Министерство науки и высшего образования РФ

Рыбинский государственный авиационный технический университет

имени П.А. Соловьева

КАФЕДРА ВЫЧИСЛИТЕЛЬЫХ СИСТЕМ

КУРСОВАЯ РАБОТА

По дисциплине «Объектно-ориентированное программирование»

на тему «Игра «Сапер»

Выполнил Лебедев А.А.

(фамилия, имя, отчество)

студент гр. ЗИС-23

Преподаватель Пруктишина В.А.

(фамилия, имя, отчество)

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Подпись преподавателя \_\_\_\_\_\_\_\_\_\_\_\_

Дата \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

г. Рыбинск

2025 г.

[Введение 3](#_Toc201008934)

[Основная часть 5](#_Toc201008935)

[1. Анализ задания 5](#_Toc201008936)

[2. Механизм игры 5](#_Toc201008937)

[3. Алгоритм и условия игры 6](#_Toc201008938)

[4. Инструменты выполнения 6](#_Toc201008939)

[4.1. Среда разработки «Visual Studio 2022» 7](#_Toc201008940)

[4.2. Операционная система Windows 7](#_Toc201008941)

[5. Технические требования 7](#_Toc201008942)

[6. Описание классов 8](#_Toc201008943)

[7. Алгоритмы 9](#_Toc201008944)

[7.1. Алгоритм заполнения поля 9](#_Toc201008945)

[7.2. Алгоритм открытия пустых ячеек 10](#_Toc201008946)

[7.3. Алгоритм проверки победы пользователя 10](#_Toc201008947)

[8. Тестирование нагрузочное, тестирование на безопасность 12](#_Toc201008948)

[9. Инструкция пользователя 14](#_Toc201008949)

[Заключение 16](#_Toc201008950)

[Список использованных источников и литературы 18](#_Toc201008951)

[Приложение А 19](#_Toc201008952)

****Введение****

Объектно-ориентированное программирование (ООП) представляет собой фундаментальную парадигму в современной разработке программного обеспечения. Его основополагающие принципы, включающие инкапсуляцию, наследование и полиморфизм, обеспечивают возможность создания гибких, масштабируемых и легко поддерживаемых программных систем.

Эффективным методом для углубленного понимания и практического применения принципов ООП является разработка игровых приложений. В контексте игровых систем объектная модель наглядно демонстрирует взаимодействие различных компонентов программного обеспечения, что способствует более эффективному освоению концепций ООП.

В данной курсовой работе рассматривается разработка классической игры «Сапер» с использованием языка программирования С#. С# был выбран благодаря своей простоте, читаемости кода и широким возможностям в области ООП. Этот язык позволяет сосредоточиться на архитектуре приложения, не углубляясь в излишние технические детали, что особенно важно при изучении основных концепций объектно-ориентированного проектирования.

Для создания графического интерфейса игры использовалась XAML разметка, которая позволяет создавать как простые визуальные компоненты, так и более сложные элементы, которые, однако, требуют большего объема затрачиваемых ресурсов ПК.

Задача игры “Сапер” состоит в обнаружении всех безопасных ячеек игрового поля, избегая при этом соприкосновения со скрытыми минами. Успех в игре достигается посредством логического анализа и дедуктивного определения местоположения мин, которые затем отмечаются визуальными индикаторами.

Выбор игры “Сапер” в качестве основы для курсовой работы обусловлен рядом факторов:

* Интуитивно понятные правила: Базовая механика игры характеризуется простотой и доступностью, что позволяет сконцентрироваться на реализации принципов объектно-ориентированного программирования.
* Четкая объектная модель: Компоненты игры, такие как игровое поле, отдельные ячейки, мины и пользовательский интерфейс, могут быть эффективно представлены как отдельные классы в объектно-ориентированной структуре.
* Практическая значимость: Проект интегрирует работу с графическими элементами, логикой игры и пользовательским взаимодействием, что обеспечивает ценный практический опыт в разработке программного обеспечения.

Таким образом, разработка игры «Сапер» на С# является хорошей демонстрацией многообразия доступных ресурсов языка программирования. В данном проекте использовались такие механизмы, как парсинг XML строк, работа со списками enum, динамическое создание элементов интерфейса, работа с различными цветовыми темами и их переключением, шифрование данных конфигурации, многопоточность и многоуровневый вызов, а также рекурсивная обработка массивов.

