ, 2005.
2. Общие требования к программным документам: ГОСТ 19.105-78 ЕСПД. – М: Стандартинформ, 2005.
3. Требования к документам, выполненным печатным способом: ГОСТ 19.106-78 ЕСПД. – М: Стандартинформ, 2005.
4. Программа и методика испытаний. Требования к содержанию, оформлению и контролю качества: ГОСТ 19.301-2000 ЕСПД. – М: Стандартинформ, 2005.
5. Описание программы. Требования к содержанию, оформлению и контролю качества: ГОСТ 19.402-2000 ЕСПД. – М: Стандартинформ, 2005.
6. Описание применения. Требования к содержанию и оформлению: ГОСТ 19.502-78 ЕСПД. – М: Стандартинформ, 2005.
7. Руководство программиста. Требования к содержанию и оформлению: ГОСТ 19.504-79 ЕСПД. – М: Стандартинформ, 2005.
8. Отчет о научно-исследовательской работе. Структура и правила оформления: ГОСТ 7.32-2001. – Минск: БелГИСС, 2002.
9. Общие положения: ГОСТ 19.001-77 ЕСПД. – М: Стандартинформ, 2005.
10. Руководство оператора: ГОСТ 19.504-79 ЕСПД. – М: Стандартинформ, 2005.

11 Хейне, Пол. Боуттке, Питер. Причитко, Дэвид «Экономический образ мышления». –М: Издательский дом «Вильямс», 2015.

# **ПРИЛОЖЕНИЕ А**

**Текстовый материал**

Таблица А.1

|  |  |
| --- | --- |
| **Структура хранения данных** | |
| **Название файла** | **Содержимое файла** |
| RecordFile | Пять строчек хранящие цифры, отвечающие за кол-во очков каждого игрока за лучшую игру. |
| GameRulesEn | =======================  "Beginning of game"  The game can involve from 2 to 4 players, each of whom represents the Director:  one of the major international corporations: "Neomir", "sky", "evolution", "Progresst".  Each player gets one chip that represents them during the game, a card.  "Central office" with the logo of its Corporation and Bank notes in the amount of 5000 credits.  Each participant in turn throws a die and moves in the direction of the arrow to the number of cells,  which fell out on the dice.  After the player completes all the necessary actions in this cell, the next player rolls the dice in the same way and moves his chip.  Throughout the game the chips make a circle  several times. Each time a player stops or passes through a cell in whichits "Central office" is located,  and it receives a bonus of 1000 credits from the Bank. The game ends when at least one of the players goes broke. Then the capital is calculated  each of the players (money and buildings) and determine who was the most successful.  =======================  "Central office".  Every Corporation has a Central office from the very beginning of the game. In which country and in which the city it is located, players determine with the help of a cube. The first roll of the die determines  country (clockwise from start: 1 Russia, 2 EU, 3 China etc.). the Second throw is already accurate  defines the cell of the city within this  country (the count is also clockwise) from 1 to 4, if 5 or 6 fall out, flip the cube. |

Продолжение таблицы А.1

|  |  |
| --- | --- |
| **Структура хранения данных** | |
| **Название файла** | **Содержание файла** |
|  | Each player determines the cell of their head office in turn and puts their "Central office" card in it. Offices of different corporations cannot be located in the same cell. The Central office card cannot be handed over to the Bank, etc, If the player's chip stops at the cell of their own Central office, the player's reputation. increases by one point.  =======================  "The reputation of the Corporation."  This is an important indicator that will affect the success of the Transaction and the amount of Revenue. At the beginning of the game, the reputation of all players is normal (0). It will change in. from events that occur with the player.  Reputation is rated from (-4 to +4).  =======================  "Stop on an 'empty' section."  If a player lands on a city square that is not occupied by another player's logo or chip, he gets a chance to make a Deal first. To do this, the player rolls the die twice. The numbers are added, and then the player's reputation is added to them. If the reputation is negative, then its indicator is taken away. The resulting number must be greater than 6. In this case  The transaction is considered concluded. If less, then the Deal is failed. After a successful conclusion  The transaction, the reputation of the player is increased by one point up. If the Deal is unsuccessful, then the reputation reduced by one point. A player can only attempt to make a trade once per turn.  =======================  "Stop on someone else's property."  When a player makes a move and gets to a place where another player's logo is already present, he must pay the owner of the cage an amount equal to the Income of this section. |

Продолжение таблицы А.1

|  |  |
| --- | --- |
| **Структура хранения данных** | |
| **Название файла** | **Содержание файла** |
|  | This amount  it is marked in the city cell, but it is constantly changing depending on the reputation  owner.  Each reputation point adds or subtracts 100 credits from the amount. For example, with a " -2 " reputation, Revenue is reduced by 200 credits, and with a "+4"reputation increases By as much as 400 credits.  =======================  "Stop on your own property."  If the player gets to the site where the logo of his Corporation already exists, he is conducting an audit of his company. To do this the player rolls a die and one of the following events may occur:  1. The company exceeds target. Pay a bonus to employees (to the Bank) equal to,  The company's revenue. Your reputation has increased by one point.  2. The company operates in accordance with the plan. No one needs to pay anything.  3. You found and confiscated the "left" income at the enterprise. Get it from the Bank 1000 credits, but your reputation has decreased by one point.  4. The company operates in accordance with the plan. Don't pay anything to anyone.  5 the Company slightly underperformed the plan. Invest an additional amount in it (to the Bank). equal to half of the company's Revenue.  6. The company incurs serious losses. Invest an additional amount in it (to the Bank), equal to the Income. And your reputation has been reduced by one point. =======================  "If the player runs out of money."  "If the player runs out of money, he can during his turn sell one of your businesses to a Bank for an amount equal to the company's income. |

Продолжение таблицы А.1

|  |  |
| --- | --- |
| **Структура хранения данных** | |
| **Название файла** | **Содержание файла** |
|  | The old token is sent to the Bank, and the site again becomes a no-man's land. If a player has no money or businesses, he leaves the game, unable to withstand competition in the brutal world of business.  The game is over, points are counted and winning places are distributed.  =======================  "If the player stops walking"  If the player remains on the same square (more than 2 moves) and does not attempt to make a deal, He starts losing money for a simple one. With each move, the loss increases by 2 times.For example: 1 move-loss of 100 credits, 2 move-loss of 200 credits, 3 move-loss of 800 credits, etc  =======================  «"SURPRISE " cells".  1) " State transformations»  Having stopped at the "State transformations" box,  the player pays 100 credits to the insurance Fund and receives one random event.  ---------------------------------  2) "Insurance Fund".  If a player falls on this cell, he takes all the money that is currently available  The moment lies in it.  ---------------------------------  3) "international economic police".  Once in the "international economic police" cage, the player Rolls a die and one of the following events may occur:  1. All documents are in order. Just skip the move.  2. All documents are in order. Just skip the move.  3. All documents are in order. Just skip the move.  4. You were fined 200 credits for violations.  5. You were fined 500 credits for violations.  6. You offered a bribe to the inspector. He might agree for 100 credits and after paying, you can continue on your way; |

Продолжение таблицы А.1

|  |  |
| --- | --- |
| **Структура хранения данных** | |
| **Название файла** | **Содержание файла** |
|  | Otherwise, you will end up behind bars with a fine of 500 credits and skipping the next two moves.  ---------------------------------  4) "Advertising".  The "Advertising" cell allows the player to increase their reputation by one point,  by paying 1000 credits to the Bank.  If you don't need to, you don't have to do it.  ---------------------------------  5) "Casino".  By stopping at the "  Casino" cell, the player can play a game of chance.  So, the player and the casino roll the dice and see who gets the most value. If the player wins, if the casino player loses and gives. money in bank.  The amount of winnings (or losses) depends on the difference on both cubes. For example, a player has 6 and the Casino has 2.  This means that the player wins  400 credits (6-2=4).  Or, for example, the player has a 3 and the pointer is 4. It means that the player loses.  100 credits (4-3=1);  If the values on the dice are equal, it means that each "remained the same" and no one pays anyone.  ---------------------------------  6) "Business Symposium".  The player whose chip is in this cage is the organizer of an international business Symposium.  All other players pay him 200 credits, and in return their reputations increase by one point. |
| GameRulesRu | =======================  "Начало игры"  В игре могут участвовать от 2-х до 4-х игроков, каждый из которых представляет директора: |

Продолжение таблицы А.1

|  |  |
| --- | --- |
| **Структура хранения данных** | |
| **Название файла** | **Содержание файла** |
|  | одной из крупных международных корпораций: «Неомир», «Скай», «Эволюшн», «Прогрессик».  Каждый игрок получает одну фишку, которая представляет его во время игры, карточку.  «Центральный офис» с эмблемой своей корпорации и банковские банкноты на сумму 5000 кредов..  Каждый участник по очереди бросает кубик и продвигается по направлению стрелки на то число клеток,  которое выпало на кубиках. После того, как игрок выполнит все необходимые действия в данной клетке,  следующий игрок таким же образом бросает кубики и передвигает свою фишку.  На протяжении всей игры фишки делают круг  несколько раз. Каждый раз, когда игрок останавливается или проходит через клетку, в которой  находится его «Центральный офис», он получает из Банка премию в 1000 кредов.  Игра завершается, когда хотя бы один из игроков разоряется. Тогда подсчитывается капиталы  каждого из игроков (деньги и постройки) и определяют, кто оказался самым удачливым.  =======================  "Центральный офис".  У каждой корпорации с самого начала игры есть центральный офис. В какой стране и в каком  городе он находится, игроки определяют с помощью кубика. Первый бросок кубика определяет  страну (по часовой стрелке от старта: 1 Россия, 2 ЕС, 3 Китай ит.д.). Второй бросок уже точно  определяет клетку города в пределах этой страны (отсчет тоже по движению часовой стрелки)  от 1 до 4, если выпало 5 или 6 перекиньте кубик. |

Продолжение таблицы А.1

|  |  |
| --- | --- |
| **Структура хранения данных** | |
| **Название файла** | **Содержание файла** |
|  | Клетку своего головного офиса по очереди определяет каждый игрок и кладет в нее свою карточку «Центральный офис».  Офисы разных корпораций находиться в одной клетке не могут.  Карточку центрального офиса нельзя сдавать в Банк и т.п,  Если фишка игрока остановится на клетке собственного центрального офиса, репутация игрока.  повышается на один пункт.  =======================  "Репутация корпорации".  'Это важный показатель, который будет влиять на успех Сделки и размеры Дохода..  В начале игры репутация у всех игроков нормальная (0). Она будет изменяться в.  от событий, происходящих с игроком.  Репутация оценивается от (-4 до +4).  =======================  "Остановка на «пустом» участке".  Если игрок попадает на клетку города, которая не занята эмблемой или фишкой другого игрока,  он получает шанс первым заключить Сделку. Для этого игрок дважды бросает кубик.  Числа складываются, после чего к ним прибавляется репутация игрока. Если репутация отрицательная, то ее показатель отнимается. Получившееся в результате число должно быть больше 6. В этом случаи Сделка считается заключенной. Если меньше, то Сделка провалена. После успешного заключения Сделки репутация игрока повышается на один пункт вверх. Если Сделка неудачная, то репутация уменьшается на один пункт. Игрок может попытаться совершить сделку только один раз за ход.  ======================= |

