Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра Информатики

Дисциплина: Программирование

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему

Карточная игра «Тысяча»

Выполнил: студент гр. 653501

Гурик Н.В.

Руководитель: ассистент кафедры информатики Козуб В.Н.

Минск 2017

**Оглавление**

Введение [3](#_Toc482476563)

Анализ предметной области4

Постановка задачи4

Теоретическая часть4

Исследование аналогов4

Разработка программы5

Объекты игровой сцены5

Логика игры и скрипты8

Работа программы:15

Заключение18

Список литературы:19

**Введение**

В последнее время актуальность сферы развлечений, и, в частности, сферы игр, довольно сильно растёт. Ещё с давних времён карточные игры были актуальны практически в любой ситуации, из-за компактности карточной колоды и её вместимости(36 – 54 карты), что обеспечивает, в большинстве случаев, удобство и скоротечность игровых партий.

Одной из интереснейших карточных игр является игра «Тысяча». Её интересность заключается в некоторых особенностях:

1. Влияние фактора случайности сведено к минимуму – практически вся колода(за исключением 4 карт) раздаётся игрокам, из-за чего каждый игрок может в начале партии попробовать предугадать, какие карты находятся у противника, и за счёт этого играть не «на шару», а используя своё личное умение и опыт.
2. Полная игра состоит из огромного множества скоротечных партий прогресс между которыми легко сохранить, например, записав его на лист бумаги, либо электронный носитель. Благодаря этому, чтобы выиграть игру, нужно провести множество партий, что отсеивает шансы игрока, который выиграл пару партий на чистом везении.
3. Колода урезается до 24 карт, что обеспечивает скоротечность партий.
4. Множество различных сложных правил, сильно усложняющих игру, которые заставляют игроков много просчитывать и думать.

Так как в последнее время карточная колода стала очень редким гостем в сумках людей, а портативные девайсы – очень частым, то перенос карточных игр в эти самые девайсы является актуальным.

**Анализ предметной области**

*Постановка задачи:*

Целью курсовой работы является создание игры, повторяющей карточную игру «Тысяча». Для этого она должна выполнять следующее:

1. Иметь понятный графический интерфейс для игрока.
2. Иметь возможность сохранять прогресс между сессиями для того, чтобы продолжить игру можно было в любой момент.
3. Полное копирование правил оригинала.

*Теоретическая часть*

 Для создания курсового проекта использовался игровой движок «Unity». Причинами выбора данного игрового движка стали:

1. Одним из языков написания скриптов является C#. Это даёт некоторые преимущества в написании скриптов, так как карточной игре не важна производительность, а наличие стандартных коллекций и других особенностей языка сильно облегчает разработку.
2. Удобство работы с графикой.
3. Наличие стандартных событий для настройки взаимодействия «Игрок - игра»

*Исследование аналогов*

При исследовании аналогов оказалось, что в большинстве случаев карточные игры перенесены в качестве онлайн-игр (браузерных, либо в социальных сетях), но доступ к Интернету есть далеко не всегда, поэтому в случае его отсутствия играть не представляется возможным. При поиске аналогов, не требующих выхода в Интернет, оказалось, что в них предусмотрено сохранение прогресса в течение игры, но при перезаходе в игру прогресс терялся. Эти два недостатка и стали одной из причин создания этого курсового проекта.

**Разработка программы**

** Объекты игровой сцены**

В целом, объекты игровой сцены данной игры можно разделить на 4 группы:

1. Панели
2. Кнопки
3. Камеры
4. Остальные объекты, не относящиеся к этим группам

Рассмотрим каждую группу в отдельности.

*Панели*

Панели в данной игре выполняют роль мест визуализации информации для игроков.

 **GlobalScoresPanel** – панель глобальных результатов игры, а именно очки игроков в течение всей игры. Также указывает, чей ход на данный момент.