****Основная часть****

1. ****Анализ задания****

Данный проект был нацелен на создание игры “Сапер”, спроектированной в соответствии с принципами объектно-ориентированного программирования (ООП). Ключевым требованием являлось обеспечение модульности и гибкости архитектуры, что позволило бы упростить интеграцию новых функций и модификацию существующих компонентов.

В ходе разработки особое внимание уделялось демонстрации ключевых концепций ООП, таких как:

Инкапсуляция: Скрытие внутренних данных и методов объектов, предоставляя контролируемый интерфейс для взаимодействия с ними.

Наследование: Создание новых классов на основе существующих, наследуя их свойства и поведение, и добавляя или изменяя функциональность по мере необходимости.

Полиморфизм: Возможность объектам разных классов реагировать на один и тот же вызов по-разному, обеспечивая гибкость и расширяемость системы.

1. Механизм игры

Задача игрока в “Сапере” - очистить игровое поле, состоящее из прямоугольных ячеек, от скрытых мин. Игрок, открывая ячейки, должен избежать активации мин. Если в открытой ячейке мины нет, отображается число, указывающее на количество мин, расположенных в соседних клетках. Используя эту информацию, игрок пытается вычислить расположение мин и помечает подозрительные ячейки флажками.

Игровой процесс включает в себя три основных действия: открытие ячейки, установка и снятие флажка. Автоматическое открытие смежных пустых ячеек при открытии пустой (без мин) ускоряет процесс. Игра завершается либо победой (открытие всех безопасных ячеек), либо поражением (обнаружение мины). Особенностью реализации является случайная генерация мин в начале каждой партии, обеспечивающая разнообразный игровой опыт. Логика игры включает в себя проверку соседних ячеек и рекурсивное открытие пустых зон, демонстрируя сочетание алгоритмических решений и объектно-ориентированного подхода.

1. Алгоритм и условия игры

Игра начинается с генерации игрового поля заданного размера, на котором в случайных клетках размещаются мины. Количество мин зависит от выбранного уровня сложности. Если игрок открывает клетку с миной — игра завершается поражением. Если клетка безопасна, в ней отображается число, показывающее количество мин в соседних клетках. Если рядом нет ни одной мины, клетка считается пустой, и игра автоматически открывает все смежные пустые клетки.

Условия победы – открыть все клетки, не содержащие мин. Игрок может помечать подозрительные клетки флажками, чтобы избежать случайного нажатия. Реализация включает проверку на победу после каждого хода, а также обработку событий мыши для взаимодействия с игровым полем. Алгоритм гарантирует, что мины не перемещаются после начала игры, а числа всегда корректно отражают их расположение.

1. ****Инструменты выполнения****

C# был выбран как язык написания игры «Сапер» не случайно. Во-первых в индустрии компьютерных игр данный язык занимает лидирующую позицию, благодаря своим сильным сторонам в работе с ООП. Кроме этого язык позволяет создавать приложения с интерфейсов и простой графикой без использования каких либо библиотек, кроме встроенных в операционную систему. Также синтаксис языка отличается свой простотой и аккуратностью. Среда разработки (Visual Studio 2022) предоставляет огромное количество инструментов, упрощающих разработку приложения.

* 1. Среда разработки «Visual Studio 2022»

Visual Studio 2022 является привлекательным выбором для разработки игры “Сапер” благодаря своей мощной и удобной интегрированной среде разработки (IDE), которая значительно ускоряет процесс создания, отладки и профилирования кода благодаря таким функциям, как IntelliSense, рефакторинг и встроенная поддержка Git. Эта IDE предлагает отличную поддержку как C++, обеспечивающего высокую производительность, так и .NET (C#), упрощающего создание графического интерфейса пользователя (GUI) и предоставляющего развитую объектно-ориентированную среду. Visual Studio 2022 предоставляет удобные инструменты, такие как Windows Forms или WPF, для эффективной разработки GUI, а также упрощает интеграцию сторонних библиотек для расширения функциональности. Встроенные инструменты отладки и профилирования помогают находить и исправлять ошибки, а также оптимизировать производительность. Наконец, Visual Studio 2022 Community Edition, будучи бесплатной для многих пользователей, делает её доступным решением для образовательных целей и личных проектов, предоставляя все необходимое для разработки “Сапера” любого уровня сложности.