Продолжение таблицы А.1

|  |  |
| --- | --- |
| **Структура хранения данных** | |
| **Название файла** | **Содержание файла** |
|  | «Остановка на чужом участке».  Когда игрок, сделав ход, попадает на участок, где уже имеется эмблема другого игрока,  он должен оплатить владельцу клетки сумму, равную Доходу этого участка. Эта сумма  обозначена в клетке города, однако она постоянно изменяется в зависимости от репутации владельца. Каждый пункт репутации прибавляет или отнимает от суммы 100 кредов.  Например, при репутации « -2» Доход сокращается на 200 кредов, а при репутации «+4»  увеличивается на Целых 400 кредов.  =======================  «Остановка на собственном участке».  Если игрок попадает на участок, где уже имеется эмблема именно его корпорации,  он проводит ревизию своего предприятия. Для этого игрок бросает кубик и ему может выпасть одно из следующих событий:  1. Предприятие перевыполняет план. Выплатите премию сотрудникам (в Банк), равную,  Доходу предприятия. Ваша репутация увеличилось на один пункт.  2. Предприятие работает в соответствии с планом. Никому ничего не надо выплачивать.  3. Вы обнаружили и конфисковали «левый» доход на предприятии. Получите из Банка  1000 кредов, но ваша репутация уменьшилась на один пункт.  4. Предприятие работает в соответствии с планом. Никому ничего не выплачивайте.  5 Предприятие слегка недовыполнило план. Инвестируйте в него дополнительную сумму  (в Банк). равную половине Дохода предприятия.  6. Предприятие несет серьезные убытки. Инвестируйте в него дополнительную сумму  (в Банк), равную Доходу. И ваша репутация уменьшилась на один пункт.  ======================= |

Продолжение таблицы А.1

|  |  |
| --- | --- |
| **Структура хранения данных** | |
| **Название файла** | **Содержание файла** |
|  | «Если у игрока закончились деньги».  Если у игрока закончились деньги, он может во время своего хода продать одно из своих предприятий Банку, за сумму равную доходу предприятия. Старый жетон отправляется в Банк, а участок снова становиться ничейным.  Если у игрока нет ни денег, ни предприятий, он выходит из игры, не выдержав конкуренции в жестоком мире бизнеса. Игра закончена, подсчитываются очки и распределяются победные места.  =======================  «Если игрок прекращает ходить»  Если игрок остается на одной клетке (больше чем на 2 хода) и не предпринимает попыток совершить сделку, Он начинает терять деньги за простой. С каждым ходом потеря увеличивается в 2 раза. К примеру: 1 ход – потеря 100 кредов, 2 ход – потеря 200 кредов, 3 ход – потеря 800 кредов, и т.д  =======================  «СЮРПРИЗНЫЕ» клетки».  1) «Государственные преобразования»  Остановившись на клетке «Государственные преобразования»,  игрок платит 100 кредов, в страховой фонд, и получает одно случайное событие.  ---------------------------------  2) «Страховой фонд».  Если игрок попадает на эту клетку, он забирает себе все деньги, которые в данный  Момент лежат в нем.  ---------------------------------  3) «Международная экономическая полиция».  Попав на клетку «Международная экономическая полиция» игрок бросает кубик и ему может выпасть одно из следующих событий:  1. Все документы в порядке. Просто пропустите ход. 2. Все документы в порядке. Просто пропустите ход. |

Продолжение таблицы А.1

|  |  |
| --- | --- |
| **Структура хранения данных** | |
| **Название файла** | **Содержание файла** |
|  | 3. Все документы в порядке. Просто пропустите ход. 4. Вас оштрафовали за нарушения на 200 кредов. 5. Вас оштрафовали за нарушения на 500 кредов. 6. Вы предложили взятку инспектору. Он может согласиться на 100 кредов и, заплатив, вы можете продолжить путь; Иначе вы попадете за решетку со штрафом 500 кредов и пропусом два следующих хода.  ---------------------------------  4) «Реклама».  Клетка «Реклама» позволяет игроку повысить свою репутацию на один пункт,  заплатив в Банк 1000 кредов. Если нет необходимости, это можно не делать.  ---------------------------------  5) «Казино».  Остановившись на клетке «Казино», игрок может сыграть в азартную игру.  Итак, игрок и казино бросают кубик и смотрят, у кого выпало большее  значение. Если у игрока он выигрывает, если у казино игрок проигрывает и отдает.  деньги в Банк. Размер выигрыша (или проигрыша) зависит от разницы на обоих  кубиках. Например, у игрока выпало 6, а у Казино 2. Значит, игрок выигрывает  400 кредов (6-2=4).  Или, например, у игрока выпало 3, а указино4. Значит, игрок проигрывает.  100 кредов (4-3=1). ;  Если значения на кубиках равны, значит, каждый «остался при своем»  и никто никому не платит.  ---------------------------------  6) «Бизнес-симпозиум».  Игрок, чья фишка оказалась в этой клетке, является организатором международного бизнес-симпозиума. Все остальные игроки платят ему по 200 кредитов, взамен их репутации увеличиваются на один пункт. |

Продолжение таблицы А.1

|  |  |
| --- | --- |
| **Структура хранения данных** | |
| **Название файла** | **Содержание файла** |
| PlaceCountry | 1. Чиновник вымогал взятку. Пришлось отдать 100 в Банк."  2. Правительство:  2.1 России вводит новый налог. Если вам принадлежат клетки в этой стране, заплатите в Банк 100, за каждую.",  2.2 ЕС вводит новый налог. Если вам принадлежат клетки в этой стране, заплатите в Банк 100, за каждую.",  2.3 Китая вводит новый налог. Если вам принадлежат клетки в этой стране, заплатите в Банк 100, за каждую.",  2.4 Канады вводит новый налог. Если вам принадлежат клетки в этой стране, заплатите в Банк 100, за каждую.",  2.5 США вводит новый налог. Если вам принадлежат клетки в этой стране, заплатите в Банк 100, за каждую.",  2.6 Японии вводит новый налог. Если вам принадлежат клетки в этой стране, заплатите в Банк 100, за каждую."  3. Экономические реформы:  3.1 России принесли вам прибыль. Если вы имеете в этой стране клетки, получите из банка 500.",  3.2 ЕС принесли вам прибыль. Если вы имеете в этой стране клетки, получите из банка 500.",  3.3 Китая принесли вам прибыль. Если вы имеете в этой стране клетки, получите из банка 500.",  3.4 Канады принесли вам прибыль. Если вы имеете в этой стране клетки, получите из банка 500.",  3.5 США принесли вам прибыль. Если вы имеете в этой стране клетки, получите из банка 500.",  3.6 Японии принесли вам прибыль. Если вы имеете в этой стране клетки, получите из банка 500."  4. Государственные инвестиции принесли владельцам построек в:  4.1 России по 300, за каждую клетку.", |

Продолжение таблицы А.1

|  |  |
| --- | --- |
| **Структура хранения данных** | |
| **Название файла** | **Содержание файла** |
|  | 4.2 ЕС по 300, за каждую клетку.",  4.3 Китае по 300, за каждую клетку.",  4.4 Канаде по 300, за каждую клетку.",  4.5 США по 300, за каждую клетку.",  4.6 Японии по 300, за каждую клетку."  5. Смена курса:  5.1 России привела к расходам. Все игроки чья фишка сейчас находиться на одной из клеток этой страны, платят в банк 500.",  5.2 ЕС привела к расходам. Все игроки чья фишка сейчас находиться на одной из клеток этой страны, платят в банк 500.",  5.3 Китая привела к расходам. Все игроки чья фишка сейчас находиться на одной из клеток этой страны, платят в банк 500.",  5.4 Канады привела к расходам. Все игроки чья фишка сейчас находиться на одной из клеток этой страны, платят в банк 500.",  5.5 США привела к расходам. Все игроки чья фишка сейчас находиться на одной из клеток этой страны, платят в банк 500.",  5.6 Японии привела к расходам. Все игроки чья фишка сейчас находиться на одной из клеток этой страны, платят в банк 500." |
| FirstMove | Россия: "Москва", "Питер", "Казань", "Новосибирск"  ЕС: "Берлин", "Париж", "Рим", "Вена"  Китай: "Пекин", "Шанхай", "Тяньцзинь", "Шеньян"  Канада: "Оттава", "Ванкувер", "Монреаль", "Торонто"  США: "Хьюстон", "Чикаго", "Лос-Анджелес", "Нью-Йорк"  Япония: "Нагоя", "Осака", "Йокогама", "Токио" |
| PanelEvent | 1. "Предприятие перевыполняет план. Вы выплатили премию сотрудникам, равную доходу предприятия. Ваша репутация увеличена на 1.",  2. "Предприятие работает в соответствии с планом. Никому ничего не выплачиваем.", |

Продолжение таблицы А.1

|  |  |
| --- | --- |
| **Структура хранения данных** | |
| **Название файла** | **Содержание файла** |
|  | 3. "Вы обнаружили и конфисковали 'левый' доход на предприятии. Вы получили из Банка 1000, но репутация уменьшилась на 1.",  4. "Предприятие работает в соответствии с планом. Никому ничего не выплачиваем.",  5. "Предприятие слегка отстает от плана. Инвестируем в него сумму равную половине дохода предприятия.",  6. "Предприятие несет серьезные убытки. Инвестируем в него сумму равную доходу предприятия. Репутация уменьшена на 1." |
| PlacePolice | 1. "Все документы в порядке. Просто пропустите ход.";  2. "Вас оштрафовали за нарушение на 200.";  3. "Вас оштрафовали за нарушение на 500.";  4. "Вы предложили взятку инспектору."; |

Таблица А.2

|  |  |  |
| --- | --- | --- |
| **Структура работы процедур и функций** | | |
| **Название** | **Тип** | **Назначение** |
| **SkipMove – класс процедур и функций** | | |
| Start | процедура | Процедура которая вызывается при загрузке сцены. |
| Update | процедура | Процедура вызываемая при от рисовке каждого кадра |
| But\_SkipMove | процедура | Демонстрация панели пропуска хода. |
| But\_Back | процедура | Сокрытие панели пропуска хода. |
| But\_OK |  | Реализация пропуска хода. |
| **RollTheDice\_Scripts – класс процедур и функций** | | |
| Start | процедура | Процедура которая вызывается при загрузке сцены. |
| Update | процедура | Процедура вызываемая при от рисовке каждого кадра |

Продолжение таблицы А.2

|  |  |  |
| --- | --- | --- |
| **Структура работы процедур и функций** | | |
| **Название** | **Тип** | **Назначение** |
| But\_RollDice | процедура | Процедура генерирующая число отвечающая за кол-во ходов. |
| MovePlayer | процедура | Процедура, отвечающая за перемещение игрока. |
| But\_OK | процедура | Процедура контроля очередности хода |
| **Player\_Script – класс процедур и функций** | | |
| Start | процедура | Процедура, которая вызывается при загрузке сцены. |
| Update | процедура | Процедура, вызываемая при от рисовки каждого кадра |
| CheckDebt | процедура | Процедура проверки на наличие кредита у игрока. |
| CheckStopMove | процедура | Процедура проверки на остановку передвижения игрока. |
| Buy | процедура | Процедура покупки игроком новой компании. |
| **Player – класс процедур и функций** | | |
| OnTriggerEnter | процедура | Процедура на проверку соприкосновения игрока с точкой на карте. |
| **Place\_Script – класс процедур и функций** | | |
| Start | процедура | Процедура, которая вызывается при загрузке сцены. |
| Update | процедура | Процедура, вызываемая при от рисовки каждого кадра |
| **PanelMenu – класс процедур и функций** | | |
| Start | процедура | Процедура, которая вызывается при загрузке сцены. |
| Update | процедура | Процедура, вызываемая при от рисовки каждого кадра |
| But\_Exit | процедура | Процедура выхода в меню. |
| But\_Back | процедура | Процедура закрытия панели. |
| But\_Menu | процедура | Процедура открытия панели. |

