Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение  
высшего образования  
«Рязанский государственный радиотехнический университет им. В. Ф. Уткина»  
(ФГБОУ ВО «РГРТУ», РГРТУ)

Кафедра «Вычислительная и прикладная математика» (ВПМ)

**Пояснительная записка**

**к курсовому проекту**

на тему: «Разработка игровой программы «Influence»

по курсу

**«Конструирование программного обеспечения»**

Выполнил:

студент группы № 6413

Лащилин Максим Александрович

Проверил:

доцент кафедры ВПМ

Столчнев В. К.

Рязань, 2019

Оглавление

[Введение 3](#_Toc29954131)

[1. Анализ задачи 4](#_Toc29954132)

[1.1. Разработка иерархии классов 4](#_Toc29954133)

[1.1.1. Выделение сущностей 4](#_Toc29954134)

[1.1.2. Зависимости между классами. Диаграмма классов 4](#_Toc29954135)

[1.2. Алгоритмы 6](#_Toc29954136)

[1.3. Разработка интерфейса программы 7](#_Toc29954137)

[2. Написание программы 10](#_Toc29954138)

[2.1.1. Описание разработанных процедур и функций 10](#_Toc29954139)

[2.2. Разработка программы 16](#_Toc29954140)

[2.2.1. Описание классов, перечислений и интерфейсов проекта 16](#_Toc29954141)

[2.3. Описание шаблонов проектирования, которые использовались при написании программы 19](#_Toc29954142)

[2.3.1. Шаблон «Одиночка» (Singleton) 19](#_Toc29954143)

[2.3.2. Шаблон «Абстрактная фабрика» (Abstract Factory) 19](#_Toc29954144)

[2.3.3. Шаблон «Адаптер» (Adapter) 19](#_Toc29954145)

[2.4. Описание методов рефакторинга, которые использовались при оптимизации исходного кода программы 20](#_Toc29954146)

[2.4.1. Замена магического числа на именованную константу 20](#_Toc29954147)

[2.4.2. Присвоение переменной более ясного или информативного имени 20](#_Toc29954148)

[2.4.3. Присвоение перечислению более ясного или информативного имени 20](#_Toc29954149)

[2.5. Разработка тестов 20](#_Toc29954150)

[2.5.1. Тест-кейсы 20](#_Toc29954151)

[2.5.2. Unit-тесты 23](#_Toc29954152)

[3. Результат работы программы 23](#_Toc29954153)

[3.1. Результаты работы консольной реализации 23](#_Toc29954154)

[3.2. Результаты работы Windows Forms реализации 24](#_Toc29954155)

[Заключение 27](#_Toc29954156)

[Приложения 27](#_Toc29954157)

Введение

На сегодняшний день компьютерные игры, как феномен культуры, активно развиваются и занимают прочную позицию в жизни людей, и не только молодежи. Из всех видов электронных развлечений компьютерные игры представляют особый интерес для науки, так как обладают максимальной информативностью и системной сложностью по отношению к другим электронным играм. Разработкой компьютерных игр может заниматься как один человек, так и большая команда специалистов. Обычно крупномасштабными коммерческими играми занимаются компании, специализирующиеся на играх для персонального компьютера или консолей. Однако разработкой несложной игры вполне может заниматься и один человек.

Рынок компьютерных игр на сегодняшний день, как в мире, так и в России в частности занимает довольно большую долю от всей разработки программного обеспечения. Объясняется это высокой доходностью данных проектов.

В рамках курсового проекта требуется разработать игру «Influence». Для достижения поставленной цели требуется выполнить следующие задачи:

* разработать иерархию классов;
* разработать алгоритмы работы программы;
* разработать макеты пользовательского интерфейса;
* программно реализовать игру;
* произвести тестирование разработанной программы;
* произвести анализ полученных результатов.

# Анализ задачи

## Разработка иерархии классов

Основной задачей данного курсового проекта является разработка игровой программы «Influence» с использованием архитектурного шаблона Model-View-Controller (Модель – Представление – Контроллер). MVC предполагает разделение ответственности классами приложения на три основных блока. Так, каждый из компонентов приложения ответственен за свою задачу:

* модель ответственна за бизнес-логику приложения (обработка данных, изменение состояния контроллером);
* контроллер отвечает за обработку действий пользователя и изменение состояния модели, вызовом её методов;
* представление ответственно за отображение данных модели пользователю.

Использование данного архитектурного паттерна имеет следующие достоинства:

* возможность независимой модификации одного из компонентов относительно других;
* на основе одной модели можно реализовать несколько представлений;
* не затрагивая реализацию представлений можно изменить реакции на действия пользователя путём использования другого контроллера.

Для определённых классов, объекты которых требуются в единственном экземпляре, необходимо применить шаблон «Одиночка». Например, для описания работы курсора.

Для создания контроллеров состояний приложения возможно использование шаблона «Абстрактная фабрика».

### Выделение сущностей

Основными сущностями в игре «Influence» являются следующие:

* Ячейка (Cell) – представляет собой единицу игрового поля, которая может принадлежать одному из игроков или являться свободной. В случае если ячейка принадлежит одному из игроков, она обязательно содержит определённое количество очков, но не больше максимально возможного.
* Игрок (Player) – представляет собой описание сущностей, которые принимают участие в игре.
* Игровое поле (Game field) – представляет собой совокупность ячеек и реализует логику захвата ячеек игроками.

### Зависимости между классами. Диаграмма классов

В данном разделе описаны связи между классами для основных компонентов программы: модели, представления и контроллера.

На рис. 1 представлена диаграмма классов модели.

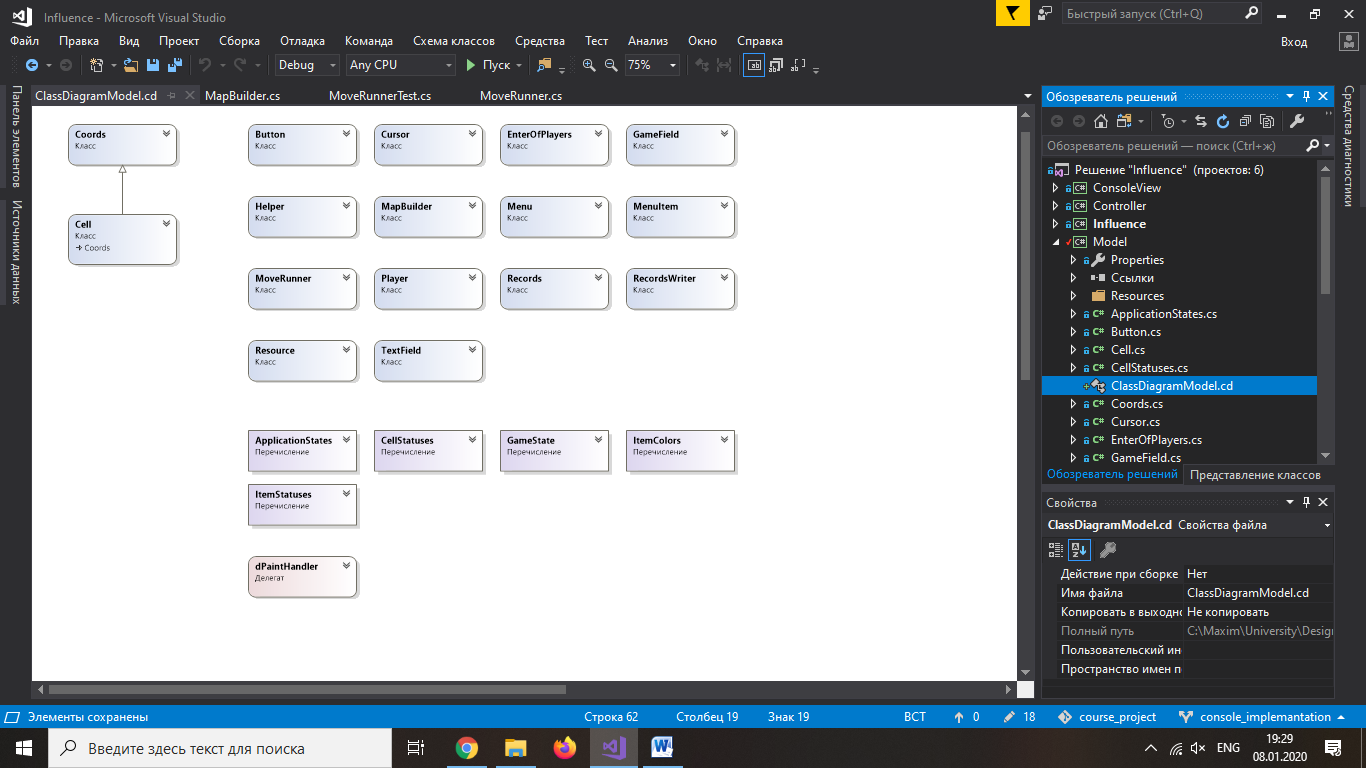


Рис. 1 – Диаграмма классов модели

На рис. 2 представлена диаграмма классов контроллера.

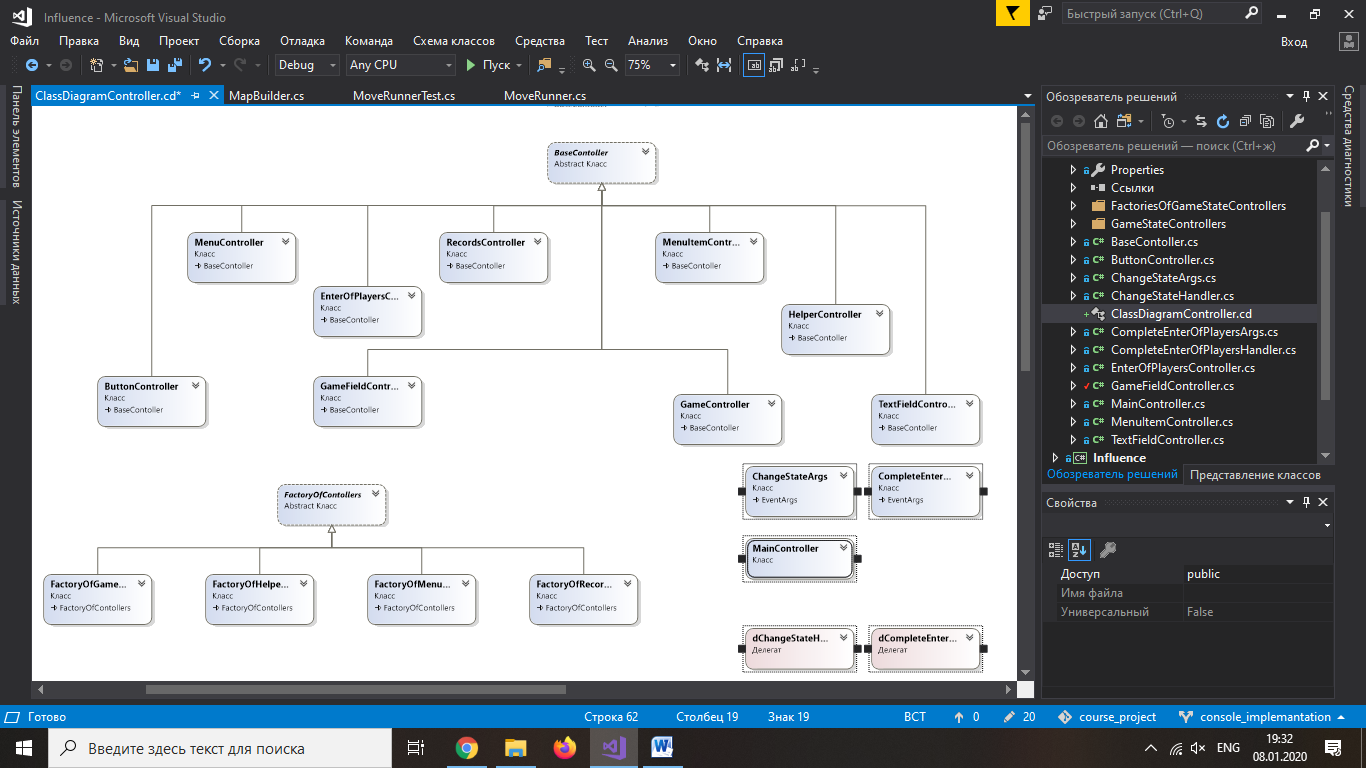


Рис. 2 – Диаграмма классов контроллера

На рис. 3 представлена диаграмма классов представления.