* 1. Операционная система Windows

Операционная система Windows использовалась в качестве платформы для разработки и тестирования игры. Её широкое распространение и совместимость с С# и .NET делают её удобным выбором для подобных проектов. Кроме того, Windows предоставляет стандартные графические драйверы и API, необходимые для корректного отображения GUI-приложений, что гарантирует стабильную работу игры на большинстве пользовательских компьютеров.

1. Технические требования

NET Framework версии не ниже 4.6

1. ****Описание классов****

Проект игры «Сапёр» представляет собой не только основной игровой процесс известной игры. Данная реализация представляет собой «свежий взгляд» на классику. В данном проекте присутствует множество экспериментальных решений, которые понравились пользователям, которые тестировали проект на всем пути его создания. В проекте представлено множество классов. Среди них есть такие, как: Achievments.xaml.cs – окно для отображения «достижений», которые получил пользователь, за все время своей игры; CharacterSelection.xaml.cs – пример экспериментальной функции, которая предоставляет игроку возможность выбора игрового персонажа (набор пассивных бонусов), а также прокачки и покупки новых персонажей; MainWindow.xaml.cs – основной класс приложения, который сосредоточил в себе почти все основные механизмы воспроизведения игры «Сапер»; ThemeManager.cs – класс, который позволяет переключать темы в приложении; patchNotes.xaml.cs –позволяет просмотреть историю обновлений проекта.

В основном классе проекта MainWindow.xaml.cs сосредоточено огромное количество различных методов, в данном документе будут описаны только несколько основных, которые являются краеугольными в плане работы самого приложения.

PlaceMines – расставляет мины по игровому полю;

CalculateAdjacentMines – просчитывает количество мин в соседних плитках для каждой безопасной плитки;

CreateButtons – заполняет специализированный элемент UniformGrid кнопками, которые представляют собой игровое поле;

CreateButtons – создает кнопку на каждой плитке игрового поля;

Button\_RMB – метод-событие для нажатия правой кнопкой мыши по любой кнопке-плитке на игровом поле, отвечает за установку флажка, а также реализует событие нажатия на плитку обоими клавишами мыши;

Button\_Click - метод-событие для нажатия левой кнопкой мыши по любой кнопке-плитке на игровом поле, отвечает за раскрытие плитки, а также реализует событие нажатия на плитку обоими клавишами мыши;

tryToReveal – рекурсивный метод, отвечающий за раскрытие плитки и в случае нахождения рядом плиток, которые могут быть раскрыты (не имеют мин в соседних плитках), рекурсивно раскрывает их;

sonar – метод срабатывающий при нажатии обоими клавишами мыши по одной и той же раскрытой плитке, у которой есть соседние нераскрытые мины. Подсвечивает область вокруг плитки – инициатора. Если в ней правильно расставлены флажки, то раскрывает остальные плитки;

clearSonar – метод отвечающий за отключение подсветки предыдущего метода.

checkForWin – метод который просчитывает количество отмеченных флажком мин и количество раскрытых безопасных плиток. В случае если одна из проверок удовлетворяет условиям для победы, то вызывается метод окончания игры.

EndGame – метод, который отвечает за окончание игры, подсчет результатов, раскрытие всего оставшегося поля. Принимает буллеву переменную, от значения которой зависит результат игры.

1. ****Алгоритмы****
   1. Алгоритм заполнения поля

При нажатии кнопки «Начать игру» запускается алгоритм PlaceMines, который заполняет двумерный массив, состоящий из переменных типа bool, который отвечает за наличие в плитке мины. Размеры массива и количество мин зависит от выбранной пользователем сложности.

После генерации мин вызывается метод CalculateAdjacentMines, который заполняет двумерный массив, состоящий из численных переменных, которые отвечают за количество соседних мин для безопасных плиток.

Последним действием алгоритма является метод CreateButtons, который создает кнопки для каждой плитки, чтобы пользователь мог взаимодействовать с игровым полем.

* 1. Алгоритм открытия пустых ячеек

Если игрок открывает клетку, рядом с которой нет мин, алгоритм автоматически раскрывает все смежные пустые клетки с помощью рекурсивного метода. Метод раскрытия вызывает сам себя, передавая координаты текущей плитки. Каждая проверенная клетка добавляется в двумерный массив проверенных плиток, чтобы избежать повторного срабатывания алгоритма на уже раскрытую плитку.