Продолжение таблицы А.2

|  |  |  |
| --- | --- | --- |
| **Структура работы процедур и функций** | | |
| **Название** | **Тип** | **Назначение** |
| But\_PlaceSaleBank | процедура | Процедура открытия панели продажи компании игрока. |
| **PanelEvent – класс процедур и функций** | | |
| Start | процедура | Процедура, которая вызывается при загрузке сцены. |
| Update | процедура | Процедура, вызываемая при от рисовки каждого кадра |
| But\_RanMessage | процедура | Процедура генерирования при остановке на собственном участке |
| But\_Back | процедура | Процедура закрытия панели |
| **MassPlace\_Script – класс процедур и функций** | | |
| Start | процедура | Процедура, которая вызывается при загрузке сцены. |
| Update | процедура | Процедура, вызываемая при от рисовки каждого кадра |
| **FirstMove – класс процедур и функций** | | |
| Start | процедура | Процедура, которая вызывается при загрузке сцены. |
| Update | процедура | Процедура, вызываемая при от рисовки каждого кадра |
| But\_Ok | процедура | Процедура сохранения позиций центрального офиса. |
| But\_RanCity | процедура | Процедура генерирования города для размещения центрального офиса. |
| But\_RanCountry | процедура | Процедура генерирования страны для размещения центрального офиса. |
| **EventStopMove – класс процедур и функций** | | |
| Start | процедура | Процедура, которая вызывается при загрузке сцены. |
| Update | процедура | Процедура, вызываемая при от рисовки каждого кадра |
| But\_DebtPlayer | процедура | Процедура наложение штрафа за прекращение движения. |

Продолжение таблицы А.2

|  |  |  |
| --- | --- | --- |
| **Структура работы процедур и функций** | | |
| **Название** | **Тип** | **Назначение** |
| **EndGame\_Script – класс процедур и функций** | | |
| Start | процедура | Процедура, которая вызывается при загрузке сцены. |
| Update | процедура | Процедура, вызываемая при от рисовки каждого кадра |
| EndGame | процедура | Процедура размещения призовых мест. |
| UpData | процедура | Процедура обновления кредита игрока. |
| But\_OK | процедура | Процедура выплаты и закрытия кредита игрока. |
| But\_Exit | процедура | Процедура сохранения результатов игры. |
| **Deal – класс процедур и функций** | | |
| But\_Ok | процедура | Процедура открытия панели. |
| UpdateDataInDeal | процедура | Процедура обновления данных. |
| But\_ShowBargaining | процедура | Процедура открытия панели бросания кубика. |
| But\_No |  | Процедура закрытия панели. |
| **Bargaining – класс процедур и функций** | | |
| Start | процедура | Процедура, которая вызывается при загрузке сцены. |
| Update | процедура | Процедура, вызываемая при от рисовки каждого кадра |
| But\_OK | процедура | Процедура вызова покупки компании. |
| But\_RandNumber | процедура | Процедура генерирования чисел для покупки компании. |
| **PlaceAd – класс процедур и функций** | | |
| Start | процедура | Процедура, которая вызывается при загрузке сцены. |
| Update | процедура | Процедура, вызываемая при от рисовки каждого кадра |
| But\_Ok | процедура | Процедура покупки рекламы. |
| But\_No | процедура | Процедура закрытия панели. |
| **PlaceBusSym – класс процедур и функций** | | |

Продолжение таблицы А.2

|  |  |  |
| --- | --- | --- |
| **Структура работы процедур и функций** | | |
| **Название** | **Тип** | **Назначение** |
| Start | процедура | Процедура, которая вызывается при загрузке сцены. |
| Update | процедура | Процедура, вызываемая при от рисовки каждого кадра |
| But\_Ok | процедура | Процедура организации бизнес симпозиума. |
| **PlaceCasino – класс процедур и функций** | | |
| Start | процедура | Процедура, которая вызывается при загрузке сцены. |
| Update | процедура | Процедура, вызываемая при от рисовки каждого кадра |
| But\_RanNum | процедура | Генерация чисел со стороны игрока |
| But\_Ok | процедура | Процедура открытия панелей. |
| But\_No | процедура | Процедура закрытия панели. |
| **PlaceCountry – класс процедур и функций** | | |
| Start | процедура | Процедура, которая вызывается при загрузке сцены. |
| Update | процедура | Процедура, вызываемая при от рисовки каждого кадра |
| But\_RanMessage | процедура | Процедура генерирования событий при остановке на пустом участке. |
| But\_Back | процедура | Процедура закрытия панели. |
| **PlaceFond – класс процедур и функций** | | |
| Start | процедура | Процедура, которая вызывается при загрузке сцены. |
| Update | процедура | Процедура, вызываемая при от рисовки каждого кадра |
| But\_Ok | процедура | Процедура получения денег фонда. |
| **PlacePolice – класс процедур и функций** | | |
| Start | процедура | Процедура, которая вызывается при загрузке сцены. |
| Update | процедура | Процедура, вызываемая при от рисовки каждого кадра |

Продолжение таблицы А.2

|  |  |  |
| --- | --- | --- |
| **Структура работы процедур и функций** | | |
| **Название** | **Тип** | **Назначение** |
| But\_RanMessage | процедура | Процедура генерации события экономической полиции. |
| But\_Back | процедура | Процедура закрытия панели. |
| **PanelSaleBank – класс процедур и функций** | | |
| Start | процедура | Процедура, которая вызывается при загрузке сцены. |
| Update | процедура | Процедура, вызываемая при от рисовки каждого кадра |
| But\_But\_OK | процедура | Процедура продажи компании игрока. |
| But\_Back | процедура | Процедура закрытия панели. |
| **PanelSalePlace – класс процедур и функций** | | |
| Start | процедура | Процедура, которая вызывается при загрузке сцены. |
| Update | процедура | Процедура, вызываемая при от рисовки каждого кадра |
| But\_OK | процедура | Процедура вызова продажи компании. |
| ChangeValue | процедура | Процедура выбора страны расположения компании на продажу. |
| ChangeValue2 | процедура | Процедура выбора города расположения компании на продажу. |
| But\_Back | процедура | Процедура закрытия панели. |
| **InterfaceScripts – класс процедур и функций** | | |
| Start | процедура | Процедура, которая вызывается при загрузке сцены. |
| Update | процедура | Процедура, вызываемая при от рисовки каждого кадра |
| But\_GameRules | процедура | Процедура открытия панели с правилами игры. |
| But\_ClosePanelMessage | процедура | Процедура закрытия панели сообщения. |
| But\_Record | процедура | Процедура открытия панели с рекордом игры. |
| But\_Start | процедура | Процедура старта игры. |

Продолжение таблицы А.2

|  |  |  |
| --- | --- | --- |
| **Структура работы процедур и функций** | | |
| **Название** | **Тип** | **Назначение** |
| But\_Exit | процедура | Процедура выхода из игры. |
| **PressSpace – класс процедур и функций** | | |
| Start | процедура | Процедура, которая вызывается при загрузке сцены. |
| Update | процедура | Процедура, вызываемая при от рисовки каждого кадра |

# **ПРИЛОЖЕНИЕ Б**

**Графическая схема алгоритма**

(блок схемы пока фикция)

RndShen

Klient

Kolday, pos

Новостей про цены нет!

kolpos:= 0;

kolday > 3

Нет

Да

Sob1

(rep)

EasyMany (many);

randomize;

A

(MaksProv = false) **and** (ProvPer = false)

MaksPer (many,kolmany,NapR,NapB,Napg,NapY,NapW, maksprov,ProvPer,ShetPer,ShetPer2);

Podskazki

= true

Podskazki = true

'Посетитель проявил сарказм!'

time<12.00

Нет

Да

tpos:=random (11);

TimeOtl (time, timeot)

time

tpos **mod**2=0

Нет

Да

Да

A

kolpos:=kolpos+1;

tslov:=random (11);

tslov **mod**2=0

Нет

Да

Нет

Да

Посетитель поздоровался!

SayHello;

Нет

Да

NoSayHello;

B

you

'- ...'

you:= 3

you = 3

Нет

Да

Amotion1(you, tslov, rep);

B

Нет

Да

(you <> 1) **and** (you <> 2) **and** (you <> 3)

DialogPos (kolmany, rep, NapR, NapB, NapG, NapY, NapW, time, timeot, kolday, levellavka);

you <> 3

Нет

Да

Amotion1(you, tslov, rep);

DialogPos (kolmany, rep, NapR, NapB, NapG, NapY, NapW, time, timeot, kolday, levellavka);

Нет

Да

kolday<= 8

Нет

Да

kolday

**mod** 2 = 0

ShetHome:= ShetHome + 25

kolday>8

Да

shethome:= 100 + (random (35) + 15);

ShetFamily:= 4 \* (random (kolday) + 2);

many:= many - (ShetHome +ShetFamily);

Data, kolday, pos

Kolpos, rep,many

kolmany

ShetHome, Shetfamily

Нет

Да

kolday **mod** 2 = 0

С

AchiveBigManyDay (many, kolmany);

AchiveMany (kolmany, many);

AchiveREP (rep);

AchiveKolday (kolday);

many:= many + kolmany;

globalMany:= globalMany + kolmany;

pos:= pos + kolpos;

NapR, NapB, NapG

С

NapY, NapW,p

Нет

Да

(p <> 1) **and** (p <> 2)

p:= 0

p=1

Нет

Да

BuyRes(many,napr,napb,napg,napy,napw,rep)

Нет

Да

(kolday > 2) **and** (kolday **mod** 7 = 0)

globalMany:= 0;

kolday:= kolday + 1;

kolmany:=0;

kolpos:= 0;

time:=8.00;

timeot:=8.50;

D

HozLavka (kolday, many);

SborNalog(kolday,globalMany,many);

AchiveOtrManyDay(kolmany,many);

Bank (many);

D

levelLavka

Нет

Да

Это максимальный уровень!

(levelLavka+1)\*50)

l

Нет

Да

l:= 2

Нет

Да

Нет

Да

Вам не хватает денег!

levellavka:=levellavka + 1;

many:= many - (levelLavka \* 50);

M

(l = 1) **and** (many < ((levelLavka + 1) \* 50))

(l = 1) **and** (many > ((levelLavka + 1) \* 50))

levellavka=5

FireLavka(many,kolday,data);

RanGrab (many,NapR,NapB,Napg,NapY,NapW);

DateOtl (data, dataot);

Menu;

M

maksprov:= false;

Конец

Records (namebar,many,rep,pos,kolday,levellavka);

Records

i:= 0;

n:= 0;

Нет

Да

reset (f);

i:= i + 1;

i=1

Нет

Да

readln (f,mas[i])

readln (f, s)

close (f);

bal:=(many **div** 10)+rep +pos+kolday+levellavka;

bal>mas[7]

Нет

Да

i:= 0;

mas[2]:=many;mas[3]:=rep; mas[4]:=pos

mas[5]:=kolday;mas[6]:=levelLavka; mas[7]:= bal;

i:= 1;

B

assign (f, 'E:\Задания\program\Kyrsovai\1TF\_\Records.txt');

**not** eof (f)

rewrite (f);

B

I<7

Нет

Да

I=1

Нет

Да

writeln (f,name)

writeln (f,mas[i]);

i:= i + 1;

close (f);

assign (f, 'E:\Задания\program\Kyrsovai\1TF\_\Records.txt');

assign (f, 'E:\Задания\program\Kyrsovai\1TF\_\Records.txt');

rewrite (f);

I<7

Нет

Да

I=1

Нет

Да

writeln (f,mas[i]);

writeln (f, s)

i:= i + 1;

close (f);

Конец

TimeOtl

ti>= ti2

Да

Нет

ti:=ti+0.4;

ti2:=ti2+1;

ti:= ti + 0.2;

Конец

dat= 1.13

Да

Нет

dat:=1.01;

dat2:=30.01;

DateOtl

dat >=dat2

Да

Нет

dat:=dat-29;

dat:=dat+0.01;

dat:= dat + 1.0;

Конец

dat2:= dat2 + 0.01;

# **ПРИЛОЖЕНИЕ B**

**Текст программы**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class SkipMove : MonoBehaviour

{

public GameObject panel;

public GameObject Mycamera;

private List<GameObject> Mass\_Player;

private bool prov = true;

// Start is called before the first frame update

void Start()

{

prov = true;

}

// Update is called once per frame

void Update()