**CurrentScoresPanel** – панель результатов конкретной игровой партии.

**StagePanel** – показывает стадию игры, происходящую в данный момент.

**TradePanel** – используется только во время стадии «торговли», содержит информацию о текущей ставке, а так же две кнопки «RaiseButton» и «PassButton», которые будут описаны д.

*Кнопки*

В Unity для кнопок есть стандартные скрипты, сильно упрощающие работу с ними, однако их функционал недостаточен, из-за чего были написаны собственные скрипты, использующие стандартные события движка, в частности использовалось событие OnMouseUp(), которое триггерится при отжатии клавиши мыши на объекте.

**PlayButton** – кнопка, появляющаяся на стартовом экране игры, начинает новую игру либо перезаписывает существующую.

**ContinueButton –** кнопка, появляющаяся на стартовом экране игры, только если существует файл сохранения. Загружает последнюю сохранённую игру.

**ExitButton –** кнопка, появляющаяся на стартовом экране игры, закрывает приложение.

**MainMenuButton –** кнопка, появляющаяся в течение игры, позволяет выйти в главное меню (без сохранения).

**RaiseButton –** часть панели TradePanel, повышает ставку, если текущий игрок может её повысить.

**PassButton –** часть панели TradePanel, позволяет игроку «пасснуть», отдав прикуп и передав управление другому игроку. После первого использования превращается в кнопку, начинающую партию.

**TRANSFERBUTTON –** отладочная кнопка, перед сборкой проекта отключается, передаёт управление другому игроку.

*Камеры*

Камеры используются для более удобного отображения состояний игры. Они содержат элементы интерфейса и фоновую картинку.

**MainCamera –** камера, на которой происходят все основные события игры.

**TransferCamera –** камера, которая нужна для передачи управления другому игроку. Содержит информацию о том, какому игроку передаётся управление.

**WinScreenCamera –** камера, вызываемая при выигрыше одного из игроков, содержит информацию о победителе.

*Остальные объекты*

Объекты данной категории нельзя отнести ни к одной из категорий, но они – одни из самых важных во всём проекте.

**CardPack** – пустой объект, который является «папкой» для хранения 24 объектов игральных карт.

Каждая из карт – отдельный объект, также на каждую из них повешен скрипт CardScript, хранящий масть, ценность карты и методы работы с ними. С технической точки зрения каждую карту можно считать кнопкой, так как взаимодействие с мышью у них определено событием OnMouseUp, как и у кнопок, однако у них есть свои особенности, из-за чего их нельзя считать одним и тем же.

**Players** – пустой объект, хранящий скрипт Hands, который отслеживает и хранит текущее состояние рук игроков, а также прикупа.

**GameRuler** – главный объект игры, хранит в себе основную массу скриптов, обеспечивающих просчет всей логики игры.

**Логика игры и скрипты**

Логика каждой партии в игре разделена на 4 части:

1. Подготовка к партии:

А) Инициализация информации о текущей партии

Б) Перемешивание колоды и раздача её игрокам

1. Стадия торгов за прикуп:

А) Сам процесс торговли игроков

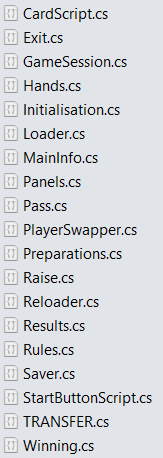
Б) Отдача игроком, который выиграл, излишков карт другому игроку

1. Стадия самой партии:

А) Разыгрывание карт

Б) Подсчёт результатов хода

1. Стадия подсчёта результатов

Логика игры реализуется скриптами, написанными на языке C#. Их примерный синтаксис выглядит так:

public class Script : MonoBehaviour   
{  
    void SomeMethod ()  
    {  
        *//method body*  
    }

...  
}

Для того, чтобы не писать лишнего, далее объявление класса будет опускаться.

Всего скриптов 19 штук, но их можно разделить по группам:

Информационные скрипты – хранят в себе информацию и методы работы с ней. Сюда относятся MainInfo и Hands.