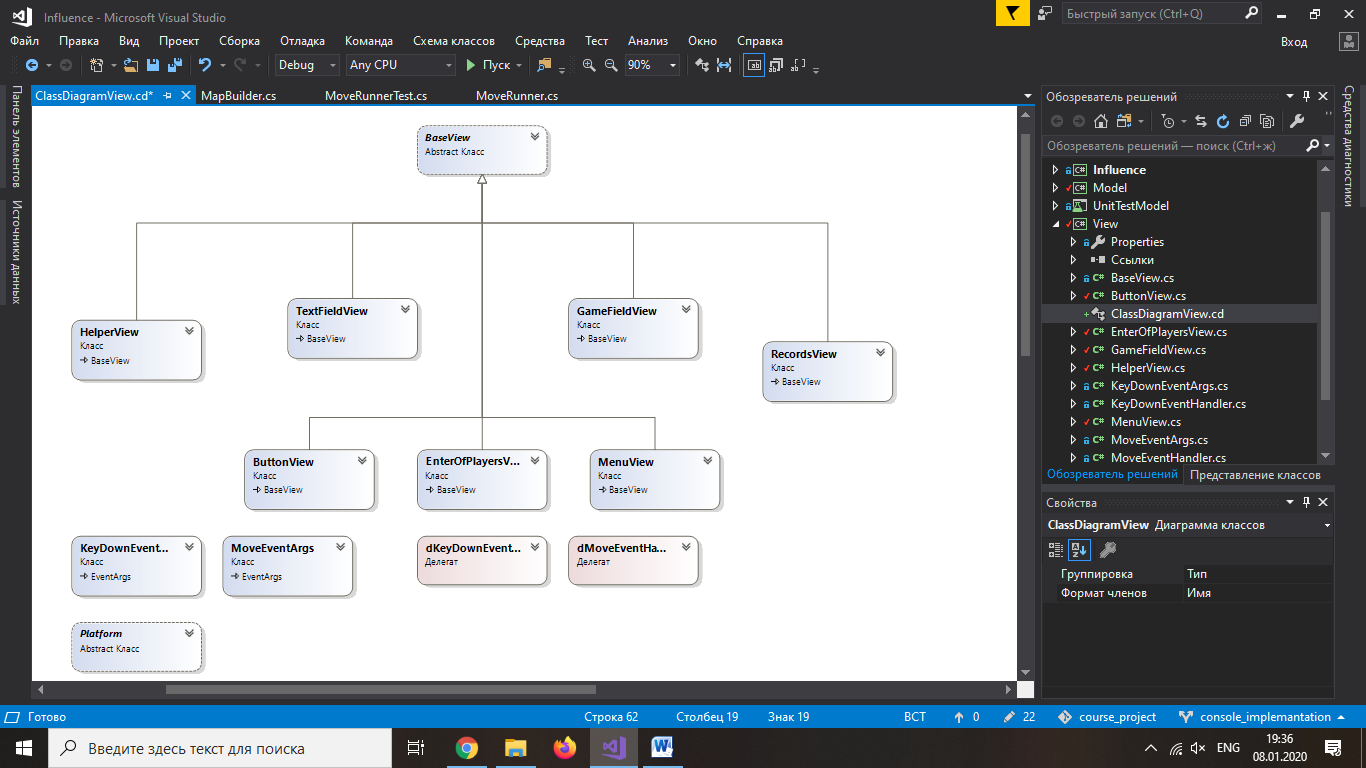


Рис. 3 – Диаграмма классов представления

## Алгоритмы

Одним из важных алгоритмов в игре является алгоритм захвата ячейки игроком. Его блок-схема представлена на рис. 4.



Рис. 4 – Блок-схема алгоритма захвата ячейки

Также важен алгоритм проверки возможности совершения хода из одной ячейки в другую. Его блок-схема представлена на рис. 5.

Рис. 5 – Блок-схема алгоритма проверки возможности совершения хода

## Разработка интерфейса программы

На этапе проектирования программы требуется разработать её графический пользовательский интерфейс. Интерфейс должен включать в себя следующие элементы:

* главное меню игры;
* ввод имён игроков;
* игровое поле;
* просмотр рекордов;
* справку.

Макеты перечисленных элементов представлены на рисунках ниже.



Рис. 6 – Главное меню



Рис. 7 – Ввод имени игроков



Рис. 8 – Игровое поле в состоянии захвата ячейки



Рис. 9 – Игровое поле в состоянии раздачи очков



Рис. 10 – Просмотр рекордов



Рис. 11 – Справка

# Написание программы

### Описание разработанных процедур и функций

Описание основных разработанных процедур и функций представлено в таблице 1.

Таблица 1 – Описание разработанных процедур и функций

| **Название процедуры/функции** | **Описание функциональности** | **Описание параметров** |
| --- | --- | --- |
| Button. CallClick() | Вызывает событие нажатия на кнопку | - |
| Button. CallPaintEvent() | Вызывает событие перерисовки | - |
| Cell.IncreaseScore() | Увеличивает счёт на единицу | - |
| Cell. DecreaseScore() | Уменьшает счёт на единицу | - |
| Cell.ActiveCell() | Делает ячейку активной | - |
| Cell.DisactiveCell() | Делает ячейку неактивной | - |
| Cell.IsCellFree() | Проверяет, имеет ли ячейка владельца | - |
| Cursor.GetInstance() | Получает экземпляр курсора | - |
| Cursor.Move(float parX, float parY) | Изменяет координаты курсора | parX – Координата X  parY - Координата Y |
| EnterOfPlayers.Initialize() | Инициализирует ввод имен игроков | - |
| EnterOfPlayers.Next() | Переводит фокус на следующее текстовое поле | - |
| EnterOfPlayers.Previous() | Переводит фокус на предыдущее текстовое поле | - |
| GameField.OnFinishedEvent(object parSender, EventArgs parE) | Обрабатывает событие окончания игры | parSender – Источник события  parE - Параметры события |
| GameField.Initialize() | Инициализирует игровое поле | - |
| GameField.SelectCell() | Выбирает ячейку | - |
| GameField.AtackCell() | Производит атаку на ячейку | - |
| GameField.DistributeScore() | Раздает очки | - |
| GameField.UnselectAllCells() | Снимает выделение со всех ячеек | - |
| GameField.PassMove() | Передает ход следующему игроку | - |
| GameField.GetFocusedCell() | Получает ячейку, на которую наведен курсор | - |
| GameField.GetSelectedCell() | Получает выбранную ячейку | - |
| GameField.GetPlayerNumOfCells(Player parPlayer) | Получает число ячеек игрока | parPlayer - Объект игрока |
| GameField.CompleteAtackOrPassMove() | Завершает атаку или передает ход | - |
| GameField.PerformGameAction() | Выполняет одно из игровых действий | - |
| GameField.CalculateScorePlayer() | Вычисляет счет текущего игрока | - |
| GameField.GetActivePlayer() | Возвращает текущего игрока | - |
| GameField.IsFinishedGame() | Проверяет, окончена ли игра | - |
| MapBuilder.BuildMap(int parVerticalSize, int parHorizontalSize, List<Player> parPlayers) | Строит карту | parVerticalSize – Вертикальный размер  parHorizontalSize – Горизонтальный размер  parPlayers – Список игроков |
| MapBuilder.CreateGameCells(int parVerticalSize, int parHorizontalSize) | Создает массив игровых ячеек | parVerticalSize – Вертикальный размер  parHorizontalSize – Горизонтальный размер |
| MapBuilder.SetPlayers(Cell[,] parCells, List<Player> parPlayers) | Устанавливает игроков для первоначальных ячеек | parCells – Массив ячеек  parPlayers – Список игроков |
| Menu.Initialize() | Инициализирует меню | - |
| Menu.AddItem(int parId, string parName) | Добавляет элемент в меню | parId – Идентификатор  parName – Название пункта |
| Menu.Next() | Переводит фокус на следующиий элемент меню | - |
| Menu.Previous() | Переводит фокус на предыдущий элемент меню | - |
| MoveRunner.Move(Cell parSourceCell, Cell parDestinationCell, Player parCurrentPlayer) | Совершает ход | parSourceCell – Исходная ячейка  parDestinationCell – Ячейка назначения  parCurrentPlayer - Текущий игрок |
| MoveRunner.IsMove(int parSourceVerticalCoord, int parSourceHorizontalCoord, int parDestinationVerticalCoord, int parDestinationHorizontalCoord) | Проверяет возможность совершения хода | parSourceVerticalCoord – Индекс ряда исходной ячейки  parSourceHorizontalCoord – Индекс колонки исходной ячейки  parDestinationVerticalCoord – Индекс ряда ячейки назначения  parDestinationHorizontalCoord – Индекс колонки ячейки назначения |
| MoveRunner.IsCellOccupied(Cell parCell, Player parCurrentPlayer) | Проверяет, занята ли ячейка другим игроком | parCell – Объект ячейки  parCurrentPlayer – Текущий игрок |
| Records.Initialize() | Инициализирует просмотр рекордов | - |
| Records.ReadTextFromFile(string parFileName) | Читает текст из файла | parFileName - Имя файла |
| Records.FilterBestResults(List<string> parRecords) | Фильтрует результаты | parRecords - Список результатов |
| RecordsWriter.RecordResult(Player parPlayer) | Записывает результат игрока | parPlayer - Объект игрока |
| TextField.Initialize() | Инициализирует текстовое поле | - |
| TextField.AddChar(char parChar) | Добавляет символ в текстовое поле | parChar - Добавляемый символ |
| TextField.DeleteLastChar() | Удаляет последний символ из текстового поля | - |
| FactoryOfContollers.CreateController(Platform parPlatform) | Создает контроллер состояния приложения | parPlatform - Платформа |
| FactoryOfGameControllers.CreateController(Platform parPlatform) | Создает контроллер состояния игры | parPlatform - Платформа |
| FactoryOfHelperControllers.CreateController(Platform parPlatform) | Создает контроллер просмотра справки | parPlatform - Платформа |
| FactoryOfMenuControllers.CreateController(Platform parPlatform) | Создает контроллер меню | parPlatform - Платформа |
| FactoryOfRecordsController.CreateController(Platform parPlatform) | Создает контроллер рекордов | parPlatform - Платформа |
| GameController.OnCompleteEnterOfPlayers(object parSender, CompleteEnterOfPlayersArgs parE) | Обрабатывает событие завершения ввода игроков | parSender – Источник события  parE – Параметры события |
| GameController.OnChangeState(object parSender, ChangeStateArgs parE) | Обрабатывает событие изменения состояния приложения | parSender – Источник события  parE – Параметры события |
| HelperController.OnEscDown(object parSender, EventArgs parE) | Обрабатывает нажатие Esc | parSender – Источник события  parE – Параметры события |
| MenuController.InitMenuItemsControllers(Platform parPlatform, List<KeyValuePair<int, MenuItem>> parMenuItems) | Инициализирует контроллеры элементов меню | parPlatform – Платформа  parMenuItems - Элементы меню |
| MenuController.SubscribeOnChangeState() | Подписывает на событие изменения состояния | - |
| MenuController.OnArrowUp(object parSender, EventArgs parEventArgs) | Обрабатывает нажатие стрелки вверх | parSender – Источник события  parEventArgs – Параметры события |
| MenuController.OnArrowDown(object parSender, EventArgs parEventArgs) | Обрабатывает нажатие стрелки вниз | parSender – Источник события  parEventArgs – Параметры события |
| MenuController.OnChangeState(object parSender, ChangeStateArgs parE) | Обрабатывает изменение состояния приложения | parSender – Источник события  parE – Параметры события |
| RecordsController.OnEscDown(object parSender, EventArgs parE) | Обрабатывает нажатие Esc | parSender – Источник события  parE – Параметры события |
| BaseContoller.CallChangeState(object parSender, ChangeStateArgs parE) | Вызывает событие изменения состояния приложения | parSender – Источник события  parE – Параметры события |
| ButtonController.Initialize() | Инициализирует контроллер кнопки | - |
| ButtonController.OnClick(object parSender, EventArgs parE) | Обрабатывает нажатие на кнопку | parSender – Источник события  parE – Параметры события |
| EnterOfPlayersController.OnEscDown(object parSender, EventArgs parE) | Обрабатывает нажатие Esc | parSender – Источник события  parE – Параметры события |
| EnterOfPlayersController. InitizlizeTextFieldControllers(Platform parPlatform, List<TextField> parTextFields) | Инициализирует контроллеры текстовых полей | parPlatform – Платформа  parTextFields – Текстовые поля |
| EnterOfPlayersController.GetTextFieldViews() | Получает представления текстовых полей | - |
| EnterOfPlayersController.OnEnterDown(object parSender, EventArgs parE) | Обрабатывает нажатие кнопки Enter | parSender – Источник события  parE – Параметры события |
| EnterOfPlayersController.OnArrowDown(object parSender, EventArgs parE) | Обрабатывает нажатие стрелки вниз | parSender – Источник события  parE – Параметры события |
| EnterOfPlayersController.OnArrowUp(object parSender, EventArgs parE) | Обрабатывает нажатие стрелки вверх | parSender – Источник события  parE – Параметры события |
| GameFieldController.OnEscDown(object parSender, EventArgs parE) | Обрабатывает нажатие Esc | parSender – Источник события  parE – Параметры события |
| GameFieldController.OnButtonClick(object parSender, EventArgs parE) | Обрабатывает нажатие на кнопку | parSender – Источник события  parE – Параметры события |
| GameFieldController.OnClick(object parSender, EventArgs parE) | Обрабатывает событие клика по поверхности платформы | parSender – Источник события  parE – Параметры события |
| GameFieldController.OnMove(object parSender, MoveEventArgs parE) | Обрабатывает событие перемещения мыши | parSender – Источник события  parE – Параметры события |
| MainController.Start() | Запускает главный контроллер | - |
| MainController.ChangeState(ApplicationStates parState, FactoryOfContollers parFactoryOfContollers) | Изменяет состояние приложения | parState – Состояние приложения  parFactoryOfContollers – Фабрика контроллера |
| MainController.OnChangeState(object parSender, ChangeStateArgs parE) | Обрабатывает событие изменения состояния приложения | parSender – Источник события  parE – Параметры события |
| MenuItemController.OnEnter(object parSender, EventArgs parE) | Обрабатывает событие нажатия клавиши Enter | parSender – Источник события  parE – Параметры события |
| TextFieldController.OnKeyDown(object parSender, KeyDownEventArgs parE) | Обрабатывает событие нажатия клавиши | parSender – Источник события  parE – Параметры события |
| TextFieldController.OnBackspaceDown(object parSender, EventArgs parE) | Обрабатывает нажатие Backspace | parSender – Источник события  parE – Параметры события |
| BaseView.Draw() | Отрисовывает представление | - |
| ButtonView.Draw() | Отрисовывает кнопку | - |
| EnterOfPlayersView.SubcribeOnTextFieldEvents() | Подписывает метод Draw на события перерисовки модели | - |
| EnterOfPlayersView. Draw() | Отрисовывает ввод имен игроков | - |
| GameFieldView.SubscribeOnButtonViewEvent() | Подписывает метод Draw на событие перерисовки кнопки | - |
| GameFieldView.Draw() | Отрисовывает игровое поле | - |
| HelperView.Draw() | Отрисовывает просмотр справки | - |
| MenuView.Draw() | Отрисовывает меню | - |
| Platform.CallArrowUpDown() | Вызывает событие нажатия клавиши со стрелкой вверх | - |
| Platform.CallArrowDown() | Вызывает событие нажатия клавиши со стрелкой вниз | - |
| Platform.CallEnterDown() | Вызывает событие нажатия клавиши Enter | - |
| Platform.CallEscDown() | Вызывает событиие нажатия клавиши Esc | - |
| Platform.CallBackspaceDown() | Вызывает событиие нажатия Backspace | - |
| Platform.CallKeyDown(KeyDownEventArgs parE) | Вызывает событие нажатия клавиши | parE - Параметры события |
| Platform.CallMove(MoveEventArgs parE) | Вызывает событие перемещения курсора мыши | parE - Параметры события |
| Platform.CallClick() | Вызывает событие клика мышью | - |
| Platform.CallReadyFrame() | Вызывает событие готовности кадра | - |
| Platform.UnsubscribeAllEvents() | Удаляет обработчики всех событий | - |
| Platform.Initialize() | Инициализирует платформу | - |
| Platform.Drop() | Уничтожает платформу | - |
| Platform.Clear() | Очищает область отрисовки | - |
| Platform.PrintText(float parX, float parY, string parText) | Печатает текст | parX – Координата X  parY – Координата Y  parText – Текст |
| Platform.DrawHexagonWithScore(float parX, float parY, int parScore, ItemColors parColor, bool parDecorativity) | Отрисовывает игровую ячейку | parX – Координата X  parY – Координата Y  parScore – Число очков ячейки  parColor – Цвет ячейки  parDecorativity - Выделение ячейки |
| RecordsView.Draw() | Отрисовывает представление рекордов | - |
| TextFieldView.Draw() | Отрисовывает текстовое поле | - |