При открытии клетки её состояние меняется со «скрытой» на «открытую», а если рядом есть мины, отображается их количество. Плитки с соседними минами и полностью пустые плитки окрашиваются в разный цвет, для упрощения визуального восприятия пользователем. Такой подход позволяет игроку быстро исследовать безопасные зоны, не выполняя лишних кликов, и визуально упрощает анализ поля.

Алгоритм также гарантирует, что открываются только те клетки, которые не содержат мин и не помечены флажками. Это предотвращает случайное открытие опасных участков и сохраняет стратегическую составляющую игры. Благодаря этому механизму «Сапер» остаётся логической игрой, где важны анализ и расчёт, а не случайные действия.

* 1. Алгоритм проверки победы пользователя

Этот алгоритм основан на простом, но действенном принципе: игрок считается победившим, когда все клетки без мин открыты, а все мины правильно отмечены флажками. Для этого система подсчитывает количество оставшихся закрытых клеток и сравнивает его с общим числом мин на поле. Если эти показатели совпадают, значит, все оставшиеся закрытые клетки — это мины, и игра считается выигранной.

Работа алгоритма следующая: сначала он собирает координаты всех клеток в список. Затем в цикле проверяет состояние каждой клетки. Если клетка закрыта, увеличивается счетчик таких клеток. После проверки всего поля, если число закрытых клеток равно количеству мин, запускается процедура завершения игры: останавливается таймер, фиксируется лучшее время (если оно рекордное), и выводится поздравительное сообщение.

Особенность этого метода в том, что он не требует от игрока обязательного расставления всех флажков. Победа наступает автоматически, как только все безопасные клетки открыты, даже если некоторые мины не отмечены. Это делает игровой процесс более гибким и уменьшает раздражение от необходимости точного подсчета мин в конце игры. Тем не менее многие игроки предпочитают заранее помечать все мины флажками перед открытием последней безопасной клетки для тактического удовольствия.

Алгоритм хорошо интегрирован в основной цикл игры: проверка победы выполняется после каждого действия — открытия клетки или установки флажка — что обеспечивает мгновенное определение победы без задержек. Также система исключает ложные срабатывания благодаря строгой проверке состояния клеток, что гарантирует надежность даже при сложных сценариях (например, если игрок случайно поставил флажок на безопасную клетку). В результате алгоритм сочетает простоту реализации с точностью определения условий победы, что соответствует принципам качественного геймдизайна в логических играх. На рисунке 2 показана блок-схема алгоритмов игры «Сапер».