{

}

public void But\_SkipMove()

{

if (prov == true)

{

panel.SetActive(true);

prov = false;

}

}

public void But\_Back()

{

if(prov == false)

{

panel.SetActive(false);

prov = true;

}

}

public void But\_OK()

{

Mass\_Player = RollTheDice\_Scripts.Mass\_Player;

//=====================================================================

if (RollTheDice\_Scripts.ProvMove == false)

{

if (RollTheDice\_Scripts.Score == 4) RollTheDice\_Scripts.Score = 1;

else RollTheDice\_Scripts.Score++;

if (Player\_Script.Mass\_Player[Player\_Script.Score].SkipMovePolice == 0)

{

Player\_Script.Mass\_Player[Player\_Script.Score].StopMovePlayer++;

}

}

else RollTheDice\_Scripts.ProvMove = false;

if (Player\_Script.Mass\_Player[Player\_Script.Score].CheckDebt == true)

{

EndGame\_Script.massdebtPlayer[Player\_Script.Score][1]--;

}

foreach (GameObject player in Mass\_Player)

{

if (RollTheDice\_Scripts.Score == 1 && player.tag == "Player2")

{

if (Player\_Script.Mass\_Player[Player\_Script.Score].SkipMovePolice != 0)

{

Player\_Script.Mass\_Player[Player\_Script.Score].SkipMovePolice -= 1;

}

var position = player.transform.position;

position.z = -100;

Mycamera.transform.position = position;

Player\_Script.Score = 1;

}

if (RollTheDice\_Scripts.Score == 2 && player.tag == "Player3")

{

if (Player\_Script.Mass\_Player[Player\_Script.Score].SkipMovePolice != 0)

{

Player\_Script.Mass\_Player[Player\_Script.Score].SkipMovePolice -= 1;

}

var position = player.transform.position;

position.z = -100;

Mycamera.transform.position = position;

Player\_Script.Score = 2;

}

if (RollTheDice\_Scripts.Score == 3 && player.tag == "Player4")

{

if (Player\_Script.Mass\_Player[Player\_Script.Score].SkipMovePolice != 0)

{

Player\_Script.Mass\_Player[Player\_Script.Score].SkipMovePolice -= 1;

}

var position = player.transform.position;

position.z = -100;

Mycamera.transform.position = position;

Player\_Script.Score = 3;

}

if (RollTheDice\_Scripts.Score == 4 && player.tag == "Player1")

{

if (Player\_Script.Mass\_Player[Player\_Script.Score].SkipMovePolice != 0)

{

Player\_Script.Mass\_Player[Player\_Script.Score].SkipMovePolice -= 1;

}

var position = player.transform.position;

position.z = -100;

Mycamera.transform.position = position;

Player\_Script.Score = 0;

}

if (Player\_Script.Mass\_Player[Player\_Script.Score].CheckDebt == true || Player\_Script.Mass\_Player[Player\_Script.Score].Money < 0)

{

EndGame\_Script.panelDeptClose.SetActive(true);

EndGame\_Script.UpData();

}

else EndGame\_Script.panelDeptClose.SetActive(false);

Player\_Script.CheckStopMovePlayer = false;

Player\_Script.CheckDebtPlayer = false;

RollTheDice\_Scripts.Prov = true;

panel.SetActive(false);

Deal.Check = false;

prov = true;

}

}

}

using System;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

public class RollTheDice\_Scripts : MonoBehaviour

{

public GameObject player1;

public GameObject player2;

public GameObject player3;

public GameObject player4;

public GameObject panelRoll;

public GameObject PanelFirstMove;

public GameObject Camera;

public Text Ran\_text;

private int Rand\_Num;

public static bool Prov = true, ProvMove = false;

public static int Score = 0;

public static List<GameObject> Mass\_Player;

// Start is called before the first frame update

void Start()

{

Rand\_Num = 0;

Prov = true;

ProvMove = false;

Score = 0;

}

// Update is called once per frame

void Update()

{

Mass\_Player = new List<GameObject>() { player1, player2, player3, player4};

}

public void But\_RollDice()

{

if (Player\_Script.Mass\_Player[Player\_Script.Score].SkipMovePolice == 0)

{

//Debug.Log("First move = " + Player\_Script.Mass\_Player[Player\_Script.Score].First\_Move);

if (Player\_Script.Mass\_Player[Player\_Script.Score].First\_Move == false)

{

PanelFirstMove.SetActive(true);

Player\_Script.Mass\_Player[Player\_Script.Score].First\_Move = true;

}

else

{

System.Random rnd = new System.Random();

if (Prov)

{

Player\_Script.Mass\_Player[Player\_Script.Score].StopMovePlayer = 0;

panelRoll.SetActive(true);

Rand\_Num = rnd.Next(1, 7) + rnd.Next(1, 7);

Ran\_text.text = Rand\_Num.ToString();

Prov = false;

}

}

}

}

public void MovePlayer(GameObject player)

{

for (int i = 0; i < Rand\_Num; i++)

{

if (player.transform.position.x < 200 && player.transform.position.x > -160 && player.transform.position.y < -155 && player.transform.position.y > -200)

{

Debug.Log(" k = 2");

var position = player.transform.position;

position.x -= 40f;

player.transform.position = position;

position.z = -100;

Camera.transform.position = position;

}

else if (player.transform.position.y > -200 && player.transform.position.y < 155 && player.transform.position.x > -200 && player.transform.position.x < -160)

{

Debug.Log(" k = 3");

var position = player.transform.position;

position.y += 45f;

player.transform.position = position;

position.z = -100;

Camera.transform.position = position;

}

else if (player.transform.position.x > -200 && player.transform.position.x < 160 && player.transform.position.y < 200 && player.transform.position.y > 155)

{

Debug.Log(" k = 4");

var position = player.transform.position;

position.x += 40f;

player.transform.position = position;

position.z = -100;

Camera.transform.position = position;

}

else if (player.transform.position.y < 200 && player.transform.position.y > -155 && player.transform.position.x > 160 && player.transform.position.x < 200)

{

Debug.Log(" k = 1");

var position = player.transform.position;

position.y -= 45f;

player.transform.position = position;

position.z = -100;

Camera.transform.position = position;

}

}

}

public void But\_OK()

{

panelRoll.SetActive(false);

if (Score != 4) Score++;

else Score = 1;

ProvMove = true;

foreach (GameObject player in Mass\_Player)

{

if (Score == 1 && player.tag == "Player1")

{

MovePlayer(player);

}

else if (Score == 2 && player.tag == "Player2")

{

MovePlayer(player);

}

else if (Score == 3 && player.tag == "Player3")

{

MovePlayer(player);

}

else if (Score == 4 && player.tag == "Player4")

{

MovePlayer(player);

}

}

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

public class Player\_Script : MonoBehaviour

{

//

public static Player[] Mass\_Player;

public Player player1;

public Player player2;

public Player player3;

public Player player4;

private string Name;

private int Money = 5000;

[HideInInspector]

public int Reputation = 0;

private int Chips = 13;

[HideInInspector]

public static bool CheckStopMovePlayer = false, CheckDebtPlayer = false;

public Text name\_;

public Text money;

public Text chips;

public Text reputation;

Transform Point;

//[HideInInspector]

//public static Collider other;

[HideInInspector]

public static int Score = 0;

//public GameObject deal;

// Start is called before the first frame update

void Start()

{

Name = "";

Chips = 13;

Score = 0;

Reputation = 0;

CheckStopMovePlayer = false;

}

// Update is called once per frame

void Update()

{

Mass\_Player = new Player[] { player1, player2, player3, player4 };

Name = Mass\_Player[Score].Name;

Money = Mass\_Player[Score].Money;

Chips = Mass\_Player[Score].Chips;

Reputation = Mass\_Player[Score].Reputation;

Point = Mass\_Player[Score].Point;

//deal = new Deal();

name\_.text = Name;

money.text = Money.ToString();

chips.text = Chips.ToString();

reputation.text = Reputation.ToString();

if (CheckStopMovePlayer == false && Mass\_Player[Score].StopMovePlayer > 1 && Mass\_Player[Score].SkipMovePolice == 0 && RollTheDice\_Scripts.ProvMove == false) CheckStopMove();

else if (Mass\_Player[Score].StopMovePlayer == 0) EventStopMove.massdebtPlayer[Score] = 50;

}

public static void CheckDebt()

{

if (CheckDebtPlayer == false && (Mass\_Player[Score].Money < 0 || Mass\_Player[Score].CheckDebt == true))

{

Debug.Log("CheckDebt");

Mass\_Player[Score].CheckDebt = true;

CheckDebtPlayer = true;

EndGame\_Script.UpData();

EndGame\_Script.ProvSetPanel = true;

}

}

void CheckStopMove()

{

CheckStopMovePlayer = true;

EventStopMove.ProvSetPanel = true;

}

public static void Buy(bool CheckOnAgree)

{

if (CheckOnAgree == true)

{

Mass\_Player[Score].Money += Mass\_Player[Score].other.gameObject.GetComponent<Place\_Script>().money;

//Deal.Status = true;

Mass\_Player[Score].StopMovePlayer = 0;

Mass\_Player[Score].other.gameObject.GetComponent<Place\_Script>().status = true;

Mass\_Player[Score].other.gameObject.GetComponent<Place\_Script>().TAG = Mass\_Player[Score].tag;

Mass\_Player[Score].Chips--;

if (Mass\_Player[Score].Reputation < 4) Mass\_Player[Score].Reputation++;

//Instantiate(Mass\_Player[Score].Point, other.transform.position, Quaternion.identity);

Transform point = Instantiate(Mass\_Player[Score].Point, Mass\_Player[Score].other.transform.position, Quaternion.identity) as Transform;

if (Mass\_Player[Score].First\_Сaptured == true)

{

Mass\_Player[Score].MassPlacePlayer = new List<Place\_Script>();

Mass\_Player[Score].First\_Сaptured = false;

Mass\_Player[Score].MassPointPlayer = new List<GameObject>();

}

Mass\_Player[Score].MassPointPlayer.Add(point.gameObject.gameObject);

Mass\_Player[Score].MassPlacePlayer.Add(Mass\_Player[Score].other.gameObject.GetComponent<Place\_Script>());

Debug.Log("Mass = " + Mass\_Player[Score].MassPlacePlayer[0].Indx1 + " + " + Mass\_Player[Score].MassPlacePlayer[0].Indx2);

}

else

{

if (Mass\_Player[Score].Reputation > -4) Mass\_Player[Score].Reputation--;

}

}

//public void Move()

//{

// //Debug.LogError("Ere = " + Mass\_Player[Score].other.gameObject.GetComponent<Place\_Script>().TAG);

// Deal.player = this;

// Deal.TagPlace = other.gameObject.GetComponent<Place\_Script>().tag;

// Deal.Status = other.gameObject.GetComponent<Place\_Script>().status;

// Deal.Check = false;

//}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Player : MonoBehaviour

{

public string Name = "Red";

public int Money = 0, MaxMoney = 5000;

public int Reputation = 0;

public int Chips = 13;

[HideInInspector]

public bool First\_Move = true, First\_Сaptured = true, CheckDebt = false;

public Transform Point;

public Transform CenterPoint;

[HideInInspector]

public static bool CheckTrigger = false;

[HideInInspector]

public List<Place\_Script> MassPlacePlayer;

[HideInInspector]

public List<GameObject> MassPointPlayer;

[HideInInspector]

public int StopMovePlayer = 0, SkipMovePolice = 0;

[HideInInspector]

public Collider other;

void OnTriggerEnter(Collider other)

{

this.other = other;

if (other.tag == "Place")

{

Debug.Log("Conct = " + other.gameObject.GetComponent<Place\_Script>().Indx1);

if (other.gameObject.GetComponent<Place\_Script>().status == false && Chips > 0)

{

//Player\_Script.other = other;

//(new Player\_Script()).Move();