## Разработка программы

В разработке программы можно выделить следующие этапы:

1. Разработка общей структуры проекта. В процессе этого этапа были выделены основные архитектурные слои (модель, представление, контроллер), разработана общая модель взаимодействия классов при помощи событий.
2. Разработка бизнес-логики игры.
3. Разработка конкретных реализаций для представлений (консольного и Windows Forms).
4. Тестирование.
5. Оптимизация алгоритмов и рефакторинг программного кода.

### Описание классов, перечислений и интерфейсов проекта

Описание классов, перечислений и интерфейсов проекта представлено в таблице 2.

Таблица 2 – Описание классов, перечислений и интерфейсов проекта

| **Пространство имен** | **Название класса / интерфейса / перечисления** | **Назначение** |
| --- | --- | --- |
| Model | ApplicationStates | Состояния приложения |
| Model | Button | Кнопка |
| Model | Cell | Игровая ячейка |
| Model | CellStatuses | Статусы ячейки |
| Model | Coords | Координаты ячейки в массиве |
| Model | Cursor | Курсор |
| Model | EnterOfPlayers | Отвечает за ввод игроков |
| Model | GameField | Игровое поле |
| Model | GameStates | Состояния игры |
| Model | Helper | Справка |
| Model | ItemColors | Цвета, отличающие игроков |
| Model | ItemStatuses | Статусы элементов управления |
| Model | MapBuilder | Строитель карты |
| Model | Menu | Меню |
| Model | MenuItem | Элемент меню |
| Model | MoveRunner | Отвечает за логику совершения ходов |
| Model | Player | Класс игрока |
| Model | Records | Отвечает за просмотр рекордов |
| Model | RecordsWriter | Отвечает за запись результата игры в файл |
| Model | TextField | Текстовое поле |
| Controller.FactoriesOfGameStateControllers | FactoryOfContollers | Абстрактная фабрика контроллеров |
| Controller.FactoriesOfGameStateControllers | FactoryOfGameControllers | Фабрика контроллера игры |
| Controller.FactoriesOfGameStateControllers | FactoryOfHelperControllers | Фабрика контроллера просмотра справки |
| Controller.FactoriesOfGameStateControllers | FactoryOfMenuControllers | Фабрика контроллера меню |
| Controller.FactoriesOfGameStateControllers | FactoryOfRecordsController | Фабрика контроллера рекордов |
| Controller | GameController | Контроллер игры |
| Controller | HelperController | Контроллер справки |
| Controller | MenuController | Контроллер меню |
| Controller | RecordsController | Контроллер рекордов |
| Controller | BaseContoller | Базовый контроллер |
| Controller | ButtonController | Контроллер кнопки |
| Controller | ChangeStateArgs | Параметры события изменения состояния приложения |
| Controller | CompleteEnterOfPlayersArgs | Параметры события завершения ввода игроков |
| Controller | EnterOfPlayersController | Контроллер ввода игроков |
| Controller | GameFieldController | Контроллер игрового поля |
| Controller | MainController | Главный контроллер |
| Controller | MenuItemController | Контроллер элемента меню |
| Controller | TextFieldController | Контроллер текстового поля |
| View | BaseView | Базовое представление |
| View | ButtonView | Представление кнопки |
| View | EnterOfPlayersView | Представление ввода игроков |
| View | GameFieldView | Представление игрового поля |
| View | HelperView | Представление просмотра справки |
| View | KeyDownEventArgs | Параметры события нажатия клавиши |
| View | MenuView | Представление меню |
| View | MoveEventArgs | Параметры события перемещения курсора мыши |
| View | Platform | Абстрактная графическая платформа |
| View | RecordsView | Представление рекордов |
| View | TextFieldView | Представление текстового поля |
| WinFormsView | WinFormsPlatform | Платформа Windows Forms |
| WinFormsView | WinFormsDrawer | Отвечает за перерисовку формы |
| ConsoleView | ConsoleDrawer | Отвечает за двойную буферизацию |
| ConsoleView | ConsolePlatform | Консольная платформа |
| ConsoleView | EventListener | Слушатель событий Windows |
| ConsoleView | NativeMethodsProvider | Провайдер нативных методов для работы с WinAPI |
| ConsoleView | ConsoleKeyboardEventArgs | Параметры события нажатия клавиши в консоле |

## Описание шаблонов проектирования, которые использовались при написании программы

### Шаблон «Одиночка» (Singleton)

Одиночка – это порождающий шаблон проектирования, который гарантирует, что у класса есть только один экземпляр, и предоставляет к нему глобальную точку доступа.

Данный шаблон используется в классах «Cursor» и «EventListener».

### Шаблон «Абстрактная фабрика» (Abstract Factory)

Абстрактная фабрика – это порождающий шаблон проектирования, который позволяет создавать семейства связанных объектов, не привязываясь к конкретным классам создаваемых объектов.

Данный шаблон применяется для создания контроллеров, которые отвечают за обработку действий от пользователя в следующих состояниях: работе в главном меню, игровом процессе, просмотре справочной информации и рекордов.

### Шаблон «Адаптер» (Adapter)

Адаптер – это структурный шаблон проектирования, предназначенный для организации использования функций объекта, недоступного для модификации, через специально созданный интерфейс.

Данный шаблон применяется в классе EventListener, который использует функциональность класса NativeMethodsProvider.

## Описание методов рефакторинга, которые использовались при оптимизации исходного кода программы

### Замена магического числа на именованную константу

Данный метод подразумевает замену численного или строкового литерала именованной константой.

Пример кода до рефакторинга:

switch (parE.VirtualKeyCode)

{

case 13:

CallEnterDown();

break;

case 38:

CallArrowUpDown();

break;

case 40:

CallArrowDown();

break;

}

Пример кода после рефакторинга:

const int ENTER\_CODE = 13;

const int ARROW\_UP\_CODE = 38;

const int ARROW\_DOWN\_CODE = 40;

if (parE.KeyDown)

{

switch (parE.VirtualKeyCode)

{

case ENTER\_CODE:

CallEnterDown();

break;

case ARROW\_UP\_CODE:

CallArrowUpDown();

break;

case ARROW\_DOWN\_CODE:

CallArrowDown();

break;

}

### Присвоение переменной более ясного или информативного имени

Присвоение переменной более ясного или информативного имени позволяет улучшить читаемость и/или информативность кода.

Пример кода до рефакторинга:

private int \_virtualKeyCode;

Пример кода после рефакторинга:

private int \_keyCode;

### Присвоение перечислению более ясного или информативного имени

Присвоение переменной более ясного или информативного имени позволяет улучшить читаемость и/или информативность кода.

Пример кода до рефакторинга:

public enum ItemStatus

Пример кода после рефакторинга:

public enum ItemStatuses

## Разработка тестов

### Тест-кейсы

Для проверки работоспособности игры были разработаны тест-кейсы, представленные в таблице 3.

Таблица 3 – Тест-кейсы

| **Название теста** | **Предусловия** | **Действия** | **Ожидаемый результат** | **Результат выполнения** |
| --- | --- | --- | --- | --- |
| Запуск игры | - | 1. Запустить игру | Отображается главное меню игры | Выполнен |
| 1. В главном меню выбрать пункт «New game» | Подсвечен пункт «New game» | Выполнен |
| 1. В появившемся окне ввода игроков ввести имена игроков | Представлены два текстовых поля для ввода имен игроков. Переключение между полями возможно при помощи клавиш со стрелками или клавиши Tab | Выполнен |
| 1. Нажать Enter | Появляется игровое поле с ячейками, кнопкой «Complete atack» и подписью текущего игрока. | Выполнен |
| Занятие свободной ячейки | Игра запущена | 1. Выбрать ячейку текущего игрока, число очков которой больше единицы | Выбранная ячейка подсвечивается. | Выполнен |
| 1. Выбрать соседнюю свободную ячейку | Счет исходной ячейки стал равен единице. Счет выбранной соседней ячейки стал равен предыдущему значению исходной ячейки минус один. Свободная ячейка окрашена в цвет ячеек текущего игрока. | Выполнен |
| Занятие ячейки соперника, счёт которой больше счета атакующей ячейки | Игра запущена, ячейки игроков соседствуют | 1. Выбрать ячейку текущего игрока, число очков которой больше единицы | Выбранная ячейка подсвечивается. | Выполнен |
| 1. Выбрать соседнюю ячейку, занятую другим игроком, счет которой больше счета атакующей ячейки | Счет соседней ячейки стал равен разнице между счетом атакуемой и атакующей ячейки. Счет атакующей ячейки стал равен единице. Цвета обоих ячеек остались прежними. | Выполнен |
| Занятие ячейки соперника, счёт которой равен счету атакующей ячейки | Игра запущена, ячейки игроков соседствуют | 1. Выбрать ячейку текущего игрока, число очков которой больше единицы | Выбранная ячейка подсвечивается. | Выполнен |
| 1. Выбрать соседнюю ячейку, занятую другим игроком, счет которой равен счету атакующей ячейки | Счет атакуемой ячейки стал равен единице. Счет атакующей ячейки стал равен единице. Цвета обоих ячеек остались прежними. | Выполнен |
| Занятие ячейки соперника, счёт которой меньше счета атакующей ячейки | Игра запущена, ячейки игроков соседствуют | 1. Выбрать ячейку текущего игрока, число очков которой больше единицы | Выбранная ячейка подсвечивается. | Выполнен |
| 1. Выбрать соседнюю ячейку, занятую другим игроком, , счет которой меньше счета атакующей ячейки | Счет атакуемой ячейки стал равен разнице между счетом атакующей ячейки и счетом атакуемой ячейки. Счет атакующей ячейки стал равен единице. Атакующая и атакуемая ячейки приняли цвет текущего игрока. | Выполнен |
| Завершение атаки | Игра запущена | 1. Нажать кнопку «Complete atack» | Надпись на кнопке изменилась на «Pass move». | Выполнен |
| Раздача очков | Игра запущена, завершена атака | 1. Выбрать ячейку текущего игрока | Счет выбранной ячейки увеличился на единицу (при условии, что выбранная ячейка не имела максимальное значение счета, равное восьми). | Выполнен |
| Передача хода | Игра запущена, завершена атака | 1. Нажать кнопку «Pass move» | Надпись на кнопке изменилась на «Complete atack». В строке внизу экрана имя текущего игрока изменилось. | Выполнен |
| Окончание игры | Игра запущена, соперник имеет только одну ячейку, соседствующую с ячейками текущего игрока |  |  |  |
|  |  |  |
|  |  |  |
| Сохранение рекорда |  |  |  |  |
| Выход в главное меню |  |  |  |  |

### Unit-тесты

Для тестирования класса MoveRunner, отвечающего за логику совершения ходов, были созданы модульные тесты в классе MoveRunnerTest.