**MainInfo** – святая святых всей игры, скрипт, в котором хранится почти вся иформация о том, что происходит в игре, за редкими исключениями. Отдельного упоминания стоит вложенный класс Flags, хранящий, на какой стадии игры находится программа.

    public static class Flags  
    {  
        public static bool Preparations = false;  
        public static bool Trade = false;  
        public static bool ReserveGiveaway = false;  
        public static bool Game = false;  
        public static bool SessionResults = false;  
        public static bool Results = false;  
    }

**Hands** – скрипт, хранящий текущие руки игроков, используя стандартные списки .NET, а именно List<GameObject>[] CardsHeld . Причиной выбора списков послужила простота и удобство добавления и удаления элементов.

Также он хранит методы ShowCards и HideCards, которые показывают и прячут руку игрока, используя следующий код:

foreach (var card in CardsHeld[i])  
   card.SetActive (true/false);

Кнопочные скрипты – цепляются на кнопки и используют событие OnMouseUp(), а если быть точнее, то при отпускании кнопки мыши на кнопке будут выполняться операции, описанные внутри. К ним относятся Exit, Loader, Pass, Raise, Reloader, StartButtonScript, TRANSFER.

Exit – скрипт, висящий на кнопке ExitButton. Вызывает метод Application.Quit ();

Loader – скрипт, висящий на кнопке ContinueButton. При запуске сцены запускает:

void Start ()   
    {  
        if (!File.Exists (@"1000Save.gsf"))  
            Button.SetActive (false);  
    }

Этот скрипт означает, что если не существует файла сохранения, то кнопка пропадает.

После чего, если файл найден, то считывается информация и начинается новая партия

Pass и Raise - скрипты, висящие на кнопках PassButton и RaiseButton, их работа будет рассмотрена позднее.

Reloader – скрипт, висящий на кнопке MainMenuButton, при нажатии перезапускает игровую сцену методом Application.LoadLevel ("MainScene");

StartButtonScript – скрипт, запускающий новую игру и инициализирующий её с помощью вспомогательного скрипта Initialistion. Также удаляет предыдущее сохранение.

А теперь переходим к скриптам, реализующим игровую логику, путём перечисления этапов игры и скриптов, с ними связанных.

*Этап подготовки*

В конце инициализации активируется флаг MainInfo.Flags.Preparations, переводя игру в стадию подготвки. Этот флаг активирует скрипт Preparations, который с помощью методов Shuffle() и Giveaway() перемешивает и выдаёт в руки игроков по 10 карт из импровизированной «колоды» и ещё 4 помещает в прикуп. После этого, используется вспомогательного скрипта Rules, который содержит:

1. CheckForSet() – проверяет, есть ли в колоде игрока «хваля» данной масти, т.е. дама и король одной масти.
2. CheckForTrump() – проверяет, есть ли в колоде игрока хотя бы одна карта, имеющая масть козыря.
3. CheckForSuit() – проверяет, есть ли в колоде игрока хотя бы одна карта данной масти.

Скрипт Preparations ипользует метод поиска «хвали», чтобы повысить максимальный порог ставки, которую может поставить игрок. После этого активируется флаг MainInfo.Flags.Trade и игра переходит в стадию торговли за прикуп.

*Этап торговли за прикуп*

На этом этапе активируется TradePanel, а вместе с ней RaiseButton и PassButton.

Скрипт Raise активируется при нажатии одноимённой кнопки, повышает ставку для текущего игрока и предаёт управление другому игроку при помощи вспомогательного скрипта PlayerSwapper и его метода Swap(), код которого представлен ниже:

public void Swap (int Direction)  
    {  
        Transfer.SetActive (true);  
        MyCamera.SetActive (false);  
        Hands.HideCards ((Direction + 1) % 2);  
        Hands.ShowCards (Direction);  
        Text.text = "Transfer device to " +

Hands.Names[Direction] + " and press any key...";  
        transfer = true;  
    }

    void Update ()  
    {  
        if (transfer && Input.anyKeyDown)   
        {  
            Transfer.SetActive (false);  
            MyCamera.SetActive (true);  
            transfer = false;  
        }  
    }

Событие Update вызывается каждый кадр, и в данном скрипте просто ожидает ввода любой клавиши для того, чтобы передать управление на главную камеру.