Deal.UpdateDataInDeal();

}

else if (other.gameObject.GetComponent<Place\_Script>().status == true && other.gameObject.GetComponent<Place\_Script>().TAG != this.tag)

{

foreach(Player player in Player\_Script.Mass\_Player)

{

if (player.tag == other.gameObject.GetComponent<Place\_Script>().TAG)

{

player.Money += other.gameObject.GetComponent<Place\_Script>().money + (player.Reputation \* 100);

this.Money -= other.gameObject.GetComponent<Place\_Script>().money + (player.Reputation \* 100);

}

}

}

else if (other.gameObject.GetComponent<Place\_Script>().status == true && other.gameObject.GetComponent<Place\_Script>().TAG == this.tag)

{

if (other.gameObject.GetComponent<Place\_Script>().CenterPlacePoint == true)

{

this.Money += 1000;

if (this.Reputation < 4) this.Reputation += 1;

}

else

{

PanelEvent.ProvSetPanel = true;

}

}

}

else if (other.tag == "PlaceCountry")

{

Debug.Log("Country");

PlaceCountry.ProvSetPanel = true;

}

else if (other.tag == "PlaceAd")

{

Debug.Log("Ad");

PlaceAd.ProvSetPanel = true;

}

else if (other.tag == "PlaceBusSym")

{

Debug.Log("PlaceBusSym");

PlaceBusSym.ProvSetPanel = true;

}

else if (other.tag == "PlaceFond")

{

Debug.Log("PlaceFond");

PlaceFond.ProvSetPanel = true;

}

else if (other.tag == "PlaceCasino")

{

Debug.Log("PlaceCasino");

PlaceCasino.ProvSetPanel = true;

}

else if (other.tag == "PlacePolice")

{

Debug.Log("PlacePolice");

PlacePolice.ProvSetPanel = true;

}

Player\_Script.CheckDebt();

if(MaxMoney < Money)

{

MaxMoney = Money;

}

}

}

using System;

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Place\_Script : MonoBehaviour

{

public int Indx1, Indx2;

public int money;

public string TAG = "";

[HideInInspector]

public bool CenterPlacePoint = false;

// [HideInInspector]

public bool status = false;

// Start is called before the first frame update

void Start()

{

}

// Update is called once per frame

void Update()

{

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine.SceneManagement;

using UnityEngine;

using UnityEngine.UI;

public class PanelMenu : MonoBehaviour

{

public static int MoneyBank, MoneyFond;

public Text TextMoneyBank;

public Text TextMoneyFond;

public GameObject panelMenu;

// Start is called before the first frame update

void Start()

{

MoneyBank = 5000;

MoneyFond = 0;

}

// Update is called once per frame

void Update()

{

TextMoneyBank.text = MoneyBank.ToString();

TextMoneyFond.text = MoneyFond.ToString();

}

public void But\_Exit()

{

//SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);

SceneManager.LoadScene("MenuScene");

}

public void But\_Back()

{

panelMenu.SetActive(false);

}

public void But\_Menu()

{

if (panelMenu.activeSelf == false) panelMenu.SetActive(true);

else panelMenu.SetActive(false);

}

public void But\_PlaceSaleBank()

{

if (Player\_Script.Mass\_Player[Player\_Script.Score].Money < 100)

{

PanelSalePlace.ProvSetPanel = true;

PanelSalePlace.ProvBank = true;

panelMenu.SetActive(false);

}

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

public class PanelEvent : MonoBehaviour

{

public GameObject panelEvent;

public Text TextMessage;

[HideInInspector]

public static bool ProvSetPanel = false, ProvSetMessage = false;

int Indx;

string[] MassMessage;

// Start is called before the first frame update

void Start()

{

ProvSetPanel = false;

ProvSetMessage = false;

MassMessage = new string[6]

{

"Предприятие перевыполняет план. Вы выплатили премию сотрудникам, равную доходу предприятия. Ваша репутация увеличена на 1.",

"Предприятие работает в соответствии с планом. Никому ничего не выплачиваем.",

"Вы обнаружили и конфисковали 'левый' доход на предприятии. Вы получили из Банка 1000, но репутация уменьшилась на 1.",

"Предприятие работает в соответствии с планом. Никому ничего не выплачиваем.",

"Предприятие слегка отстает от плана. Инвестируем в него сумму равную половине дохода предприятия.",

"Предприятие несет серьезные убытки. Инвестируем в него сумму равную доходу предприятия. Репутация уменьшена на 1."

};

}

// Update is called once per frame

void Update()

{

if (ProvSetPanel == true)

{

panelEvent.SetActive(true);

ProvSetPanel = false;

TextMessage.text = "";

}

if (ProvSetMessage == true)

{

TextMessage.text = MassMessage[Indx];

}

}

public void But\_RanMessage()

{

if (ProvSetMessage == false)

{

System.Random rnd = new System.Random();

Indx = rnd.Next(0, 6);

switch (Indx)

{

case 0:

PanelMenu.MoneyBank += Player\_Script.Mass\_Player[Player\_Script.Score].other.gameObject.GetComponent<Place\_Script>().money;

Player\_Script.Mass\_Player[Player\_Script.Score].Money -= Player\_Script.Mass\_Player[Player\_Script.Score].other.gameObject.GetComponent<Place\_Script>().money;

if(Player\_Script.Mass\_Player[Player\_Script.Score].Reputation < 4) Player\_Script.Mass\_Player[Player\_Script.Score].Reputation++;

break;

case 2:

PanelMenu.MoneyBank -= 1000;

Player\_Script.Mass\_Player[Player\_Script.Score].Money += 1000;

if (Player\_Script.Mass\_Player[Player\_Script.Score].Reputation > -4) Player\_Script.Mass\_Player[Player\_Script.Score].Reputation--;

break;

case 4:

PanelMenu.MoneyBank += Player\_Script.Mass\_Player[Player\_Script.Score].other.gameObject.GetComponent<Place\_Script>().money / 2;

Player\_Script.Mass\_Player[Player\_Script.Score].Money -= Player\_Script.Mass\_Player[Player\_Script.Score].other.gameObject.GetComponent<Place\_Script>().money / 2;

break;

case 5:

PanelMenu.MoneyBank += Player\_Script.Mass\_Player[Player\_Script.Score].other.gameObject.GetComponent<Place\_Script>().money;

Player\_Script.Mass\_Player[Player\_Script.Score].Money -= Player\_Script.Mass\_Player[Player\_Script.Score].other.gameObject.GetComponent<Place\_Script>().money;

if (Player\_Script.Mass\_Player[Player\_Script.Score].Reputation > -4) Player\_Script.Mass\_Player[Player\_Script.Score].Reputation--;

break;

}

ProvSetMessage = true;

}

}

public void But\_Back()

{

if (ProvSetMessage == true)

{

panelEvent.SetActive(false);

ProvSetMessage = false;

}

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class MassPlace\_Script : MonoBehaviour

{

public Place\_Script place1\_1;

public Place\_Script place1\_2;

public Place\_Script place1\_3;

public Place\_Script place1\_4;

public Place\_Script place2\_1;

public Place\_Script place2\_2;

public Place\_Script place2\_3;

public Place\_Script place2\_4;

public Place\_Script place3\_1;

public Place\_Script place3\_2;

public Place\_Script place3\_3;

public Place\_Script place3\_4;

public Place\_Script place4\_1;

public Place\_Script place4\_2;

public Place\_Script place4\_3;

public Place\_Script place4\_4;

public Place\_Script place5\_1;

public Place\_Script place5\_2;

public Place\_Script place5\_3;

public Place\_Script place5\_4;

public Place\_Script place6\_1;

public Place\_Script place6\_2;

public Place\_Script place6\_3;

public Place\_Script place6\_4;

[HideInInspector]

public static Place\_Script[] MassPlace;

// Start is called before the first frame update

void Start()

{

}

// Update is called once per frame

void Update()

{

MassPlace = new Place\_Script[]

{

place1\_1, place1\_2, place1\_3, place1\_4,

place2\_1, place2\_2, place2\_3, place2\_4,

place3\_1, place3\_2, place3\_3, place3\_4,

place4\_1, place4\_2, place4\_3, place4\_4,

place5\_1, place5\_2, place5\_3, place5\_4,

place6\_1, place6\_2, place6\_3, place6\_4

};

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

public class FirstMove : MonoBehaviour

{

public GameObject Panel;

public GameObject ColorPanel;

public Text TextCountry;

public Text TextCity;

public Button BRanCountry;

public Button BRanCity;

public Button BOK;

string textcountry = "None";

string textcity = "None";

bool ProvContry = false;

bool ProvCity = false;

int IndxCountry, IndxCity, IndxCityMassPlace;

public static string[][] Mass;

ColorBlock mycolor;

// Start is called before the first frame update

void Start()

{

textcountry = "None";

textcity = "None";

ProvContry = false;

ProvCity = false;

IndxCountry = 0;

IndxCity = 0;

IndxCityMassPlace = 0;

Mass = new string[6][];

Mass[0] = new string[4] { "Москва", "Питер", "Казань", "Новосибирск"};

Mass[1] = new string[4] { "Берлин", "Париж", "Рим", "Вена" };

Mass[2] = new string[4] { "Пекин", "Шанхай", "Тяньцзинь", "Шеньян" };

Mass[3] = new string[4] { "Оттава", "Ванкувер", "Монреаль", "Торонто" };

Mass[4] = new string[4] { "Хьюстон", "Чикаго", "Лос-Анджелес", "Нью-Йорк" };

Mass[5] = new string[4] { "Нагоя", "Осака", "Йокогама", "Токио" };

}

// Update is called once per frame

void Update()

{

TextCountry.text = textcountry;

TextCity.text = textcity;

if (ProvContry == true) BRanCountry.GetComponent<Image>().color = Color.green;

else BRanCountry.GetComponent<Image>().color = Color.white;

if (ProvCity == true)

{

BRanCity.GetComponent<Image>().color = Color.green;

ColorPanel.GetComponent<Image>().color = Color.green;

mycolor = BOK.GetComponent<Button>().colors;

mycolor.pressedColor = Color.green;

BOK.GetComponent<Button>().colors = mycolor;

}

else

{

//mycolor = BOK.GetComponent<Button>().colors;

//mycolor.pressedColor = Color.red;

//BOK.GetComponent<Button>().colors = mycolor;

//mycolor = BRanCity.GetComponent<Button>().colors;

//mycolor.pressedColor = Color.red;

//BRanCity.GetComponent<Button>().colors = mycolor;

ColorPanel.GetComponent<Image>().color = Color.red;

BRanCity.GetComponent<Image>().color = Color.white;

}

}

public void But\_Ok()

{

if (ProvCity == true)

{

foreach(Place\_Script place in MassPlace\_Script.MassPlace)

{

if (place.Indx1 == IndxCountry+1 && place.Indx2 == IndxCity+1)

{

Instantiate(Player\_Script.Mass\_Player[Player\_Script.Score].CenterPoint, place.transform.position, Quaternion.identity);

place.status = true;

place.TAG = Player\_Script.Mass\_Player[Player\_Script.Score].tag;

place.CenterPlacePoint = true;

}

}

textcity = "";

textcountry = "";

ProvCity = false;

ProvContry = false;

Panel.SetActive(false);

}

}

int ind = 0;

public void But\_RanCity()

{

if (ProvContry == true && ProvCity == false)

{

System.Random rnd = new System.Random();

IndxCity = rnd.Next(0, 4);

if (MassPlace\_Script.MassPlace[IndxCity + IndxCityMassPlace].status == false)

{

textcity = Mass[IndxCountry][IndxCity];

ProvCity = true;

ind = 0;

}

ind++;

if (ind >= 4)

{

ProvContry = false;

ind = 0;

}

}

}

public void But\_RanCountry()

{

if (ProvContry == false)

{

System.Random rnd = new System.Random();

IndxCountry = rnd.Next(0, 6);

switch (IndxCountry)