В классе MoveRunner были протестированы следующие методы:

* Move;
* IsMove;
* IsCellOccupied.

# Результат работы программы

## Результаты работы консольной реализации

На следующих рисунках представлены результаты работы программы, реализованной в консольном режиме.

На рис. отображается главное меню игры, появляющееся при запуске программы.

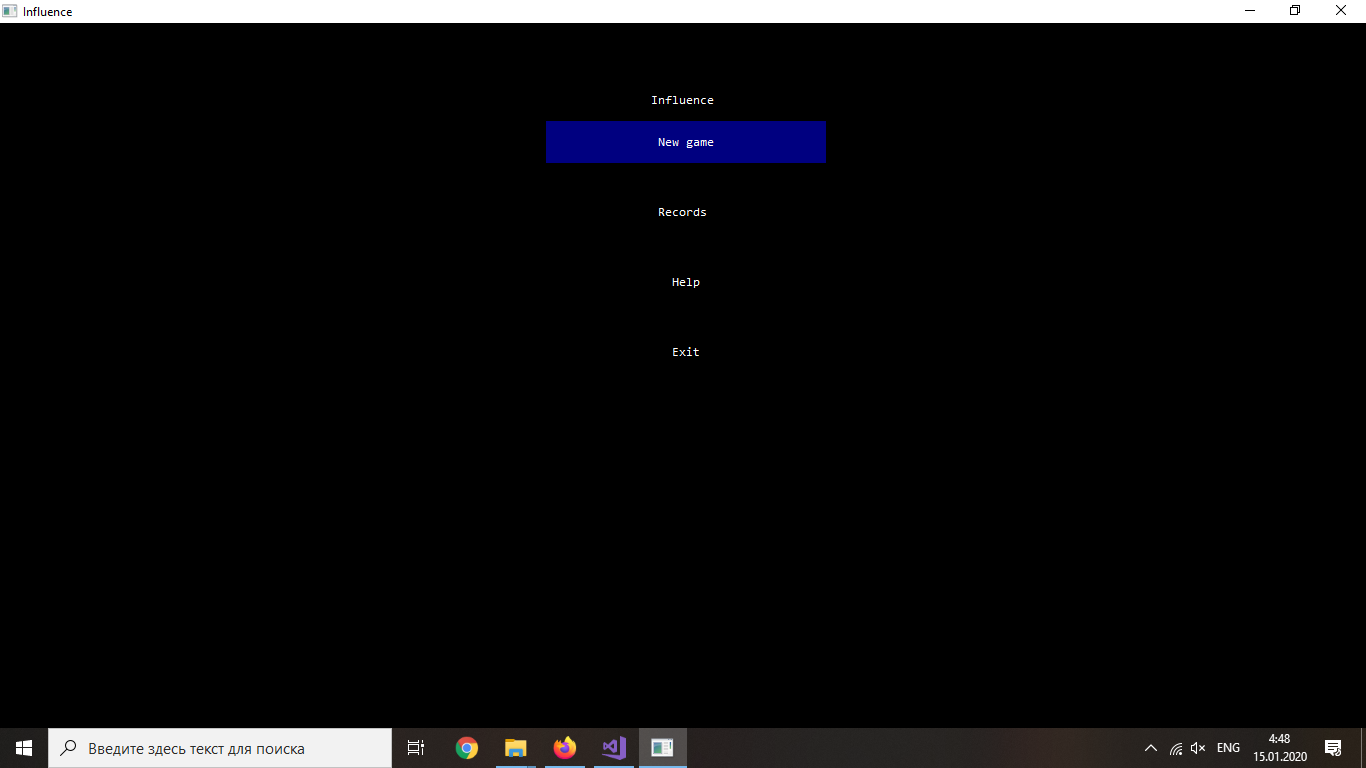


Рис. – Главное меню

На рис. представлен ввод имен игроков, переход к которому осуществляется через выбор пункта главного меню «New game».

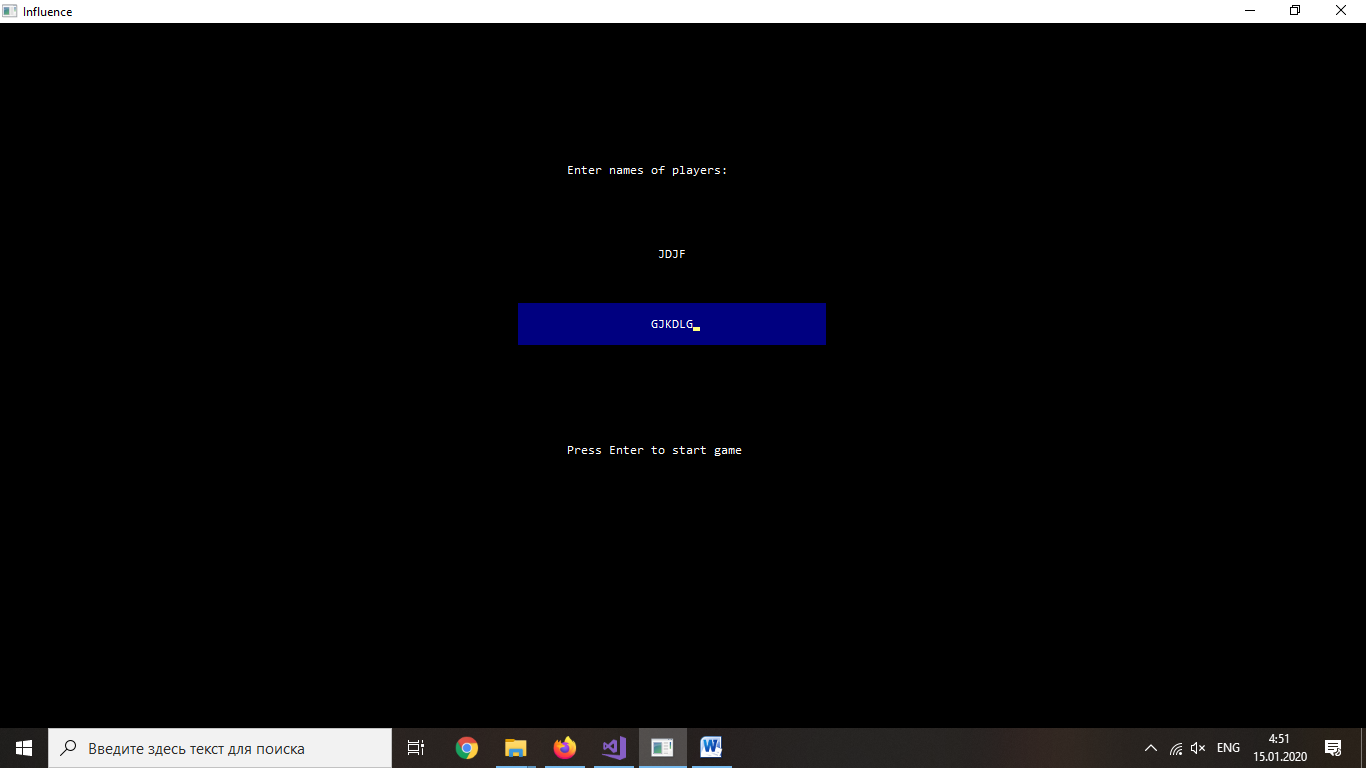


Рис. – Ввод имен игроков

После ввода имен игроков и нажатия клавиши Enter на экране появляется игровое поле, представленное на рис.

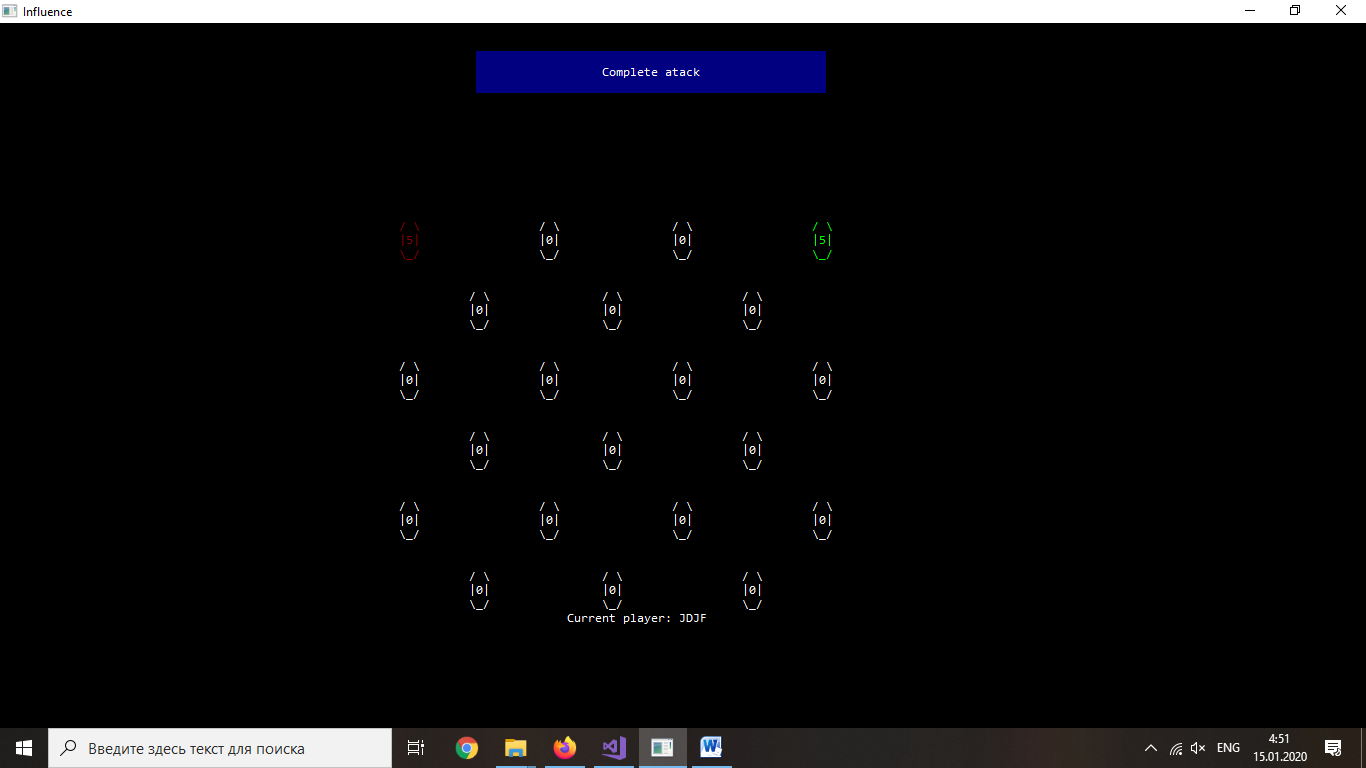


Рис. – Игровое поле в режиме атаки

При нажатии кнопки «Complete atack» происходит переход в режим раздачи очков, представленный на рис.

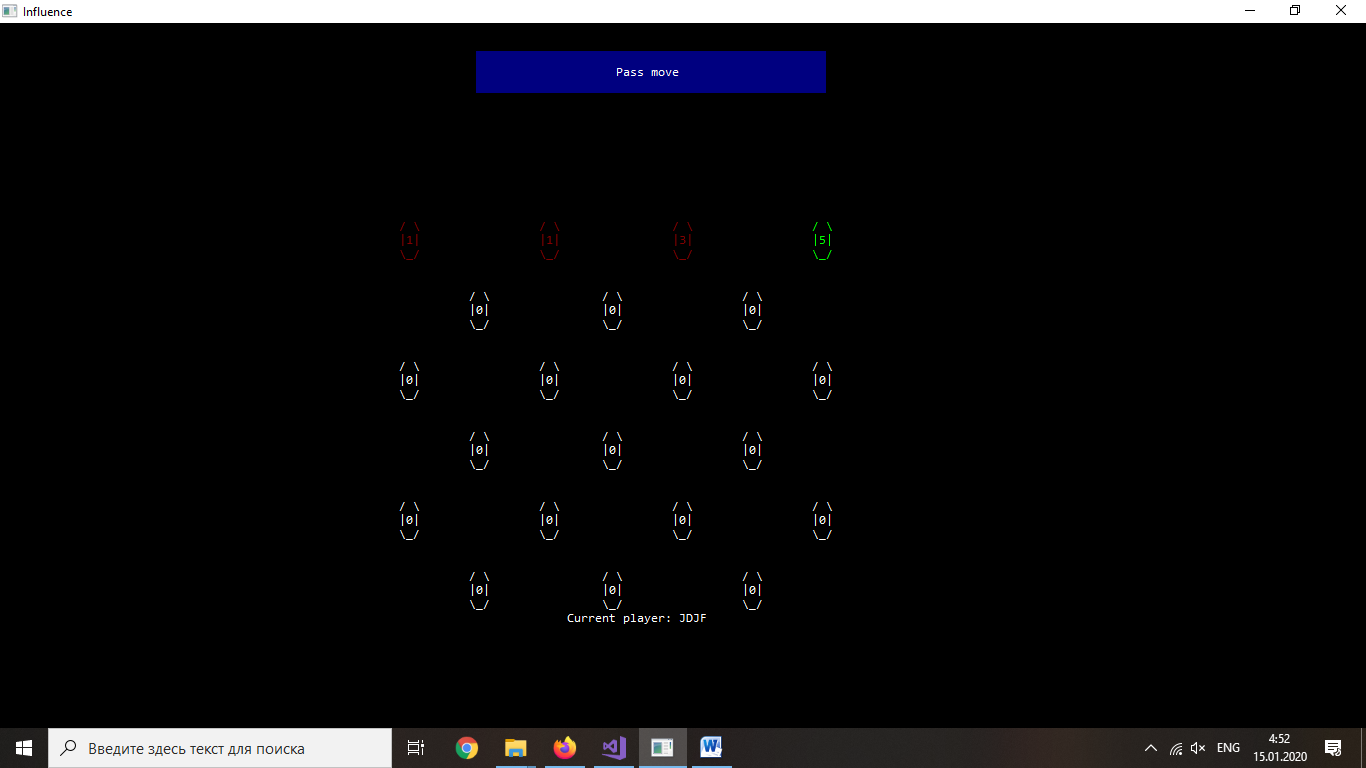


Рис. – Игровое поле в режиме раздачи очков

После того, как на игровом поле остаются ячейки только одного игрока, происходит остановка игры и выводится сообщение, что текущий игрок выиграл.