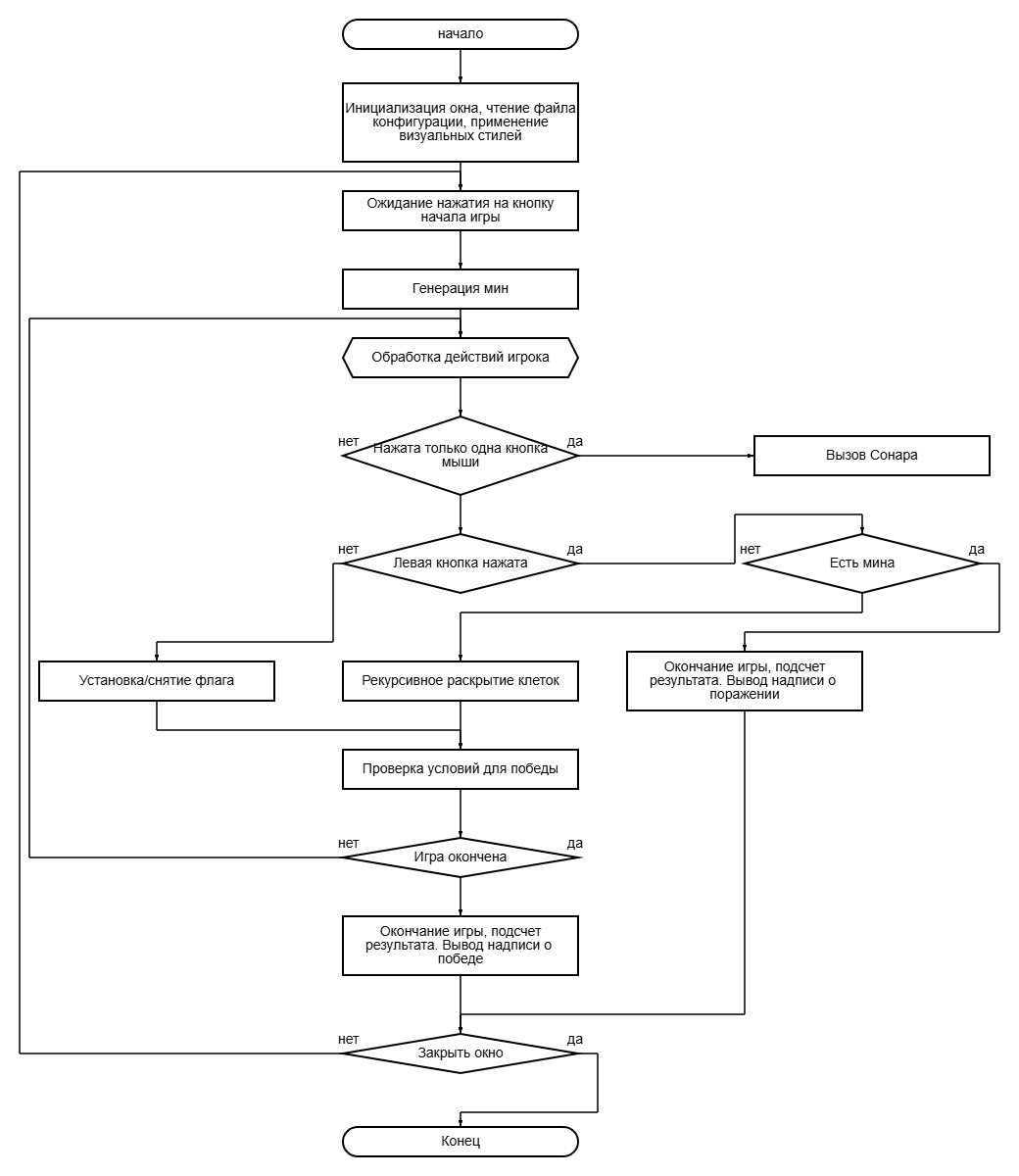


Рисунок 1 - Блок-схема алгоритмов игры «Сапер»

1. ****Тестирование нагрузочное, тестирование на безопасность****

тестирование игры «Сапер» проводилось с целью проверить стабильность работы приложения при длительной эксплуатации и большом объеме пользовательских действий. Основное внимание уделялось оценке производительности алгоритмов генерации игрового поля, обработки кликов и автоматического открытия пустых участков. Особое значение имело тестирование рекурсивных методов (например, при открытии пустых клеток), чтобы исключить риск переполнения стека при обработке больших пустых зон. Результаты показали, что игра остается стабильной даже в условиях высокой нагрузки благодаря использованию оптимизированных структур данных и отсутствию тяжелых вычислений в основном цикле.

Тестирование на безопасность было направлено на выявление возможных уязвимостей, которые могли бы привести к сбоям или нарушению логики игры. Проверялась правильность обработки крайних случаев: попытки открыть уже открытые клетки или установить флажки на мины. Также анализировалась работа таймера и системы подсчета очков на предмет возможных манипуляций, таких как сброс времени или искусственное завышение результата. Важным аспектом было удостовериться в целостности данных: убедиться, что состояние клеток (наличие флагов, мин) не может быть изменено внешними воздействиями или ошибочными вызовами методов.

Дополнительно проводилось стресс-тестирование интерфейса, моделирующее интенсивное взаимодействие пользователя с приложением. В рамках этого проверялись быстрые последовательные клики и двойные нажатия по одной и той же клетке. В результате удалось обеспечить высокий уровень отказоустойчивости при сохранении плавности игрового процесса.

Таблица 1 - Результаты тестирования игры «Сапер»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№ теста** | **Описание** | **Ожидаемая реакция** | **Фактическая реакция** | **Вывод** |
| **1** | Двойной клик на плитку, которая помечена флагом | Ничего не произойдет | Ничего не произошло | Успешно |
| **2** | Двойной клик на плитку, которая не раскрыта | Ничего не произойдет | Ничего не произошло | Успешно |
| **3** | Раскрытие всех безопасных плиток, ни одна мина не отмечена | Игроку будет засчитана победа из-за раскрытия всех безопасных плиток | Игра окончилась победой игрока | Успешно |
| **4** | Отметка всех мин флагом, ни одна безопасная клетка не раскрыта | Игроку будет засчитана победа из-за отметки всех мин на поле | Игра окончилась победой игрока | Успешно |
| **5** | Левый клик по плитке с флагом | Ничего не произойдет | Ничего не произошло | Успешно |
| **6** | Двойной клик по клетке, имеющей соседние мины, но все соседние клетки будут помечены флагом. Таким образом количество флагов превысит количество соседних мин | Ничего не произойдет, клетки не раскроются, проигрыш не произойдет, так как отмечено мин больше, чем существует по соседству | Ничего не произошло | Успешно |