{

case 0:

textcountry = "Россия";

IndxCityMassPlace = 0;

break;

case 1:

textcountry = "ЕС";

IndxCityMassPlace = 4;

break;

case 2:

textcountry = "Китай";

IndxCityMassPlace = 8;

break;

case 3:

textcountry = "Канада";

IndxCityMassPlace = 12;

break;

case 4:

textcountry = "США";

IndxCityMassPlace = 16;

break;

case 5:

textcountry = "Япония";

IndxCityMassPlace = 20;

break;

}

ProvContry = true;

}

}

}

using UnityEngine.UI;

using UnityEngine;

public class EventStopMove : MonoBehaviour

{

public GameObject panelStopMove;

public Text textmess;

[HideInInspector]

public static bool ProvSetPanel = false;

public static int[] massdebtPlayer;

// Start is called before the first frame update

void Start()

{

ProvSetPanel = false;

massdebtPlayer = new int[4];

massdebtPlayer[0] = 50;

massdebtPlayer[1] = 50;

massdebtPlayer[2] = 50;

massdebtPlayer[3] = 50;

}

// Update is called once per frame

void Update()

{

if (ProvSetPanel == true)

{

panelStopMove.SetActive(true);

ProvSetPanel = false;

}

textmess.text = "Вы потеряли " + (massdebtPlayer[Player\_Script.Score]\*2) + ", за простой!";

}

public void But\_DebtPlayer()

{

massdebtPlayer[Player\_Script.Score] \*= 2;

PanelMenu.MoneyBank += massdebtPlayer[Player\_Script.Score];

Player\_Script.Mass\_Player[Player\_Script.Score].Money -= massdebtPlayer[Player\_Script.Score];

panelStopMove.SetActive(false);

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine.SceneManagement;

using UnityEngine.UI;

using UnityEngine;

using System.IO;

public class EndGame\_Script : MonoBehaviour

{

public GameObject panelDebt;

public Text textmess;

public static GameObject panelDeptClose;

//==================================

public GameObject panelInterface;

public GameObject panelEndGame;

public Text TextPointRed;

public Text TextPointBlue;

public Text TextPointGreen;

public Text TextPointYellow;

public Text TextPlaceRed;

public Text TextPlaceBlue;

public Text TextPlaceGreen;

public Text TextPlaceYellow;

[HideInInspector]

public static bool ProvSetPanel = false, ProvPanelEnd = false;

public static int[][] massdebtPlayer;

// Start is called before the first frame update

void Start()

{

ProvPanelEnd = false;

ProvSetPanel = false;

massdebtPlayer = new int[4][];

massdebtPlayer[0] = new int[2] { 0, 3 };

massdebtPlayer[1] = new int[2] { 0, 3 };

massdebtPlayer[2] = new int[2] { 0, 3 };

massdebtPlayer[3] = new int[2] { 0, 3 };

}

// Update is called once per frame

void Update()

{

panelDeptClose = panelDebt;

if (ProvSetPanel == true)

{

panelDebt.SetActive(true);

ProvSetPanel = false;

}

if(massdebtPlayer[Player\_Script.Score][1] < 0)

{

panelInterface.SetActive(false);

panelDebt.SetActive(false);

panelEndGame.SetActive(true);

if (ProvPanelEnd == false) EndGame();

}

textmess.text = "Вы должны банку " + (-massdebtPlayer[Player\_Script.Score][0]) + ", у вас " + massdebtPlayer[Player\_Script.Score][1];

if (massdebtPlayer[Player\_Script.Score][1] == 1) textmess.text += " ход чтобы отдать!";

else if (massdebtPlayer[Player\_Script.Score][1] < 5) textmess.text += " хода чтобы отдать!";

else textmess.text += " ходов чтобы отдать!";

}

public void EndGame()

{

ProvPanelEnd = true;

int i = 4;

var mass = new List<int>();

int redpoint = Player\_Script.Mass\_Player[0].MaxMoney \* ((13 - Player\_Script.Mass\_Player[0].Chips) + 1);

int bluepoint = Player\_Script.Mass\_Player[1].MaxMoney \* ((13 - Player\_Script.Mass\_Player[1].Chips) + 1);

int greenpoint = Player\_Script.Mass\_Player[2].MaxMoney \* ((13 - Player\_Script.Mass\_Player[2].Chips) + 1);

int yellownpoint = Player\_Script.Mass\_Player[3].MaxMoney \* ((13 - Player\_Script.Mass\_Player[3].Chips) + 1);

mass.Add(redpoint);

mass.Add(bluepoint);

mass.Add(greenpoint);

mass.Add(yellownpoint);

mass.Sort();

TextPointRed.text = redpoint.ToString();

TextPointBlue.text = bluepoint.ToString();

TextPointGreen.text = greenpoint.ToString();

TextPointYellow.text = yellownpoint.ToString();

foreach(var item in mass)

{

if (item == redpoint) TextPlaceRed.text = i.ToString();

if (item == bluepoint) TextPlaceBlue.text = i.ToString();

if (item == greenpoint) TextPlaceGreen.text = i.ToString();

if (item == yellownpoint) TextPlaceYellow.text = i.ToString();

i--;

}

}

public static void UpData()

{

massdebtPlayer[Player\_Script.Score][0] += Player\_Script.Mass\_Player[Player\_Script.Score].Money;

PanelMenu.MoneyBank += Player\_Script.Mass\_Player[Player\_Script.Score].Money;

Player\_Script.Mass\_Player[Player\_Script.Score].Money = 0;

}

public void But\_OK()

{

int money = 0;

if(Player\_Script.Mass\_Player[Player\_Script.Score].Money > 0)

{

money = Player\_Script.Mass\_Player[Player\_Script.Score].Money + massdebtPlayer[Player\_Script.Score][0];

if (money >= 0)

{

Player\_Script.Mass\_Player[Player\_Script.Score].CheckDebt = false;

Player\_Script.Mass\_Player[Player\_Script.Score].Money = money;

PanelMenu.MoneyBank -= massdebtPlayer[Player\_Script.Score][0];

massdebtPlayer[Player\_Script.Score][0] = 0;

massdebtPlayer[Player\_Script.Score][1] = 3;

panelDebt.SetActive(false);

}

else

{

PanelMenu.MoneyBank += Player\_Script.Mass\_Player[Player\_Script.Score].Money;

Player\_Script.Mass\_Player[Player\_Script.Score].Money = 0;

massdebtPlayer[Player\_Script.Score][0] = money;

massdebtPlayer[Player\_Script.Score][1]++;

}

}

}

public void But\_Exit()

{

string str;

string[] mass;

if (!Directory.Exists(Application.dataPath + "/Resources/TextFiles"))

Directory.CreateDirectory(Application.dataPath + "/Resources/TextFiles");

if (!File.Exists((Application.dataPath + "/Resources/TextFiles/RecordFile.txt")))

{

//File.Create(Application.dataPath + "/Resources/TextFiles/RecordFile.txt");

//Debug.Log("File Create!");

FileStream fileStream = new FileStream(Application.dataPath + "/Resources/TextFiles/RecordFile.txt", FileMode.Create);

using (var textfile = new StreamWriter(fileStream))

{

textfile.WriteLine("20000 ");

textfile.WriteLine("5000 ");

textfile.WriteLine("5000 ");

textfile.WriteLine("5000 ");

textfile.WriteLine("5000");

Debug.Log("Start File Save!");

}

fileStream.Close();

}

FileStream fileStream2 = new FileStream(Application.dataPath + "/Resources/TextFiles/RecordFile.txt", FileMode.Open);

using (var textfile = new StreamReader(fileStream2))

{

str = textfile.ReadToEnd();

mass = str.Split(' ');

}

fileStream2.Close();

int sumfile = System.Convert.ToInt32(mass[0]);

int sum = System.Convert.ToInt32(TextPointRed.text) + System.Convert.ToInt32(TextPointBlue.text) +

System.Convert.ToInt32(TextPointGreen.text) + System.Convert.ToInt32(TextPointYellow.text);

if (sumfile < sum)

{

mass[0] = sum.ToString() + " ";

mass[1] = TextPointRed.text + " ";

mass[2] = TextPointBlue.text + " ";

mass[3] = TextPointGreen.text + " ";

mass[4] = TextPointYellow.text;

fileStream2 = new FileStream(Application.dataPath + "/Resources/TextFiles/RecordFile.txt", FileMode.Open);

using (var textfile = new StreamWriter(fileStream2))

{

foreach (var s in mass)

{

textfile.WriteLine(s);

}

Debug.Log("File Save!");

}

fileStream2.Close();

}

SceneManager.LoadScene("MenuScene");

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Deal : MonoBehaviour

{

public GameObject deal;

public GameObject bargaining;

[HideInInspector]

public static bool Check;

const string TagPlaceDefault = "Place";

static string TagPlace;

static bool Status;

static int Chips;

public void But\_Ok()

{

deal.SetActive(false);

bargaining.SetActive(true);

//Bargaining.player = player;

}

public static void UpdateDataInDeal()

{

Status = Player\_Script.Mass\_Player[Player\_Script.Score].other.gameObject.GetComponent<Place\_Script>().status;

TagPlace = Player\_Script.Mass\_Player[Player\_Script.Score].other.tag;

Chips = Player\_Script.Mass\_Player[Player\_Script.Score].Chips;

}

public void But\_ShowBargaining()

{

//Status = Player\_Script.Mass\_Player[Player\_Script.Score].other.gameObject.GetComponent<Place\_Script>().status;

//TagPlace = Player\_Script.Mass\_Player[Player\_Script.Score].other.tag;

UpdateDataInDeal();

Debug.Log("Check = " + Check + " TagPlace = " + TagPlace + " Status = " + Status);

if (Check == false && TagPlace == TagPlaceDefault && Status == false && Chips > 0)

{

bargaining.SetActive(true);

//Bargaining.player = player;

Check = true;

RollTheDice\_Scripts.Prov = false;

}

}

public void But\_No()

{

deal.SetActive(false);

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

public class Bargaining : MonoBehaviour

{

public GameObject Panel;

public Text TextNumberRand;

public Text TextSign;

public Text TextNumber;

public GameObject panel;

//[HideInInspector]

//public static Player\_Script player;

[HideInInspector]

public int Number, NumberRan;

private int Quantity;

// Start is called before the first frame update

void Start()

{

Number = 6;

NumberRan = 0;

Quantity = 0;

}

// Update is called once per frame

void Update()

{

TextNumber.text = Number.ToString();

TextNumberRand.text = NumberRan.ToString();

if (NumberRan <= Number)

{

TextSign.text = "<";

panel.GetComponent<Image>().color = Color.red;

}

else

{

TextSign.text = ">";

panel.GetComponent<Image>().color = Color.green;

}

}

public void But\_OK()

{

Panel.SetActive(false);

if (NumberRan > Number)

{

//player.Buy(true);

Player\_Script.Buy(true);

}

else Player\_Script.Buy(false);//player.Buy(false);

Number = 6;

NumberRan = 0;

Quantity = 0;

}

public void But\_RandNumber()

{

if (Quantity < 2)

{

System.Random rnd = new System.Random();

NumberRan += rnd.Next(1, 7);

if (Quantity == 0) NumberRan += Player\_Script.Mass\_Player[Player\_Script.Score].Reputation;

Quantity++;

}

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class PlaceAd : MonoBehaviour

{

public GameObject panelAd;

[HideInInspector]

public static bool ProvSetPanel = false;

// Start is called before the first frame update

void Start()

{

ProvSetPanel = false;

}

// Update is called once per frame

void Update()

{

if (ProvSetPanel == true)

{

panelAd.SetActive(true);

ProvSetPanel = false;

}

}

public void But\_Ok()

{

if (Player\_Script.Mass\_Player[Player\_Script.Score].Money >= 1000)