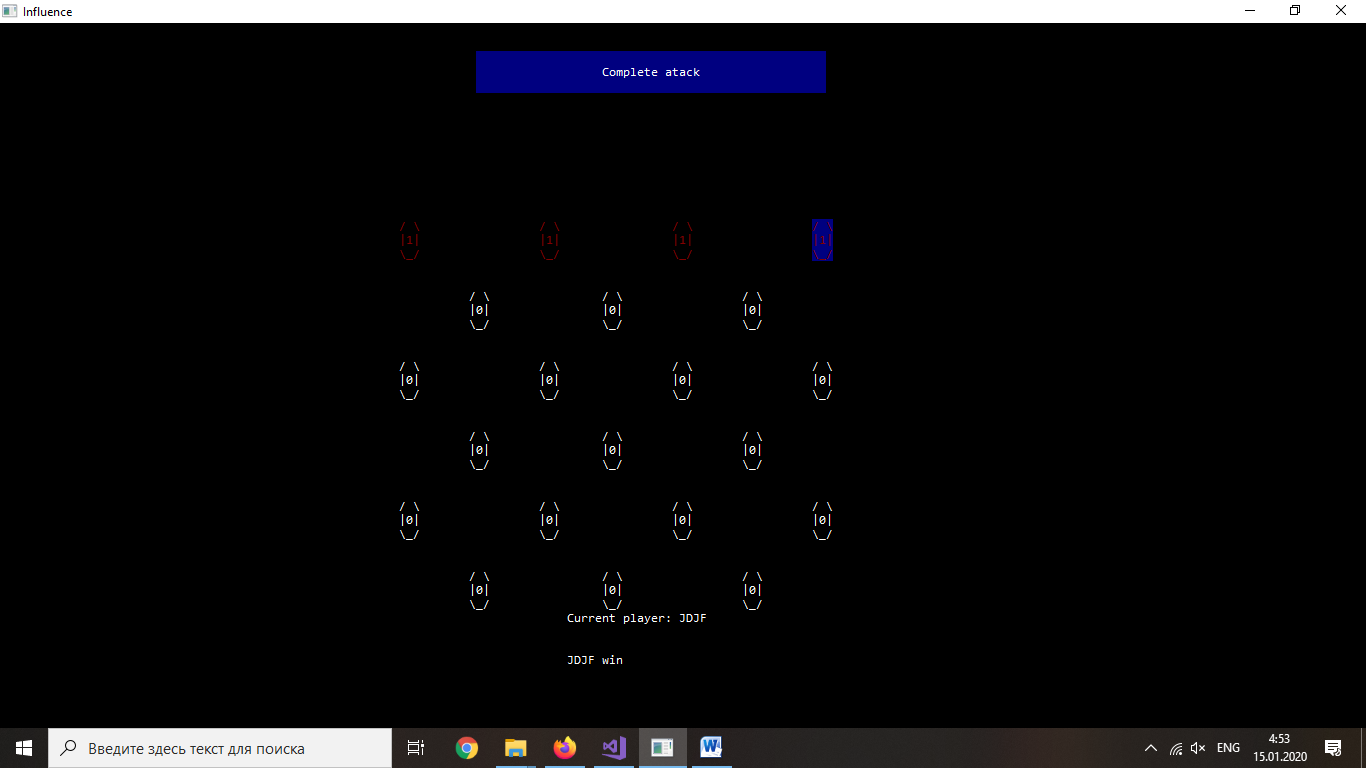


Рис. – Победа игрока

На рис. отображается экран просмотра рекордов, переход на который осуществляется через выбор пункта главного меню «Records».

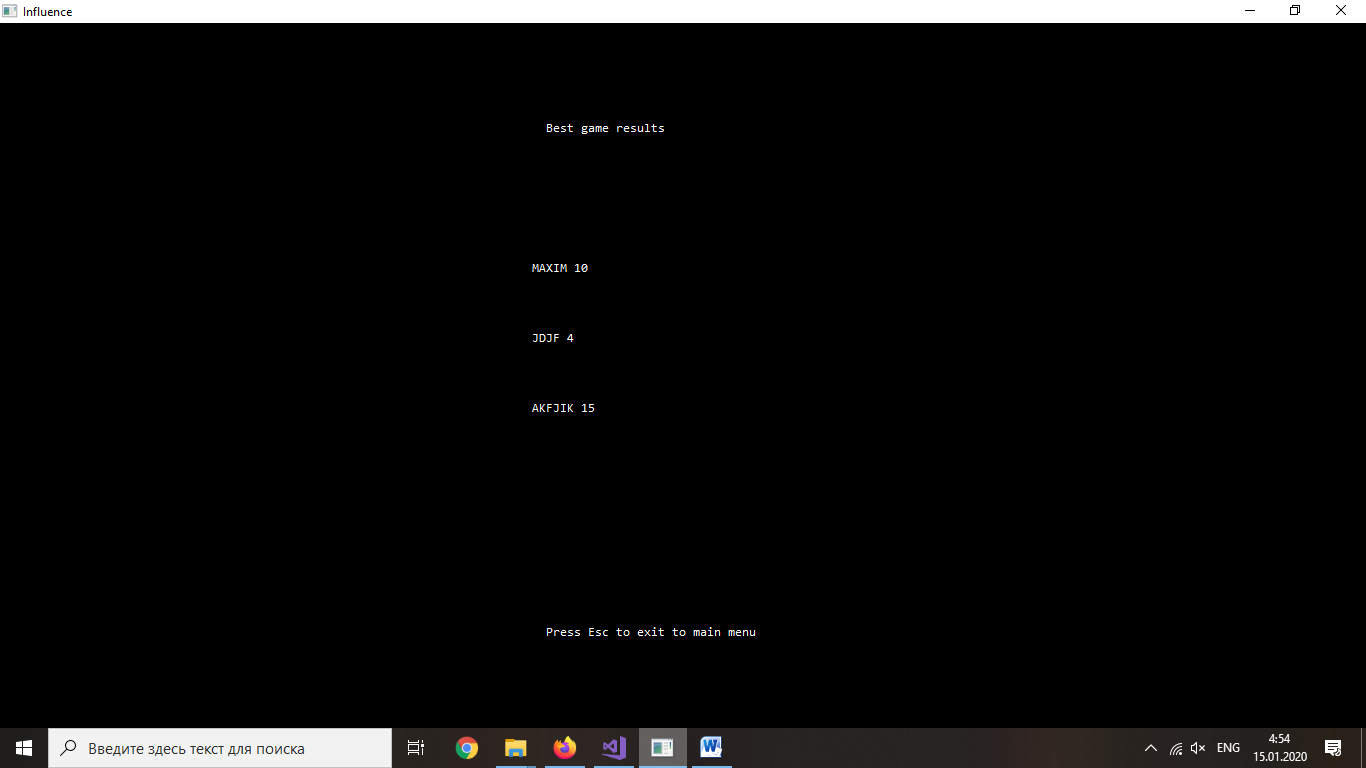


Рис. – Просмотр рекордов

На рис. отображается экран просмотра справки, который доступен при выборе пункта меню «Help».

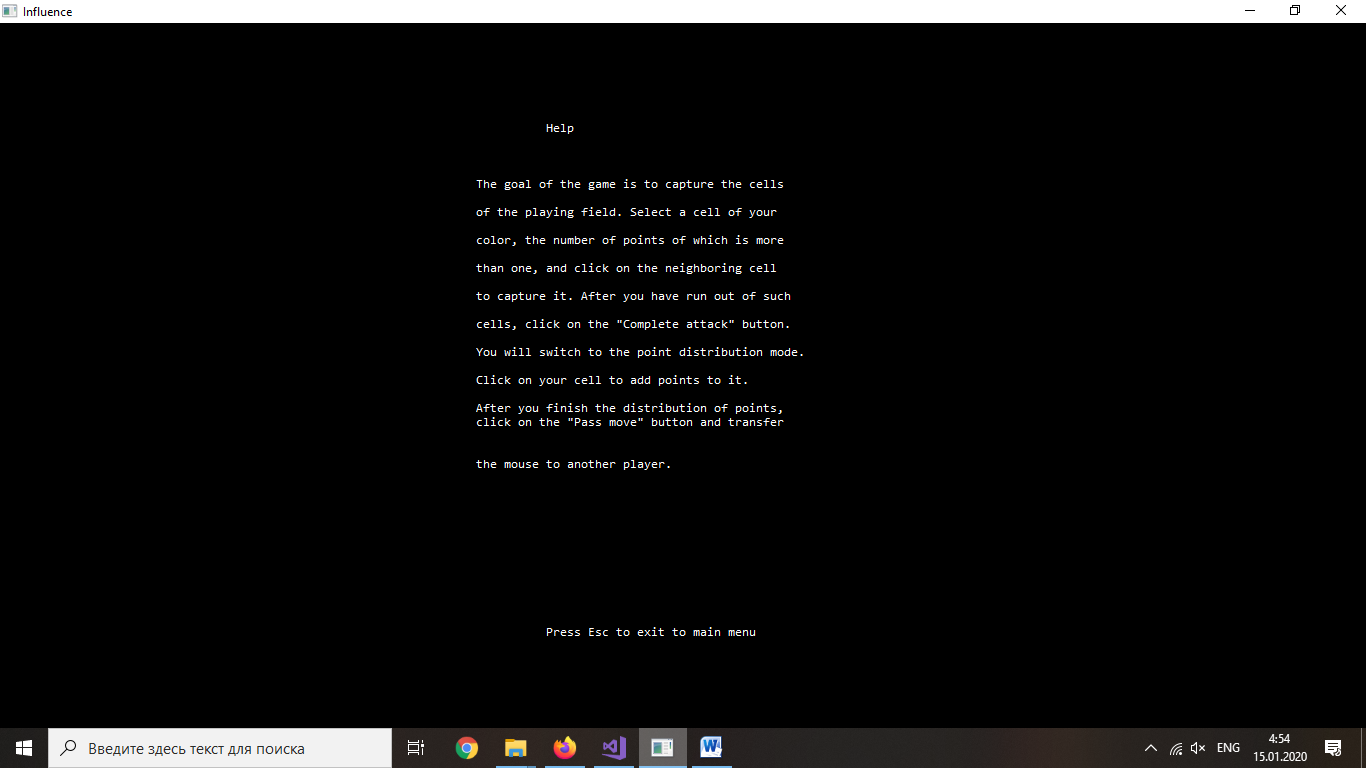


Рис. – Просмотр справки

## Результаты работы Windows Forms реализации

На следующих рисунках представлены результаты работы программы, реализованной при помощи Windows Forms.

На рис. отображается главное меню игры, появляющееся при запуске программы.