Скрипт Pass тоже активируется при нажатии одноимённой кнопки. При активации скрипта прикуп и управление передаётся второму игроку. С этого момента активируется флаг ReserveGiveaway, где игрок, взявший прикуп, всё ещё может поднять ставку, не передавая управления другому игроку, а также он с этого момента обязан отдать две карты другому игроку.

Взаимодействие игрока с картами на этой фазе и в дальнейшем реализуется тем, что каждая карта представляется как кнопка. На данной фазе нажатие на карту в любом случае отдаст карту другому игроку.

if (MainInfo.Flags.ReserveGiveaway && CardsGiven < 2)   
       {  
          Hands.CardsHeld [(MainInfo.CurrentTurn + 1) % 2].Add (card);  
          Hands.CardsHeld [MainInfo.CurrentTurn].Remove (card);  
          card.transform.position =

new Vector2 ((float)(4f + 1.2f \* (CardsGiven)), -4f);  
          card.SetActive (false);  
          CardsGiven++;  
       }

После передачи двух карт и нажатия кнопки Proceed, которая является переименованной кнопкой Pass, активируется флаг MainInfo.Flags.Game и начинается сама партия.

*Этап партии*

На этом этапе один из игроков ходит, а другой отвечает на его ход. Первый ход всегда делает игрок, взявший прикуп. Игрок, начинающий ход, может сыграть любую карту на его усмотрение, при этом, если он разыгрывает часть «хвали», которая лежит у него в руке, то он получает определённое количество очков, соответствующее масти, также эта масть становится козырем. Эта операция называется «захвалить», и возможна только если это не первая сыгранная карта за партию.

Игрок, отвечающий на ход, может ответить на него, следуя нескольким правилам, которые проверяются по порядку:

1. Если у игрока есть карты той же масти, то он ОБЯЗАН ответить картой той же масти.
2. Иначе, если у игрока есть козырь, то он ОБЯЗАН ответить козырем.
3. Иначе, можно ответить любой картой.

Эти правила и сама игра регулируются вышеупомянутым Rules и скриптом GameSession()

С точки зрения скриптов этот этап происходит так:

1. При наведении курсора мыши на карту вызывается метод GameSession.TryToPlay(), который возвращает true, если карту можно сыграть по всем описанным выше правилам, и в этом случае карта «подпрыгивает», подсказывая игроку, что её можно сыграть
2. При клике по карте, если предыдущий метод вернул true, вызывается метод GameSession.PlayCard(), который кладёт карту на стол, и подсчитывает, «захвалил» ли игрок масть.
3. После этого, если на столе лежит одна карта, то передаётся управление другому игроку, а если два, то происходит подсчет результатов хода.

Подсчёт результатов хода происходит путём вызова метода GameSession.SessionResults(), который делает следующее:

1. Подсчитывает сумму очков карт, лежащих на столе.
2. С помощью метода GameSession.PlayerWon получает номер выигравшего игрока и зачисляет ему очки.
3. Если сыграны не все карты, то передаёт управление игроку, взявшему взятку, иначе переводит игру в состояние подсчёта результатов путём установки флага MainInfo.Flags.Results.

Определение выигравшего игрока определяется следующим способом:

1. Если на столе лежат карты одной масти, то выигрывает взятку игрок, у которого ценность карты выше.
2. Иначе, если на столе есть козырь, то выигрывает взятку игрок, положивший козырь.
3. Иначе взятку выигрывает игрок, положивший карту первым.