1. ****Инструкция пользователя****

После запуска приложения откроется игровое поле, состоящее из клеток, а также панель управления с таймером, счетчиком оставшихся мин и кнопкой для перезапуска. Игра начинается с первого клика — нужно нажать на любую клетку, чтобы открыть её. После этого таймер начнёт отсчет времени.

Основные действия:

Левый клик — открыть выбранную клетку. Если под ней находится мина, игра завершится поражением. Если клетка пуста, на ней появится число — количество мин в соседних клетках.

Правый клик — поставить или снять флажок, обозначая предполагаемые мины. Количество оставшихся мин отображается на панели управления.

Клик обоими клавишами – подсветить соседние клетки, если в них корректно установлены флаги, то произойдет раскрытие остальных соседних клеток, иначе произойдет поражение.

Автоматическое открытие — если вокруг открытой клетки нет мин, автоматически откроются все соседние пустые клетки.

Задача — открыть все клетки без мин. Для этого нужно анализировать числа на открытых клетках: они показывают, сколько мин расположено в соседних ячейках. Также используют флажки для отметки предполагаемых мин и избегают случайных кликов по опасным клеткам. Например, если на клетке стоит число «3», и вокруг неё уже поставлены три флажка, значит остальные соседние клетки безопасны.

Игра завершается в двух случаях: победа — когда все безопасные клетки открыты, а мины либо помечены флажками, либо остались закрытыми; или поражение — если откроется клетка с миной. В этом случае все мины станут видимыми, и игра автоматически завершится.

Для начала новой игры нужно нажать кнопку «Начать заново» на панели управления.

****Заключение****

В ходе выполнения курсовой работы была разработана классическая игра «Сапер» с использованием объектно-ориентированного подхода на языке C#. Проект реализован без использования внешних библиотек, только используя встроенную функцию XAML разметки. В процессе разработки были успешно решены ключевые задачи: проектирование архитектуры приложения, реализация игровой логики, обработка пользовательского ввода, визуализация игрового процесса и работа с файлом конфигурации.

В работе особое внимание уделялось соблюдению принципов объектно-ориентированного программирования. Классы и вспомогательные компоненты были организованы таким образом, чтобы четко разделять ответственность: одни отвечают за интерфейс и обработку пользовательских событий, другие — за логику игрового поля, а утилитные классы обеспечивают повторное использование кода. Особенно стоит отметить реализацию алгоритмов автоматического открытия пустых клеток с помощью поиска в глубину и проверку условий победы — эти решения ярко иллюстрируют применение рекурсии и методов обхода данных в контексте разработки игры.

Тестирование проекта происходило по ходу разработки двумя независимыми тестировщиками. Правки и исправления вносились на основе обратной связи. Все исправления, как ошибок, так и баланса можно отследить в окне «Обновления».

Выполненная работа не только закрепила теоретические знания по объектно-ориентированному программированию, но и предоставила ценный практический опыт в создании полноценного приложения. Игра «Сапер» показывает, как сочетание четкой архитектуры, продуманных алгоритмов и внимания к пользовательскому интерфейсу способствует созданию качественного программного продукта даже в рамках учебного проекта. Полученные навыки проектирования классов, обработки исключений и оптимизации производительности станут полезными при решении более сложных задач в будущем.

В заключение нужно отметить то, что данный проект не только достиг всех изначальных целей, но и позволил внести новое видение в классическую игру, расширив ее горизонты, путем использования многих нестандартных решений. Разработанный проект совмещает в себе и классическую составляющую всем известной игры «Сапер», но и обладает современными решениями, которые затрагивают многие аспекты ООП и могут служить прекрасным обучающим материалом для изучения C# и XAML, а также и самого объектно-ориентированного программирования.