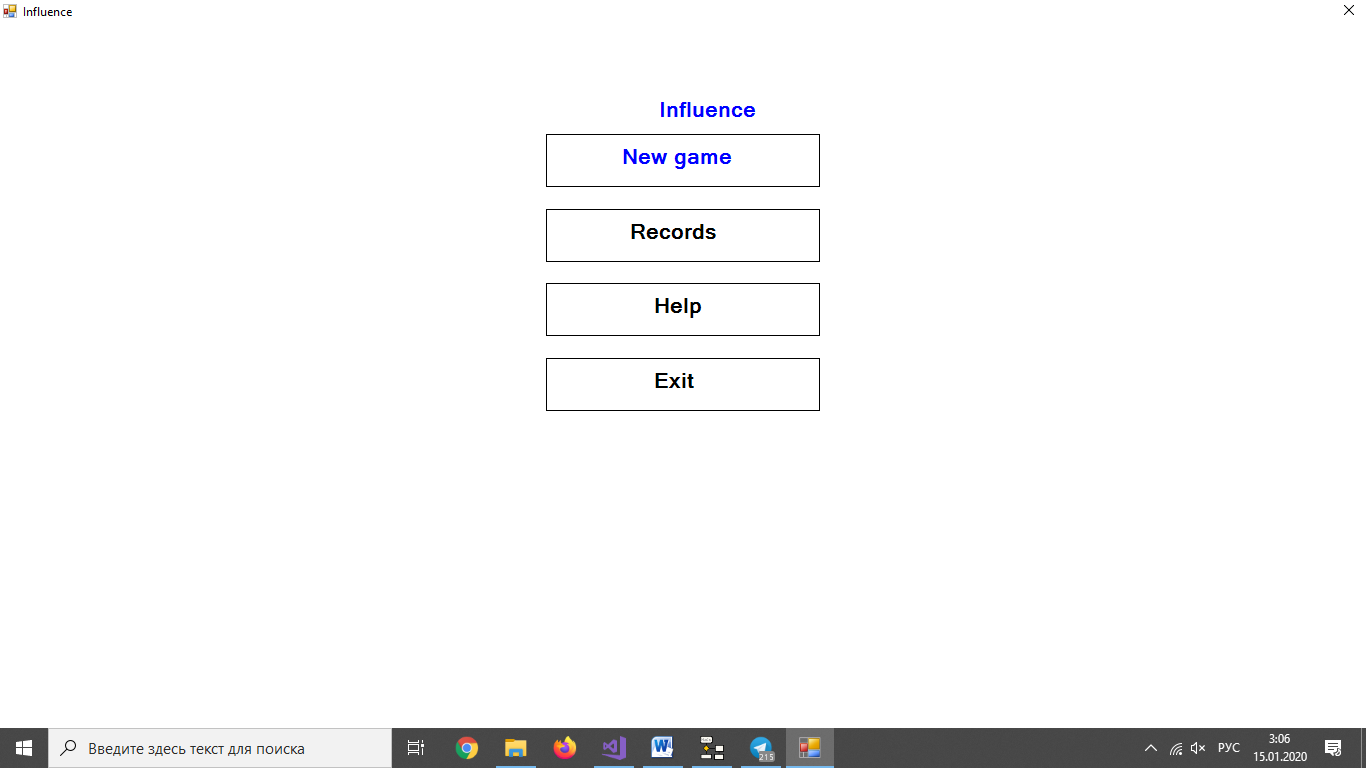


Рис. – Главное меню

На рис. представлен ввод имен игроков, переход к которому осуществляется через выбор пункта главного меню «New game».



Рис. – Ввод имен игроков

После ввода имен игроков и нажатия клавиши Enter на экране появляется игровое поле, представленное на рис.

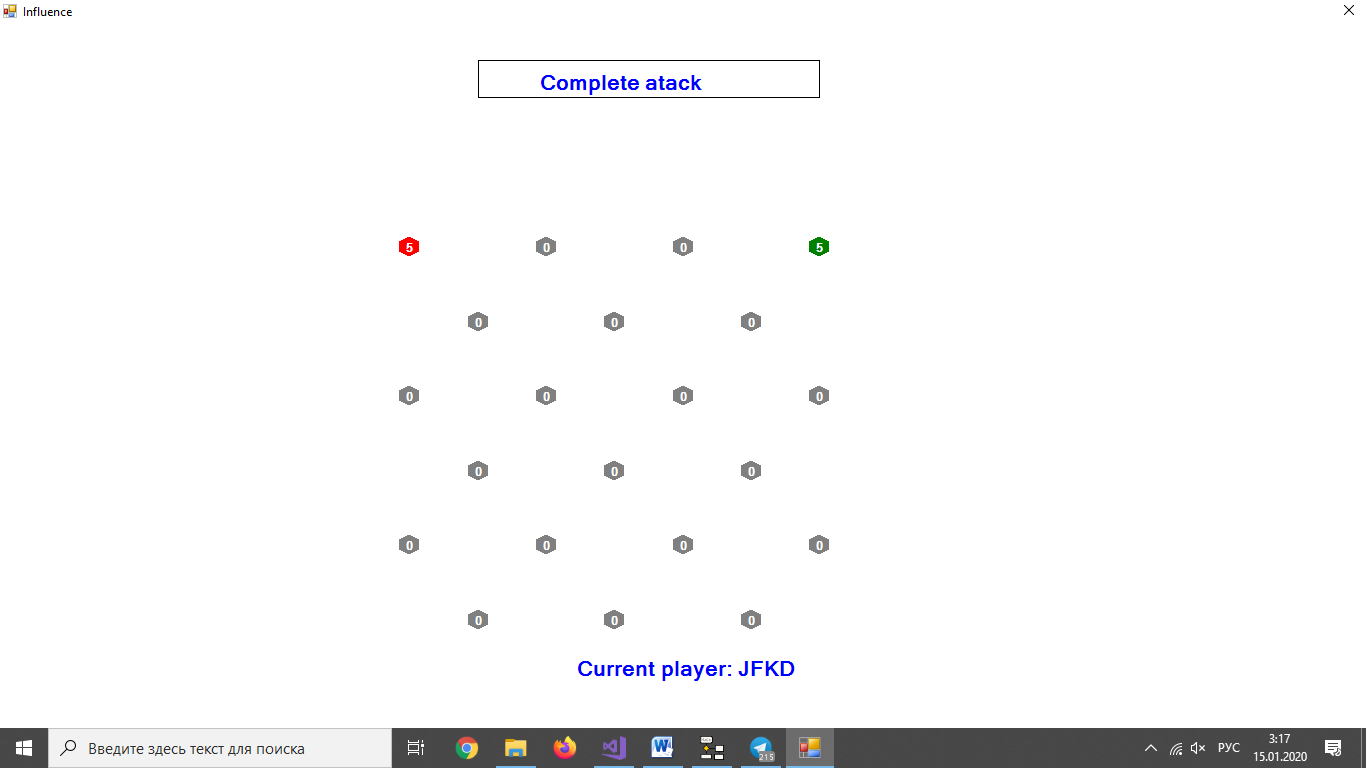


Рис. – Игровое поле в режиме атаки

При нажатии кнопки «Complete atack» происходит переход в режим раздачи очков, представленный на рис.

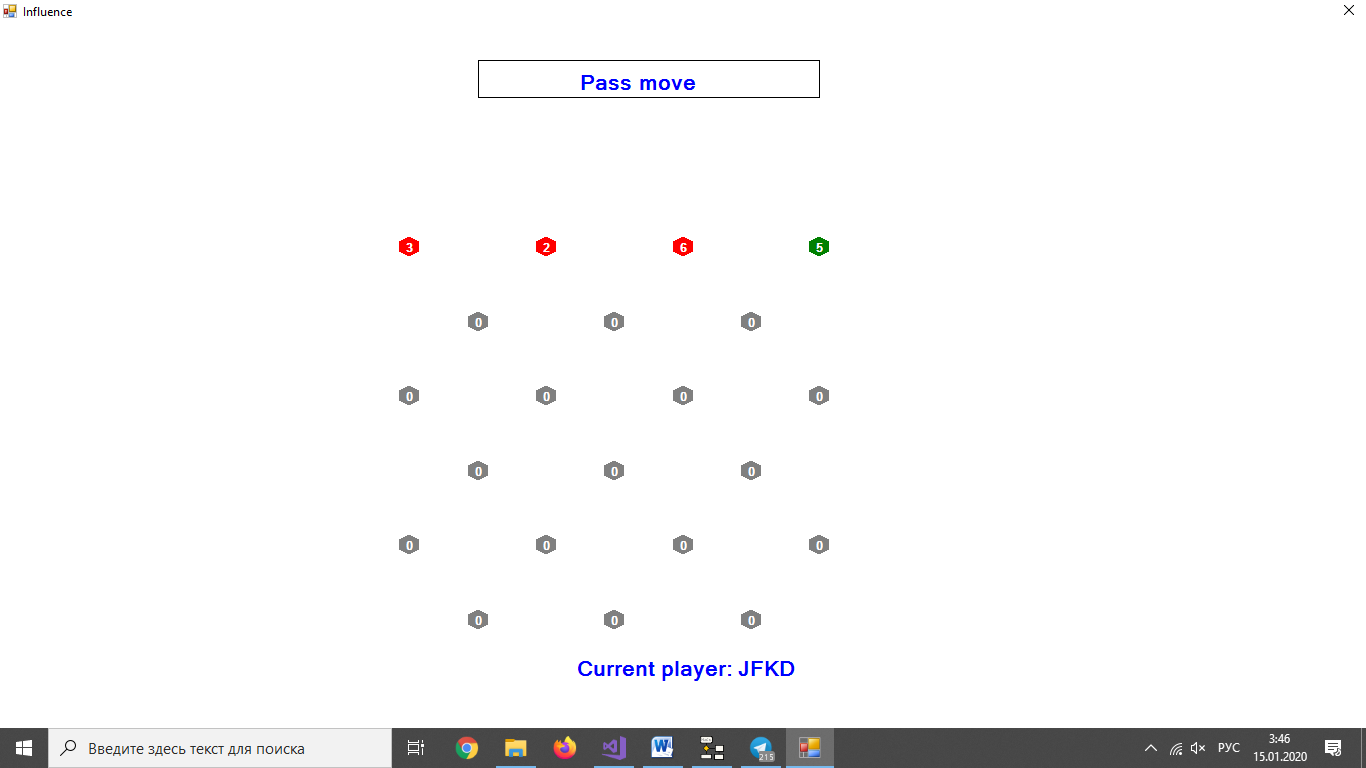


Рис. – Игровое поле в режиме раздачи очков

После того, как на игровом поле остаются ячейки только одного игрока, происходит остановка игры и выводится сообщение, что текущий игрок выиграл.

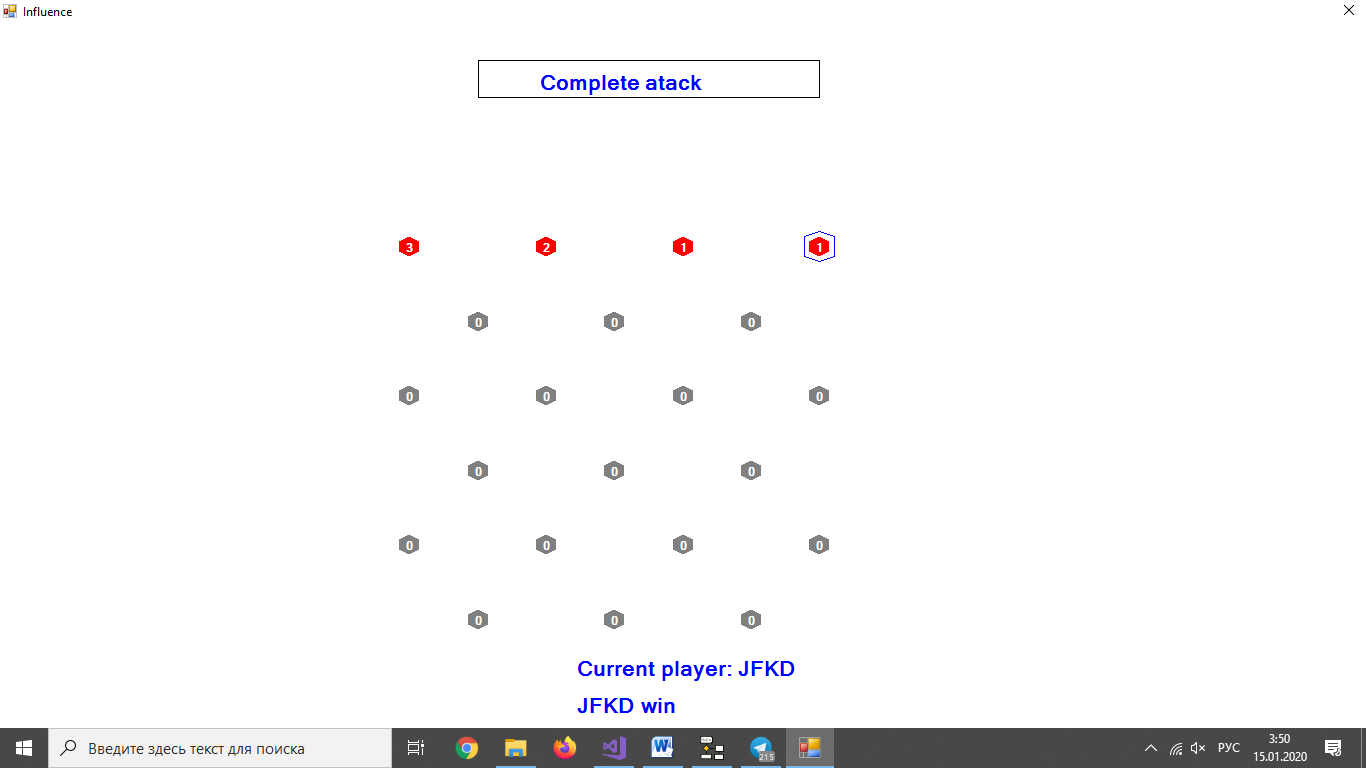


Рис. – Победа игрока

На рис. отображается экран просмотра рекордов, переход на который осуществляется через выбор пункта главного меню «Records».