Метод PlayerWon описывается следующим образом:

int PlayerWon ()  
    {  
        if (CardsOnBoard [0].GetComponent<CardScript> ().suit ==

CardsOnBoard [1].GetComponent<CardScript> ().suit)   
        {  
            if (CardsOnBoard [0].GetComponent<CardScript> ().value >

CardsOnBoard [1].GetComponent<CardScript> ().value)   
            {  
                return MainInfo.CurrentStartingTurn;  
            }  
            else  
            {  
                return (MainInfo.CurrentStartingTurn + 1) % 2;  
            }  
        }  
        if (CardsOnBoard [1].GetComponent<CardScript> ().suit == MainInfo.Trump)  
        {  
            return (MainInfo.CurrentStartingTurn + 1) % 2;  
        }  
        else  
        {  
            return MainInfo.CurrentStartingTurn;  
        }  
    }

*Этап подсчёта результатов партии*

В конце партии для каждого игрока высчитывается результат по отдельности. Но перед тем, как перейти к описанию алгоритма подсчёта очков – введём понятие «бочки»

Когда игрок набирает 900 или больше очков, количество его очков фиксируется на уровне 900 и считается, что игрок «залез на бочку». Теперь, чтобы выиграть всю игру, игроку на бочке нужно взять прикуп и набрать количество очков, которое он поставил. Если в течение 3-х ходов он этого не делает, либо теряет очки по другой причине, то он «слетает с бочки» и теряет 100 очков. Если игрок «слетает с бочки» 3 раза, то его счет становится равным нулю.

Также введём понятие «болт». Каждый раз, когда игрок получает 0 очков, считается, что он получает «болт». Как только у игрока скапливается 3 «болта», он теряет 100 очков.

Теперь перейдём к подсчёту очков за партию.

Очки каждого игрока округляются до пяти по следующей формуле:

MainInfo.CurrentScores [i] = (int)(Mathf.Round(MainInfo.CurrentScores [i] / 5f)) \* 5;

Для игрока, взявшего прикуп:

1. Если он не набрал количество очков, которое объявил в ставке за прикуп, то он теряет количество очков, равное ставке и «слетает с бочки», если он там находился.
2. Если он набрал нужное количество очков, то он либо выигрывает игру, если он в этот момент был на «бочке», либо получает количество очков, равное его ставке в обратном случае.

Для игрока, не взявшего прикуп:

1. Если он набрал 0 очков, то количество «болтов» увеличивается.
2. Иначе он получает количество очков, которое набрал, только если он не «сидит на бочке»

Все эти расчёты производит скрипт Results. Как только все расчёты проведены, запускается новая партия установкой флага MainInfo.Flags.Preparations, либо, если один из игроков выиграл, переносит на экран, оповещающий о победе одного из игроков, удаляет сохранение и перезапускает сцену.

**Работа программы**

Главное меню при запуске выглядит следующим образом:

При нажатии кнопок новой игры или продолжения игры игра переходит в стадию торговли:

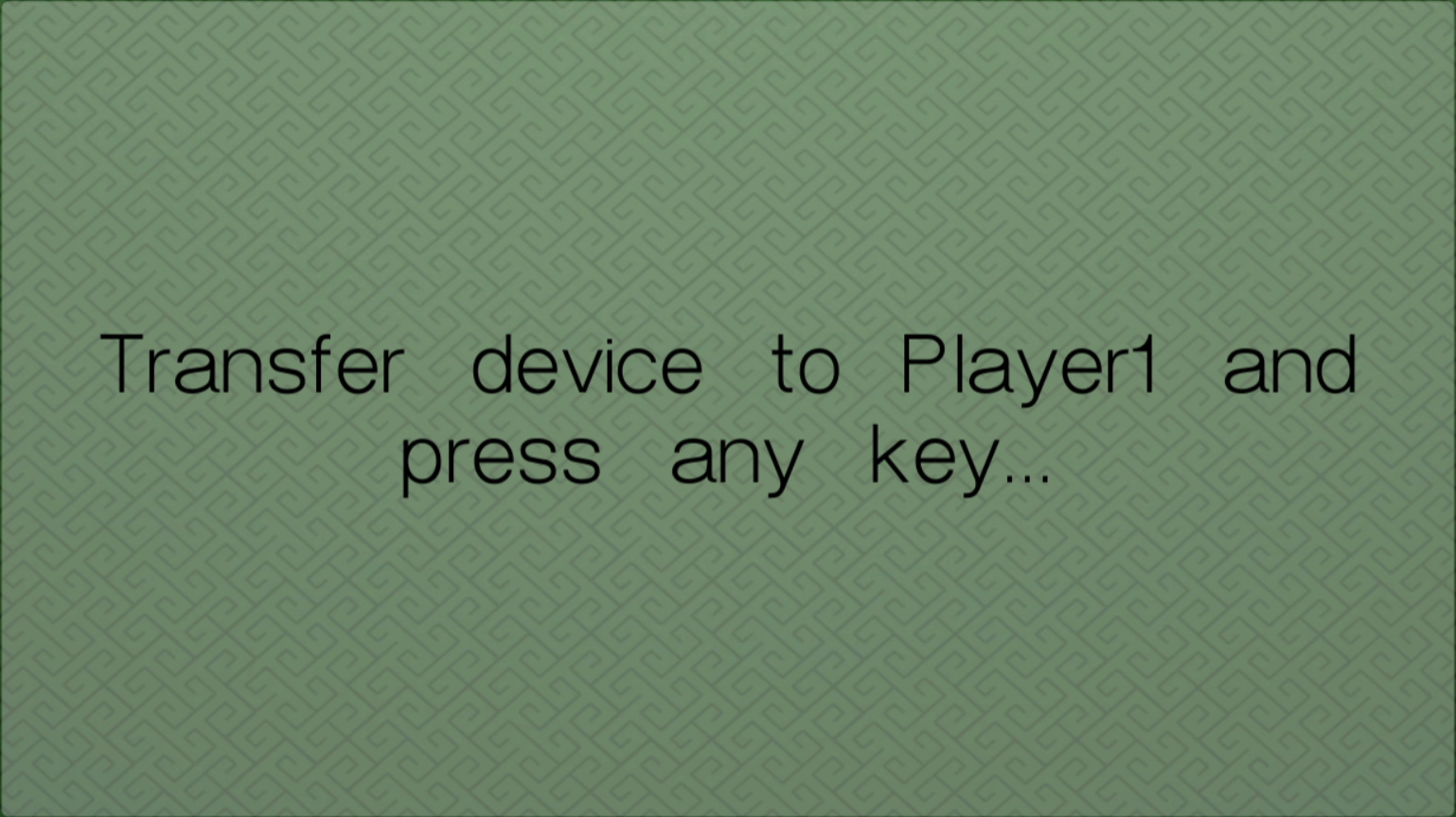


После того, как прикуп отдан, игроки могут начать ходить:





В свою очередь, экран перехода управления выглядит следующим образом:



**Заключение**

При выполнении курсовой работы были выполнены следующие задачи:

- Был освоен игровой движок «Unity», а именно его графическая составляющая и стандартные события.

- Был освоен язык программирования C#, а именно отточен навык использования стандартных коллекций и работы различных классов между собой.

- Была написана карточная игра «Тысяча», в следствие чего повысился навык в написании сложных игр, работающих по множеству не самых простых правил. Также написание проекта было устроено таким образом, что возможна его модификация простой заменой другого игрока на бота. Также организация хранения информации об игроках устроена на коллекциях, а не на конкретных полях, что позволяет модифицировать игру под большее число игроков, без переписывания всего кода.

**Список литературы**

1. Волосевич А.А. Базовые технологии платформы .NET /Пособие – Минск: Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники», 2013.
2. Документация по игровому движку Unity. [Электронный ресурс] – Режим доступа: https://docs.unity3d.com/ru/current/Manual/index.html. Дата доступа: 02.05.2017