{

Player\_Script.Mass\_Player[Player\_Script.Score].Money -= 1000;

if (Player\_Script.Mass\_Player[Player\_Script.Score].Reputation < 4) Player\_Script.Mass\_Player[Player\_Script.Score].Reputation += 1;

PanelMenu.MoneyBank += 1000;

panelAd.SetActive(false);

}

}

public void But\_No()

{

panelAd.SetActive(false);

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class PlaceBusSym : MonoBehaviour

{

public GameObject panelBusSym;

[HideInInspector]

public static bool ProvSetPanel = false;

// Start is called before the first frame update

void Start()

{

ProvSetPanel = false;

}

// Update is called once per frame

void Update()

{

if (ProvSetPanel == true)

{

panelBusSym.SetActive(true);

ProvSetPanel = false;

}

}

public void But\_Ok()

{

foreach(Player player in Player\_Script.Mass\_Player)

{

Player\_Script.Mass\_Player[Player\_Script.Score].Money += 200;

player.Money -= 200;

if (player.Reputation < 4) player.Reputation++;

}

panelBusSym.SetActive(false);

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine.UI;

using UnityEngine;

public class PlaceCasino : MonoBehaviour

{

public GameObject panelCasino1;

public GameObject panelCasino2;

public GameObject panelCasino3;

public GameObject panelcolor;

public Text text, textfinale;

[HideInInspector]

public static bool ProvSetPanel = false;

[HideInInspector]

public static int CanNum, MyNum, Sum = 0;

System.Random rnd;

bool ProvRan = false;

// Start is called before the first frame update

void Start()

{

ProvSetPanel = false;

CanNum = 0;

MyNum = 0;

Sum = 0;

rnd = new System.Random();

ProvRan = false;

}

// Update is called once per frame

void Update()

{

if (ProvSetPanel == true)

{

panelCasino1.SetActive(true);

ProvSetPanel = false;

}

else if (panelCasino2.activeInHierarchy == true)

{

text.text = MyNum.ToString() + " - " + CanNum.ToString() + " = " + Sum;

if (Sum > 0) panelcolor.GetComponent<Image>().color = Color.green;

else panelcolor.GetComponent<Image>().color = Color.red;

}

else if (panelCasino3.activeInHierarchy == true)

{

if (Sum > 0) textfinale.text = "Вы выйграли (" + (Sum\*100) + ")!";

if (Sum < 0) textfinale.text = "Вы проиграли (" + (Sum \* 100) + ")!";

if (Sum == 0) textfinale.text = "Вы ничего не выиграли!";

}

}

public void But\_RanNum()

{

if (ProvRan == false)

{

MyNum = rnd.Next(1,7);

Sum = MyNum - CanNum;

ProvRan = true;

}

}

public void But\_Ok()

{

if (panelCasino1.activeInHierarchy == true)

{

panelCasino2.SetActive(true);

panelCasino1.SetActive(false);

CanNum = rnd.Next(1,7);

}

else if (panelCasino2.activeInHierarchy == true)

{

panelCasino2.SetActive(false);

panelCasino3.SetActive(true);

}

else if (panelCasino3.activeInHierarchy == true)

{

Player\_Script.Mass\_Player[Player\_Script.Score].Money += (Sum \* 100);

panelCasino3.SetActive(false);

CanNum = 0;

MyNum = 0;

Sum = 0;

ProvRan = false;

}

}

public void But\_No()

{

panelCasino1.SetActive(false);

ProvRan = false;

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine.UI;

using UnityEngine;

public class PlaceCountry : MonoBehaviour

{

public GameObject panelEvent;

public Text TextMessage;

[HideInInspector]

public static bool ProvSetPanel = false, ProvSetMessage = false;

static int Indx = 0, Indx2 = 0;

string[][] MassMessage;

// Start is called before the first frame update

void Start()

{

ProvSetPanel = false;

ProvSetMessage = false;

Indx = 0;

Indx2 = 0;

MassMessage = new string[5][];

MassMessage[0] = new string[] { "Чиновник вымогал взятку. Пришлось отдать 100 в Банк." };

MassMessage[1] = new string[6]

{

"Правительство России вводит новый налог. Если вам принадлежат клетки в этой стране, заплатите в Банк 100, за каждую.",

"Правительство ЕС вводит новый налог. Если вам принадлежат клетки в этой стране, заплатите в Банк 100, за каждую.",

"Правительство Китая вводит новый налог. Если вам принадлежат клетки в этой стране, заплатите в Банк 100, за каждую.",

"Правительство Канады вводит новый налог. Если вам принадлежат клетки в этой стране, заплатите в Банк 100, за каждую.",

"Правительство США вводит новый налог. Если вам принадлежат клетки в этой стране, заплатите в Банк 100, за каждую.",

"Правительство Японии вводит новый налог. Если вам принадлежат клетки в этой стране, заплатите в Банк 100, за каждую."

};

MassMessage[2] = new string[6]

{

"Экономические реформы России принесли вам прибыль. Если вы имеете в этой стране клетки, получите из банка 500.",

"Экономические реформы ЕС принесли вам прибыль. Если вы имеете в этой стране клетки, получите из банка 500.",

"Экономические реформы Китая принесли вам прибыль. Если вы имеете в этой стране клетки, получите из банка 500.",

"Экономические реформы Канады принесли вам прибыль. Если вы имеете в этой стране клетки, получите из банка 500.",

"Экономические реформы США принесли вам прибыль. Если вы имеете в этой стране клетки, получите из банка 500.",

"Экономические реформы Японии принесли вам прибыль. Если вы имеете в этой стране клетки, получите из банка 500."

};

MassMessage[3] = new string[6]

{

"Государственные инвестиции принесли владельцам построек в России по 300, за каждую клетку.",

"Государственные инвестиции принесли владельцам построек в ЕС по 300, за каждую клетку.",

"Государственные инвестиции принесли владельцам построек в Китае по 300, за каждую клетку.",

"Государственные инвестиции принесли владельцам построек в Канаде по 300, за каждую клетку.",

"Государственные инвестиции принесли владельцам построек в США по 300, за каждую клетку.",

"Государственные инвестиции принесли владельцам построек в Японии по 300, за каждую клетку."

};

MassMessage[4] = new string[6]

{

"Смена курса России привела к расходам. Все игроки чья фишка сейчас находиться на одной из клеток этой страны, платят в банк 500.",

"Смена курса ЕС привела к расходам. Все игроки чья фишка сейчас находиться на одной из клеток этой страны, платят в банк 500.",

"Смена курса Китая привела к расходам. Все игроки чья фишка сейчас находиться на одной из клеток этой страны, платят в банк 500.",

"Смена курса Канады привела к расходам. Все игроки чья фишка сейчас находиться на одной из клеток этой страны, платят в банк 500.",

"Смена курса США привела к расходам. Все игроки чья фишка сейчас находиться на одной из клеток этой страны, платят в банк 500.",

"Смена курса Японии привела к расходам. Все игроки чья фишка сейчас находиться на одной из клеток этой страны, платят в банк 500."

};

}

// Update is called once per frame

void Update()

{

if (ProvSetPanel == true)

{

panelEvent.SetActive(true);

ProvSetPanel = false;

}

if (ProvSetMessage == true)

{

TextMessage.text = MassMessage[Indx][Indx2];

}

else TextMessage.text = "";

}

public void But\_RanMessage()

{

if (ProvSetMessage == false)

{

System.Random rnd = new System.Random();

Indx = rnd.Next(0, 5);

PanelMenu.MoneyFond += 100;

if (Player\_Script.Mass\_Player[Player\_Script.Score].First\_Сaptured == true)

{

Debug.Log("IndxPlace = " + Indx);

Indx = 0;

}

if (Indx == 0) //"Чиновник вымогал взятку. Пришлось отдать 100 в Банк."

{

Player\_Script.Mass\_Player[Player\_Script.Score].Money -= 100;

PanelMenu.MoneyBank += 100;

Indx2 = 0;

}

else if (Indx == 1) //"Правительство России вводит новый налог. Если вам принадлежат клетки в этой стране, заплатите в Банк 100, за каждую."

{

Indx2 = rnd.Next(0,6);

foreach (var place in Player\_Script.Mass\_Player[Player\_Script.Score].MassPlacePlayer)

{

Debug.Log("Place.Indx = " + place.Indx1 + "/ Indx2 = " + (Indx2+1));

if (place.Indx1 == (Indx2+1))

{

Player\_Script.Mass\_Player[Player\_Script.Score].Money -= 100;

PanelMenu.MoneyBank += 100;

}

}

}

else if (Indx == 2) //"Экономические реформы России принесли вам прибыль. Если вы имеете в этой стране клетки, получите из банка 500."

{

Indx2 = rnd.Next(0, 6);

foreach (var place in Player\_Script.Mass\_Player[Player\_Script.Score].MassPlacePlayer)

{

Debug.Log("Place.Indx = " + place.Indx1 + "/ Indx2 = " + (Indx2 + 1));

if (place.Indx1 == (Indx2 + 1))

{

Player\_Script.Mass\_Player[Player\_Script.Score].Money += 500;

PanelMenu.MoneyBank -= 500;

break;

}

}

}

else if (Indx == 3) //"Государственные инвестиции принесли владельцам построек в Японии по 300, за каждую клетку."

{

Indx2 = rnd.Next(0, 6);

foreach (Player player in Player\_Script.Mass\_Player)

{

Debug.Log("Player = " + player.tag + "/ Count = " + player.MassPlacePlayer.Count);

foreach (var place in player.MassPlacePlayer)

{

Debug.Log("Place.Indx = " + place.Indx1 + "/ Indx2 = " + (Indx2 + 1));

if (place.Indx1 == (Indx2 + 1))

{

player.Money += 300;

}

}

}

}

else if (Indx == 4) //"Смена курса Японии привела к расходам. Все игроки чья фишка сейчас находиться на одной из клеток этой страны, платят в банк 500."

{

Indx2 = rnd.Next(0, 6);

foreach (Player player in Player\_Script.Mass\_Player)

{

if (player.other != null)

{

Debug.Log("Player = " + player.tag + "/ Count = " + player.other.gameObject.GetComponent<Place\_Script>().Indx1);

if (player.other.gameObject.GetComponent<Place\_Script>().Indx1 == (Indx2+1))

{

player.Money -= 500;

PanelMenu.MoneyBank += 500;

}

}

}

}

ProvSetMessage = true;

}

}

public void But\_Back()

{

if (ProvSetMessage == true)

{

panelEvent.SetActive(false);

ProvSetMessage = false;

}

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class PlaceFond : MonoBehaviour

{

public GameObject panelFond;

[HideInInspector]

public static bool ProvSetPanel = false;

// Start is called before the first frame update

void Start()

{

ProvSetPanel = false;

}

// Update is called once per frame

void Update()

{

if (ProvSetPanel == true)

{

panelFond.SetActive(true);

ProvSetPanel = false;

}

}

public void But\_Ok()

{

Player\_Script.Mass\_Player[Player\_Script.Score].Money += PanelMenu.MoneyFond;

PanelMenu.MoneyFond = 0;

panelFond.SetActive(false);

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine.UI;

using UnityEngine;

public class PlacePolice : MonoBehaviour

{

public GameObject panelEvent;

public Text TextMess;

[HideInInspector]

public static bool ProvSetPanel = false, ProvSetMessage = false;

string[] MassMessage;

string Mess;

int RanNum;

// Start is called before the first frame update

void Start()

{

ProvSetPanel = false;

ProvSetMessage = false;

MassMessage = new string[4];

MassMessage[0] = "Все документы в порядке. Просто пропустите ход.";

MassMessage[1] = "Вас оштрафовали за нарушение на 200.";

MassMessage[2] = "Вас оштрафовали за нарушение на 500.";

MassMessage[3] = "Вы предложили взятку инспектору.";

}

// Update is called once per frame

void Update()

{

if (ProvSetPanel == true)

{

panelEvent.SetActive(true);