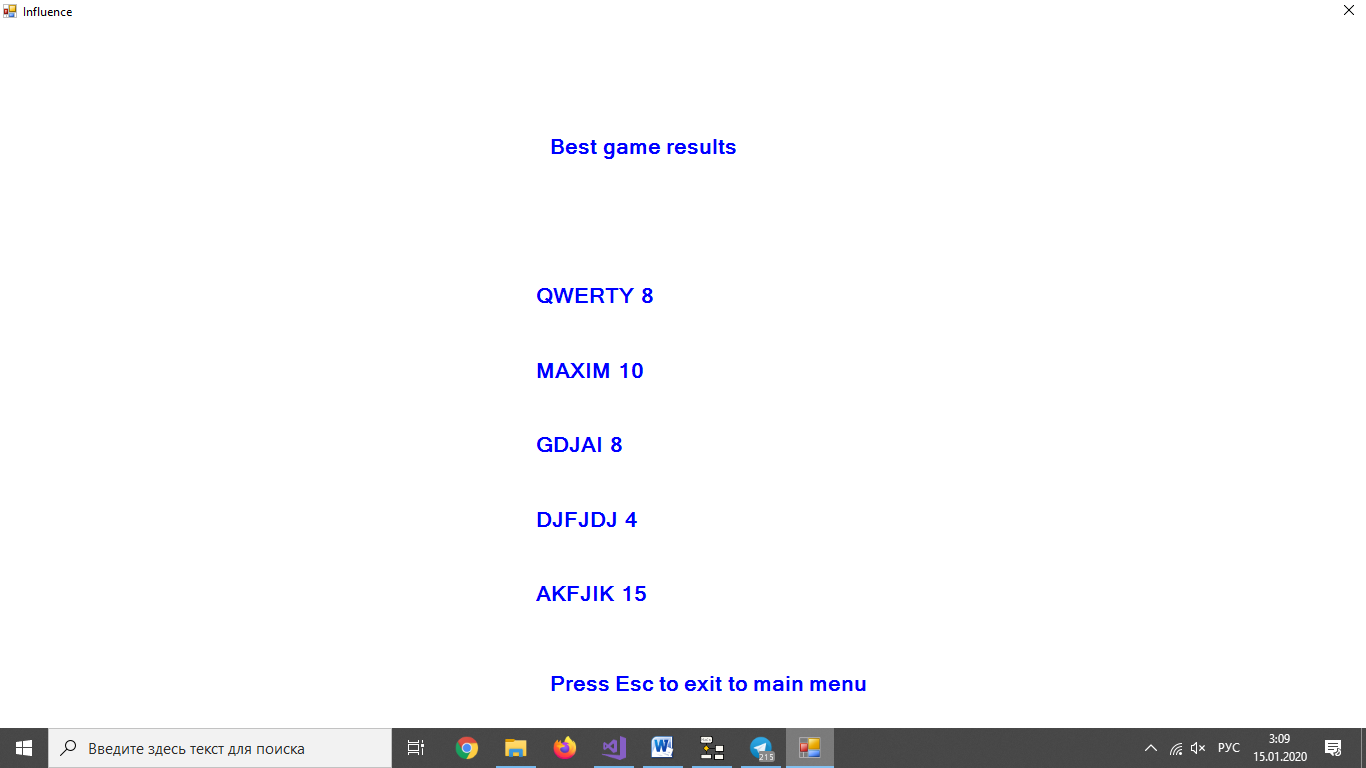


Рис. – Просмотр рекордов

На рис. отображается экран просмотра справки, который доступен при выборе пункта меню «Help».

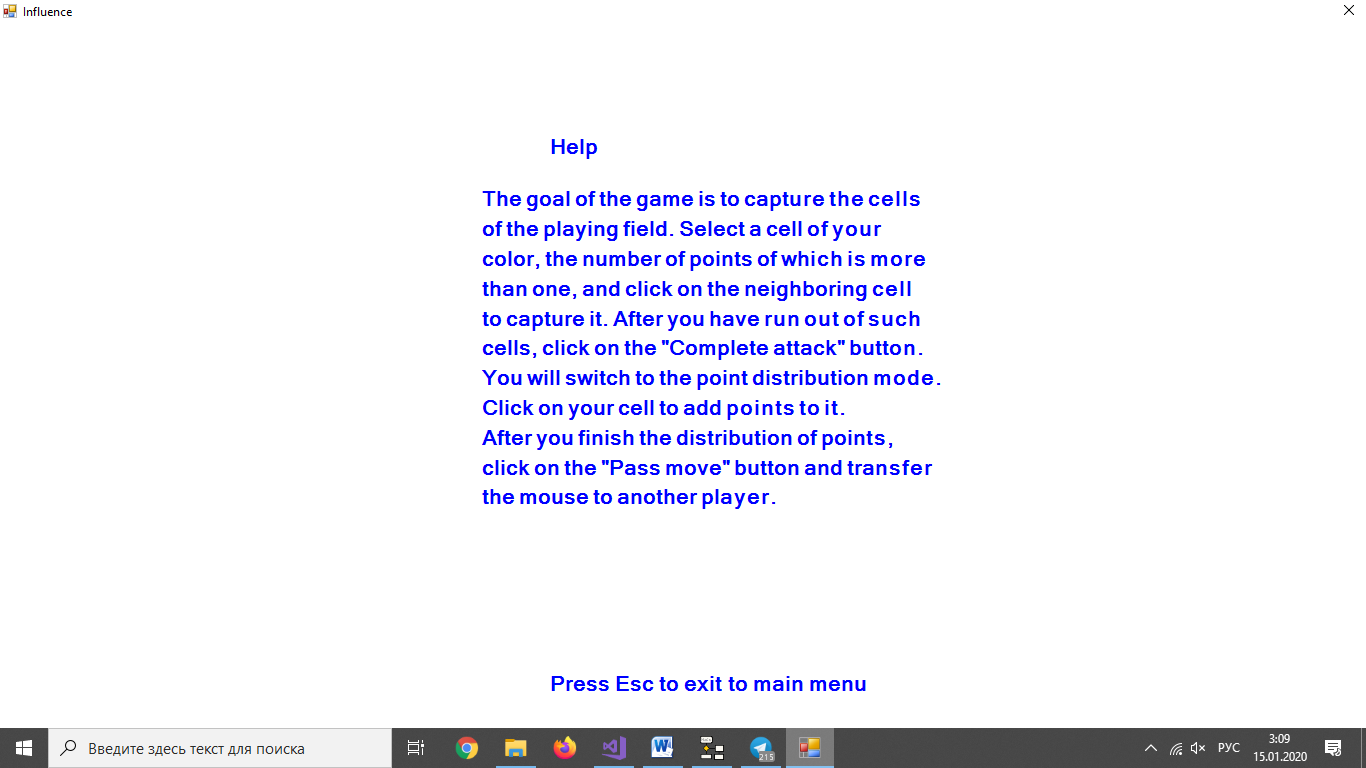


Рис. – Просмотр справки

Заключение

В результате работы над курсовым проектом была разработана игровая программа «Influence». При разработке были использованы программа Diagram Designer для разработки прототипов пользовательского интерфейса и интегрированная среда разработки Microsoft Visual Studio 2017 Community.

В процессе разработки был использован архитектурный паттерн Модель – Представление – Контроллер, что позволило сделать игру легко модифицируемой. Также это позволило облегчить переносимость игры, что было доказано реализацией игры в консольном и Windows Forms режимах без модификации основных слоёв программы.

В работе были использованы шаблоны проектирования «Одиночка», «Абстрактная фабрика» и «Адаптер».

Кроме того, была использована многопоточность для работы вспомогательных модулей программы.

Приложения

MainController.cs

namespace Controller

{

/// <summary>

/// Главный контроллер

/// </summary>

public class MainController

{

/// <summary>

/// Платформа

/// </summary>

private Platform \_platform;

/// <summary>

/// Текущее состояние

/// </summary>

private ApplicationStates \_currentState;

/// <summary>

/// Экземпляр текущего контроллера состояния

/// приложения

/// </summary>

private BaseContoller \_currentController;

/// <summary>

/// Экземпляр текущей фабрики контроллера

/// </summary>

private FactoryOfContollers \_currentFactoryOfControllers;

/// <summary>

/// Конструктор

/// </summary>

/// <param name="parPlatform">Платформа</param>

public MainController(Platform parPlatform)

{

\_platform = parPlatform;

}

/// <summary>

/// Запускает главный контроллер

/// </summary>

public void Start()

{

\_currentState = ApplicationStates.MenuWork;

\_currentFactoryOfControllers = new FactoryOfMenuControllers();

ChangeState(\_currentState, \_currentFactoryOfControllers);

\_platform.Initialize();

}

/// <summary>

/// Изменяет состояние приложения

/// </summary>

/// <param name="parState">Состояние приложения</param>

/// <param name="parFactoryOfContollers">Фабрика контроллера</param>

private void ChangeState(ApplicationStates parState, FactoryOfContollers parFactoryOfContollers)

{

\_currentState = parState;

\_currentFactoryOfControllers = parFactoryOfContollers;

if (parState != ApplicationStates.Exit)

{

\_currentController = \_currentFactoryOfControllers.CreateController(\_platform);

\_currentController.ChangeState += OnChangeState;

}

else

{

\_platform.Drop();

}

}

/// <summary>

/// Обрабатывает событие изменения

/// состояния приложения

/// </summary>

/// <param name="parSender">Источник события</param>

/// <param name="parE">Параметры события</param>

private void OnChangeState(object parSender, ChangeStateArgs parE)

{

\_currentController.View?.Platform.UnsubscribeAllEvents();

ChangeState(parE.ApplicationState, parE.FactoryOfContollers);

}

}

}

GameField.cs

namespace Model

{

/// <summary>

/// Игровое поле

/// </summary>

public class GameField

{

/// <summary>

/// Событие перерисовки игрового поля

/// </summary>

public event dPaintHandler PaintEvent;

/// <summary>

/// Событие окончания игры

/// </summary>

public event EventHandler FinishedEvent;

/// <summary>

/// Объект класса, который отвечает за логику

/// выполнения ходов

/// </summary>

private MoveRunner \_moveRunner;

/// <summary>

/// Текущие игроки

/// </summary>

private List<Player> \_players;

/// <summary>

/// Номер текущего игрока

/// </summary>

private int \_currentPlayer;

/// <summary>

/// Текущее состояние игры

/// </summary>

private GameStates \_currentGameState;

/// <summary>

/// Кнопка переключения состояний игры

/// </summary>

private Button \_button;

/// <summary>

/// Объект класса, отвечающего за запись игрового результата

/// </summary>

private RecordsWriter \_recordsWriter;

/// <summary>

/// Сообщение о выигрыше

/// </summary>

private string \_winMessage;

/// <summary>

/// Ячейки игрового поля

/// </summary>

public Cell[,] Cells

{

get

{

return \_moveRunner.Cells;

}

}

/// <summary>

/// Кнопка переключения состояний игры

/// </summary>

public Button Button

{

get

{

return \_button;

}

set

{

\_button = value;

}

}

/// <summary>

/// Сообщение о выигрыше

/// </summary>

public string WinMessage

{

get

{

return \_winMessage;

}

}

/// <summary>

/// Конструктор

/// </summary>

/// <param name="parVerticalSize">Вертикальный размер поля</param>

/// <param name="parHorizontalSize">Горизонтальный размер поля</param>

/// <param name="parPlayers">Список игроков</param>

public GameField(int parVerticalSize, int parHorizontalSize, List<Player> parPlayers)

{

\_moveRunner = new MoveRunner(new MapBuilder().BuildMap(parVerticalSize, parHorizontalSize, parPlayers));

\_currentGameState = GameStates.Select;

\_players = parPlayers;

\_currentPlayer = 0;

\_button = new Button(35.0f, 5.0f, 60.0f, 10.0f, "Complete atack");

\_recordsWriter = new RecordsWriter();

FinishedEvent += OnFinishedEvent;

}

/// <summary>

/// Обрабатывает событие окончания игры

/// </summary>

/// <param name="parSender">Источник события</param>

/// <param name="parE">Параметры события</param>

private void OnFinishedEvent(object parSender, EventArgs parE)

{

GetActivePlayer().Score = GetPlayerScore(GetActivePlayer());

\_recordsWriter.RecordResult(GetActivePlayer());

\_winMessage = GetActivePlayer().Name + " win";

}

/// <summary>

/// Получает счет игрока

/// </summary>

/// <param name="parPlayer">Объект игрока</param>

/// <returns>Счет игрока</returns>

private int GetPlayerScore(Player parPlayer)

{

int result = 0;

int rows = Cells.GetLength(0);

int colomns = Cells.GetLength(1);

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

{

for (int j = 0; j < colomns; j++)

{

if (null != Cells[i, j])

{

if (Cells[i, j].Owner == parPlayer)

{

result += Cells[i, j].Score;

}

}

}

}

return result;

}

/// <summary>

/// Инициализирует игровое поле

/// </summary>

public void Initialize()

{

PaintEvent?.Invoke();

}

/// <summary>

/// Выбирает ячейку

/// </summary>

private void SelectCell()

{

Cell clickedCell = GetFocusedCell();

UnselectAllCells();

if (clickedCell != null)

{

if (clickedCell?.Owner == GetActivePlayer())

{

clickedCell.ActiveCell();

\_currentGameState = GameStates.Atack;