ProvSetPanel = false;

}

if (ProvSetMessage == true)

{

TextMess.text = Mess;

}

else TextMess.text = "";

}

public void But\_RanMessage()

{

if (ProvSetMessage == false)

{

System.Random rnd = new System.Random();

RanNum = rnd.Next(0, 6);

if (RanNum == 0 || RanNum == 1 || RanNum == 2)

{

Mess = MassMessage[0];

ProvSetMessage = true;

Player\_Script.Mass\_Player[Player\_Script.Score].SkipMovePolice = 2;

}

else if (RanNum == 3)

{

Mess = MassMessage[1];

ProvSetMessage = true;

Player\_Script.Mass\_Player[Player\_Script.Score].Money -= 200;

PanelMenu.MoneyBank += 200;

}

else if (RanNum == 4)

{

Mess = MassMessage[2];

ProvSetMessage = true;

Player\_Script.Mass\_Player[Player\_Script.Score].Money -= 500;

PanelMenu.MoneyBank += 500;

}

else if (RanNum == 5)

{

RanNum = rnd.Next(1,7);

if (RanNum < 4)

{

Mess = MassMessage[3] + " И он согласился на 100, можете продолжить путь!";

Player\_Script.Mass\_Player[Player\_Script.Score].Money -= 100;

}

else

{

Mess = MassMessage[3] + " И он отказался. Вы попали за решетку на два хода, и штраф 500.";

ProvSetMessage = true;

Player\_Script.Mass\_Player[Player\_Script.Score].SkipMovePolice = 3;

Player\_Script.Mass\_Player[Player\_Script.Score].Money -= 500;

PanelMenu.MoneyBank += 500;

}

}

}

}

public void But\_Back()

{

if (ProvSetMessage == true)

{

panelEvent.SetActive(false);

ProvSetMessage = false;

Mess = "";

}

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine.UI;

using UnityEngine;

public class PanelSaleBank : MonoBehaviour

{

public GameObject panelEvent;

public Text textMess;

[HideInInspector]

public static bool ProvSetPanel = false, ProvPlace = false, ProvBank = false, ProvPlayer = false;

// Start is called before the first frame update

void Start()

{

ProvSetPanel = false;

ProvPlace = false;

ProvBank = false;

ProvPlayer = false;

}

// Update is called once per frame

void Update()

{

if (ProvSetPanel == true)

{

panelEvent.SetActive(true);

ProvSetPanel = false;

}

textMess.text = PanelSalePlace.Mess;

}

public void But\_OK()

{

Player\_Script.Mass\_Player[Player\_Script.Score].Money += Player\_Script.Mass\_Player[Player\_Script.Score].MassPlacePlayer[PanelSalePlace.IndxPlace].money;

PanelMenu.MoneyBank -= Player\_Script.Mass\_Player[Player\_Script.Score].MassPlacePlayer[PanelSalePlace.IndxPlace].money;

Player\_Script.Mass\_Player[Player\_Script.Score].MassPlacePlayer[PanelSalePlace.IndxPlace].status = false;

Player\_Script.Mass\_Player[Player\_Script.Score].MassPlacePlayer[PanelSalePlace.IndxPlace].TAG = "0";

Debug.Log("IndxPlace = " + PanelSalePlace.IndxPlace);

Player\_Script.Mass\_Player[Player\_Script.Score].MassPlacePlayer.RemoveAt(PanelSalePlace.IndxPlace);

Destroy(Player\_Script.Mass\_Player[Player\_Script.Score].MassPointPlayer[PanelSalePlace.IndxPlace]);

Player\_Script.Mass\_Player[Player\_Script.Score].MassPointPlayer.RemoveAt(PanelSalePlace.IndxPlace);

PanelSalePlace.Mess = "";

panelEvent.SetActive(false);

}

public void But\_Back()

{

panelEvent.SetActive(false);

PanelSalePlace.Mess = "";

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine.UI;

using UnityEngine;

public class PanelSalePlace : MonoBehaviour

{

public GameObject panelEvent;

public Dropdown dropdown1, dropdown2;

public Button But\_ok;

[HideInInspector]

public static bool ProvSetPanel = false, ProvPlace = false, ProvBank = false, ProvPlayer = false;

[HideInInspector]

public static int Indx1 = 1, Indx2 = 1, IndxPlace = -1;

[HideInInspector]

public static string Mess;

[HideInInspector]

public Place\_Script MyPlace;

ColorBlock mycolor;

// Start is called before the first frame update

void Start()

{

ProvSetPanel = false;

ProvPlace = false;

ProvBank = false;

ProvPlayer = false;

Indx1 = 1;

Indx2 = 1;

IndxPlace = -1;

Mess = "";

MyPlace = null;

}

// Update is called once per frame

void Update()

{

if (ProvSetPanel == true)

{

panelEvent.SetActive(true);

ProvSetPanel = false;

}

}

public void But\_OK()

{

if (ProvPlace == true && ProvBank == true)

{

PanelSaleBank.ProvSetPanel = true;

Mess += dropdown2.options[Indx2 - 1].text + " за " + MyPlace.money;

ProvBank = false;

panelEvent.SetActive(false);

ProvPlace = false;

}

else if (ProvPlace == true && ProvPlayer == true)

{

ProvPlayer = false;

panelEvent.SetActive(false);

ProvPlace = false;

}

}

public void ChangeValue()

{

Debug.Log("Drop1 = " + dropdown1.value);

if (dropdown1.value == 0)

{

Indx1 = 1;

Mess = "В России, в городе ";

dropdown2.captionText.text = "Москва";

dropdown2.options[0].text = "Москва";

dropdown2.options[1].text = "Санкт-Петербург";

dropdown2.options[2].text = "Казань";

dropdown2.options[3].text = "Новосибирск";

}

else if (dropdown1.value == 1)

{

Indx1 = 2;

Mess = "В ЕС, в городе ";

dropdown2.captionText.text = "Берлин";

dropdown2.options[0].text = "Берлин";

dropdown2.options[1].text = "Париж";

dropdown2.options[2].text = "Рим";

dropdown2.options[3].text = "Вена";

}

else if (dropdown1.value == 2)

{

Indx1 = 3;

Mess = "В Катае, в городе ";

dropdown2.captionText.text = "Пекин";

dropdown2.options[0].text = "Пекин";

dropdown2.options[1].text = "Шанхай";

dropdown2.options[2].text = "Тяньцзинь";

dropdown2.options[3].text = "Шеньян";

}

else if (dropdown1.value == 3)

{

Indx1 = 4;

Mess = "В Канаде, в городе ";

dropdown2.captionText.text = "Оттава";

dropdown2.options[0].text = "Оттава";

dropdown2.options[1].text = "Ванкувер";

dropdown2.options[2].text = "Монреаль";

dropdown2.options[3].text = "Торонто";

}

else if (dropdown1.value == 4)

{

Indx1 = 5;

Mess = "В США, в городе ";

dropdown2.captionText.text = "Хьюстон";

dropdown2.options[0].text = "Хьюстон";

dropdown2.options[1].text = "Чикаго";

dropdown2.options[2].text = "Лос-Анджелес";

dropdown2.options[3].text = "Нью-Йорк";

}

else if (dropdown1.value == 5)

{

Indx1 = 6;

Mess = "В Японии, в городе ";

dropdown2.captionText.text = "Нагоя";

dropdown2.options[0].text = "Нагоя";

dropdown2.options[1].text = "Осака";

dropdown2.options[2].text = "Йокогама";

dropdown2.options[3].text = "Токио";

}

}

public void ChangeValue2()

{

Debug.Log("Drop2 = " + dropdown2.value);

if (dropdown2.value == 0)

{

Indx2 = 1;

}

else if (dropdown2.value == 1)

{

Indx2 = 2;

}

else if (dropdown2.value == 2)

{

Indx2 = 3;

}

else if (dropdown2.value == 3)

{

Indx2 = 4;

}

if (ProvPlace == false)

{

int i = -1;

foreach (Place\_Script place in Player\_Script.Mass\_Player[Player\_Script.Score].MassPlacePlayer)

{

i++;

Debug.Log("My.Indx1 = " + Indx1 + "/ My.Indx2 = " + Indx2);

Debug.Log("Place.Indx1 = " + place.Indx1 + "/ Place.Indx2 = " + place.Indx2);

if (place.Indx1 == Indx1 && place.Indx2 == Indx2)

{

IndxPlace = i;

ProvPlace = true;

MyPlace = place;

break;

}

}

}

mycolor = But\_ok.colors;

if (ProvPlace == true)

{

mycolor.pressedColor = Color.green;

}

else mycolor.pressedColor = Color.red;

But\_ok.colors = mycolor;

}

public void But\_Back()

{

panelEvent.SetActive(false);

IndxPlace = -1;

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine.SceneManagement;

using UnityEngine.UI;

using UnityEngine;

using System.IO;

public class InterfaceScripts : MonoBehaviour

{

public GameObject panelMessage;

public GameObject panelRecord;

public Text textBestRecord;

public Text textRedRecord;

public Text textBlueRecord;

public Text textGreenRecord;

public Text textYellowRecord;

// Start is called before the first frame update

void Start()

{

}

// Update is called once per frame

void Update()

{

}

public void But\_GameRules()

{

if (!panelMessage.activeSelf)

{

panelMessage.SetActive(true);

}

else panelMessage.SetActive(false);

}

public void But\_ClosePanelMessage()

{

if (!panelMessage.activeSelf)

{

panelMessage.SetActive(true);

}

else panelMessage.SetActive(false);

}

public void But\_Record()

{

if (!panelRecord.activeSelf)

{

panelRecord.SetActive(true);

if (!Directory.Exists(Application.dataPath + "/Resources/TextFiles"))

Directory.CreateDirectory(Application.dataPath + "/Resources/TextFiles");

if (!File.Exists((Application.dataPath + "/Resources/TextFiles/RecordFile.txt")))

{

FileStream fileStream = new FileStream(Application.dataPath + "/Resources/TextFiles/RecordFile.txt", FileMode.Create);

using (var textfile = new StreamWriter(fileStream))

{

textfile.WriteLine("20000 ");

textfile.WriteLine("5000 ");

textfile.WriteLine("5000 ");

textfile.WriteLine("5000 ");

textfile.WriteLine("5000");

textBestRecord.text = "Лучшая игра: 20000 очков!";

textRedRecord.text = "Red: 5000 очков!";

textBlueRecord.text = "Blue: 5000 очков!";

textGreenRecord.text = "Green: 5000 очков!";

textYellowRecord.text = "Yellow: 5000 очков!";

Debug.Log("Start File Save!");

}

fileStream.Close();

}

else

{

string str;

string[] mass;

int num;

FileStream fileStream = new FileStream(Application.dataPath + "/Resources/TextFiles/RecordFile.txt", FileMode.Open);

using (var textfile = new StreamReader(fileStream))

{

str = textfile.ReadToEnd();

mass = str.Split(' ');

}

textBestRecord.text = "Лучшая игра: " + mass[0] + " очков!";

num = int.Parse(mass[1]);

textRedRecord.text = "Red: " + num + " очков!";

num = int.Parse(mass[2]);

textBlueRecord.text = "Blue: " + num + " очков!";

num = int.Parse(mass[3]);

textGreenRecord.text = "Green: " + num + " очков!";

num = int.Parse(mass[4]);

textYellowRecord.text = "Yellow: " + num + " очков!";

}

}

else panelRecord.SetActive(false);

}

public void But\_Start()

{

SceneManager.LoadScene("Scene1");

}

public void But\_Exit()

{

Application.Quit();

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine.SceneManagement;

using UnityEngine;

public class PressSpace : MonoBehaviour

{

// Start is called before the first frame update

void Start()

{

}

// Update is called once per frame

void Update()

{

if (Input.GetKeyDown(KeyCode.Space))

{

SceneManager.LoadScene("MenuScene");

}

}

}