}

}

PaintEvent?.Invoke();

}

/// <summary>

/// Производит атаку на ячейку

/// </summary>

private void AtackCell()

{

Cell selectedCell = GetSelectedCell();

Cell clickedCell = GetFocusedCell();

if (selectedCell != null && clickedCell != null)

{

if (clickedCell?.Owner == GetActivePlayer())

{

\_currentGameState = GameStates.Select;

selectedCell.DisactiveCell();

clickedCell.ActiveCell();

}

else

{

\_moveRunner.Move(selectedCell, clickedCell, GetActivePlayer());

if (IsFinishedGame())

{

\_currentGameState = GameStates.Finished;

FinishedEvent?.Invoke(this, EventArgs.Empty);

}

}

}

PaintEvent?.Invoke();

\_button.CallPaintEvent();

}

/// <summary>

/// Раздает очки

/// </summary>

private void DistributeScore()

{

Cell clickedCellCoords = GetFocusedCell();

if (null != clickedCellCoords)

{

if (clickedCellCoords.Owner == GetActivePlayer())

{

if (GetActivePlayer().Score > 0)

{

GetActivePlayer().Score -= 1;

clickedCellCoords.Score += 1;

}

}

PaintEvent?.Invoke();

\_button.CallPaintEvent();

}

}

/// <summary>

/// Снимает выделение со всех ячеек

/// </summary>

private void UnselectAllCells()

{

int rows = Cells.GetLength(0);

int colomns = Cells.GetLength(1);

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

{

for (int j = 0; j < colomns; j++)

{

if (Cells[i, j] != null)

{

Cells[i, j].DisactiveCell();

}

}

}

}

/// <summary>

/// Передает ход следующему игроку

/// </summary>

private void PassMove()

{

if (\_currentPlayer >= \_players.Count - 1)

{

\_currentPlayer = 0;

}

else

{

\_currentPlayer++;

}

UnselectAllCells();

\_currentGameState = GameStates.Select;

\_button.CallPaintEvent();

}

/// <summary>

/// Получает ячейку, на которую

/// наведен курсор

/// </summary>

/// <returns>Объект ячейки</returns>

private Cell GetFocusedCell()

{

Cursor cursor = Cursor.GetInstance();

int rows = Cells.GetUpperBound(0) + 1;

int colomns = Cells.GetUpperBound(1) + 1;

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

{

for (int j = 0; j < colomns; j++)

{

if (null != Cells[i, j])

{

if ((cursor.X >= Cells[i, j].X - 1.5f) && (cursor.Y >= Cells[i, j].Y - 1.5f)

&& (cursor.X <= Cells[i, j].X + 1.5f) && (cursor.Y <= Cells[i, j].Y + 1.5f))

{

return Cells[i, j];

}

}

}

}

return null;

}

/// <summary>

/// Получает выбранную ячейку

/// </summary>

/// <returns>Объект ячейки</returns>

private Cell GetSelectedCell()

{

int rows = Cells.GetLength(0);

int colomns = Cells.GetLength(1);

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

{

for (int j = 0; j < colomns; j++)

{

if (null != Cells[i, j])

{

if (Cells[i, j].CellStatus == CellStatuses.Active)

{

return Cells[i, j];

}

}

}

}

return null;

}

/// <summary>

/// Получает число ячеек игрока

/// </summary>

/// <param name="parPlayer">Объект игрока</param>

/// <returns>Число ячеек</returns>

private int GetPlayerNumOfCells(Player parPlayer)

{

int result = 0;

int rows = Cells.GetLength(0);

int colomns = Cells.GetLength(1);

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

{

for (int j = 0; j < colomns; j++)

{

if (null != Cells[i, j])

{

if (Cells[i, j].Owner == parPlayer)

{

result++;

}

}

}

}

return result;

}

/// <summary>

/// Завершает атаку или передает ход

/// </summary>

public void CompleteAtackOrPassMove()

{

switch (\_currentGameState)

{

case GameStates.Select:

case GameStates.Atack:

{

UnselectAllCells();

GetActivePlayer().Score = CalculateScorePlayer();

\_currentGameState = GameStates.ScoreDistributing;

\_button.Name = "Pass move";

break;

}

case GameStates.ScoreDistributing:

{

UnselectAllCells();

PassMove();

\_button.Name = "Complete atack";

break;

}

}

PaintEvent?.Invoke();

\_button.CallPaintEvent();

}

/// <summary>

/// Выполняет одно из игровых действий

/// </summary>

public void PerformGameAction()

{

switch (\_currentGameState)

{

case GameStates.Select:

SelectCell();

break;

case GameStates.Atack:

AtackCell();

break;

case GameStates.ScoreDistributing:

DistributeScore();

break;

}

}

/// <summary>

/// Вычисляет счет текущего игрока

/// </summary>

/// <returns>Счет текущего игрока</returns>

private int CalculateScorePlayer()

{

int scorePlayer = GetPlayerNumOfCells(GetActivePlayer());

return scorePlayer;

}

/// <summary>

/// Возвращает текущего игрока

/// </summary>

/// <returns>Текущий игрок</returns>

public Player GetActivePlayer()

{

return \_players[\_currentPlayer];

}

/// <summary>

/// Проверяет, окончена ли игра

/// </summary>

/// <returns>Признак окончания игры</returns>

private bool IsFinishedGame()

{

int rows = Cells.GetLength(0);

int colomns = Cells.GetLength(1);

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

{

for (int j = 0; j < colomns; j++)

{

if (null != Cells[i, j])

{

if (Cells[i, j].Owner != null && Cells[i, j].Owner != GetActivePlayer())

{

return false;

}

}

}

}

return true;

}

}

}

Menu.cs

namespace Model

{

/// <summary>

/// Меню

/// </summary>

public class Menu

{

/// <summary>

/// Событие перерисовки

/// </summary>

public event dPaintHandler PaintEvent;

/// <summary>

/// Название меню

/// </summary>

private string \_name;

/// <summary>

/// Список пунктов меню

/// </summary>

private SortedList<int, MenuItem> \_menuItems = new SortedList<int, MenuItem>();

/// <summary>

/// Название меню

/// </summary>

public string Name

{

get

{

return \_name;

}

}

/// <summary>

/// Список пунктов меню

/// </summary>

public SortedList<int, MenuItem> MenuItems

{

get

{

return \_menuItems;

}

set

{

\_menuItems = value;

}

}

/// <summary>

/// Конструктор

/// </summary>

/// <param name="parName">Название меню</param>

public Menu(string parName)

{

\_name = parName;

}

/// <summary>

/// Инициализирует меню

/// </summary>

public void Initialize()

{

PaintEvent?.Invoke();

}

/// <summary>

/// Добавляет элемент в меню

/// </summary>

/// <param name="parId">Идентификатор</param>

/// <param name="parName">Название пункта</param>

public void AddItem(int parId, string parName)

{

if (null != parName)

{

\_menuItems.Add(parId, new MenuItem(parId, parName));

if (parId == 0)

{

\_menuItems[parId].MenuItemStatus = ItemStatuses.Selected;

}

else

{

\_menuItems[parId].MenuItemStatus = ItemStatuses.Unselected;

}

}

}

/// <summary>

/// Переводит фокус на следующиий элемент меню

/// </summary>

public void Next()

{

for (int i = 0; i < \_menuItems.Count; i++)

{

if (\_menuItems[i].MenuItemStatus == ItemStatuses.Selected)

{

\_menuItems[i].MenuItemStatus = ItemStatuses.Unselected;

if (i == \_menuItems.Count - 1)

{

\_menuItems[0].MenuItemStatus = ItemStatuses.Selected;

break;

}

else

{

\_menuItems[i + 1].MenuItemStatus = ItemStatuses.Selected;

break;

}

}

}

PaintEvent?.Invoke();

}

/// <summary>

/// Переводит фокус на предыдущий элемент меню

/// </summary>

public void Previous()

{

for (int i = 0; i < \_menuItems.Count; i++)

{

if (\_menuItems[i].MenuItemStatus == ItemStatuses.Selected)

{

\_menuItems[i].MenuItemStatus = ItemStatuses.Unselected;

if (i == 0)

{

\_menuItems[\_menuItems.Count - 1].MenuItemStatus = ItemStatuses.Selected;

break;

}

else

{

\_menuItems[i - 1].MenuItemStatus = ItemStatuses.Selected;

break;

}

}

}

PaintEvent?.Invoke();

}

}

}

MapBuilder.cs

namespace Model

{

/// <summary>

/// Строитель карты

/// </summary>

public class MapBuilder

{

/// <summary>

/// Строит карту

/// </summary>

/// <param name="parVerticalSize">Вертикальный размер</param>

/// <param name="parHorizontalSize">Горизонтальный размер</param>

/// <param name="parPlayers">Список игроков</param>

/// <returns>Массив игровых ячеек</returns>

public Cell[,] BuildMap(int parVerticalSize, int parHorizontalSize, List<Player> parPlayers)

{

Cell[,] map = CreateGameCells(parVerticalSize, parHorizontalSize);

map = SetPlayers(map, parPlayers);

return map;

}

/// <summary>

/// Создает массив игровых ячеек

/// </summary>

/// <param name="parVerticalSize">Вертикальный размер массива</param>

/// <param name="parHorizontalSize">Горизонтальный размер массива</param>

/// <returns>Массив ячеек</returns>

private Cell[,] CreateGameCells(int parVerticalSize, int parHorizontalSize)

{

const int DELTA = 10;

const int X = 30;

const int Y = 30;

const int Y\_SHIFT = 5;

Cell[,] cells = new Cell[parVerticalSize, parHorizontalSize];

int rows = cells.GetLength(0);

int colomns = cells.GetLength(1);

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

{

for (int j = 0; j < colomns; j++)

{

if (i % 2 == 1)

{

cells[i, j] = new Cell(j \* DELTA + Y + Y\_SHIFT, i \* DELTA + X);

cells[i, j].I = i;

cells[i, j].J = j;

}

else

{

cells[i, j] = new Cell(j \* DELTA + Y, i \* DELTA + X);

cells[i, j].I = i;

cells[i, j].J = j;

}

}

}

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

{

if (i % 2 == 1)

{

cells[i, cells.GetUpperBound(1)] = null;

}

}

return cells;

}

/// <summary>

/// Устанавливает игроков для первоначальных ячеек

/// </summary>

/// <param name="parCells">Массив ячеек</param>

/// <param name="parPlayers">Список игроков</param>

/// <returns>Массив ячеек</returns>

private Cell[,] SetPlayers(Cell[,] parCells, List<Player> parPlayers)

{

Cell[,] map = parCells;

map[0, 0].Owner = parPlayers[0];

map[0, 3].Owner = parPlayers[1];

map[0, 0].Score = 5;

map[0, 3].Score = 5;

return map;

}

}

}

GameFieldController.cs

namespace Controller

{

/// <summary>

/// Контроллер игрового поля

/// </summary>

public class GameFieldController : BaseContoller

{

/// <summary>

/// Число рядов

/// </summary>

private const int VERTICAL\_SIZE = 6;

/// <summary>

/// Число ячеек в строке

/// </summary>

private const int HORIZONTAL\_SIZE = 4;

/// <summary>

/// Экземпляр игрового поля

/// </summary>

private GameField \_game;

/// <summary>

/// Контроллер кнопки

/// </summary>

private ButtonController \_buttonController;

/// <summary>

/// Конструктор

/// </summary>

/// <param name="parPlayers">Игроки</param>

/// <param name="parPlatform">Платформа</param>

public GameFieldController(List<Player> parPlayers, Platform parPlatform)

{

\_game = new GameField(VERTICAL\_SIZE, HORIZONTAL\_SIZE, parPlayers);

\_buttonController = new ButtonController(parPlatform, \_game.Button);

View = new GameFieldView(parPlatform, \_game, (ButtonView)\_buttonController.View);

\_game.Initialize();

\_buttonController.Initialize();

\_game.Button.Click += OnButtonClick;

parPlatform.EscDown += OnEscDown;

parPlatform.Move += OnMove;

parPlatform.Click += OnClick;

}

/// <summary>

/// Обрабатывает нажатие Esc

/// </summary>

/// <param name="parSender">Источник события</param>

/// <param name="parE">Параметры события</param>

private void OnEscDown(object parSender, EventArgs parE)

{

CallChangeState(this, new ChangeStateArgs(new FactoryOfMenuControllers(), ApplicationStates.MenuWork));

}

/// <summary>

/// Обрабатывает нажатие на кнопку

/// </summary>

/// <param name="parSender">Источник события</param>

/// <param name="parE">Параметры события</param>

private void OnButtonClick(object parSender, EventArgs parE)

{

\_game.CompleteAtackOrPassMove();

}

/// <summary>

/// Обрабатывает событие клика по поверхности платформы

/// </summary>

/// <param name="parSender">Источник события</param>

/// <param name="parE">Параметры события</param>

private void OnClick(object parSender, EventArgs parE)

{

\_game.PerformGameAction();

}

/// <summary>

/// Обрабатывает событие перемещения мыши

/// </summary>

/// <param name="parSender">Источник события</param>

/// <param name="parE">Параметры события</param>

private void OnMove(object parSender, MoveEventArgs parE)

{

Cursor.GetInstance().Move(parE.X, parE.Y);

}

}

}