Список использованных источников и литературы

1. Документация по языку C#, информационный ресурс 2025 г., URL: https://learn.microsoft.com/ru-ru/dotnet/csharp/
2. XAML - C# и WPF, информационный ресурс 2016 г., URL: https://metanit.com/sharp/wpf/2.php
3. Minesweeper Game in C#, информационный ресурс 2021 г., URL: https://www.codeproject.com/Articles/5293335/Minesweeper-Game-in-Csharp
4. Документация по Visual Studio, информационный ресурс 2022 г., URL: https://learn.microsoft.com/ru-ru/visualstudio/windows/?view=vs-2022

Приложение А

private void PlaceMines()

{

Random random = new Random();

int minesPlaced = 0;

while (minesPlaced < mineCount)

{

int row = random.Next(rows);

int col = random.Next(columns);

if (!minefield[row, col])

{

minefield[row, col] = true;

minesPlaced++;

}

}

}

private void CalculateAdjacentMines()

{

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

{

for (int col = 0; col < columns; col++)

{

if (!minefield[row, col])

{

int count = 0;

// Check adjacent cells

for (int i = -1; i <= 1; i++)

{

for (int j = -1; j <= 1; j++)

{

if (i == 0 && j == 0) continue; // Skip the current cell

int adjRow = row + i;

int adjCol = col + j;

if (adjRow >= 0 && adjRow < rows && adjCol >= 0 && adjCol < columns && minefield[adjRow, adjCol])

{

count++;

}

}

}

adjacentMines[row, col] = count;

}

}

}

}

private void CreateButtons()

{

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

{

for (int col = 0; col < columns; col++)

{

Button button = new Button();

button.Content = ""; // Initially hidden

button.Tag = new Tuple<int, int>(row, col); // Store row/col information in the button's Tag

button.Click += Button\_Click;

button.MouseRightButtonDown += Button\_RMB;

button.Style = new Style(typeof(Button), buttonStyle) { Setters = { new Setter(Button.BackgroundProperty, (Brush)buttonBackgroundBrush), new Setter(Button.ForegroundProperty, (Brush)buttonBackgroundBrush) } };

buttons[row, col] = button;

GameGrid.Children.Add(button);

button.UpdateLayout();

}

}

ThemeButtonFromWindows();

//App.setGlowEffect(Settings.Default.Theme);

}

private void Button\_RMB(object sender, MouseEventArgs e)

{

if (!gameOver)

{

clearSonar();

Button button = (Button)sender;

Tuple<int, int> position = (Tuple<int, int>)button.Tag;

int row = position.Item1;

int col = position.Item2;

//lets try. FUCKING WORKED

if (!doubleDown)

{

if (!btnRight)

{

doubleDown = true;

btnRight = true;

buffButton = button;

disableTimer.Tag = button;

bufMEA = e;

boostX = row;

boostY = col;

disableTimer.Start();

}

else

{

CheckForWin();

if (flagged[row,col])

{

if (minefield[row,col])

{

if (!shopRevealed[row,col]) openedMineCount--;

score -= 20;

}

else

{

score += 15;

}

flagged[row, col] = false;

button.Content = "";

if (ThemeManager.CurrentTheme == "CyberpunkTheme" && cbGlow) button.Style = buttonGlowStyle;

else button.Style = new Style(typeof(Button), buttonStyle) { Setters = { new Setter(Button.BackgroundProperty, (Brush)buttonBackgroundBrush) } };

minesLeft.Content = (Convert.ToInt32(minesLeft.Content) + 1).ToString();

flagged[row, col] = false;

}

else

{

if (!revealed[row, col])

{

if (minefield[row, col])

{

score += 20;

if (!shopRevealed[row,col])openedMineCount++;

CheckForWin();

}

else

{

score -= 20;

}

minesLeft.Content = (Convert.ToInt32(minesLeft.Content) - 1).ToString();

Image flag = new Image();

BitmapImage bitmap = new BitmapImage(new Uri("pack://application:,,,/saperdun;component/Images/333.png"));

flag.Source = bitmap;

flag.Width = 20;

flag.Height = 20;

button.Content = flag;

button.Style = new Style(typeof(Button), buttonStyle) { Setters = { new Setter(Button.BackgroundProperty, (Brush)buttonFlaggedBackgroundBrush) } };

flagged[row, col] = true;

}

}

btnRight = false;

}

}

else

{

//some double click code

if(buffButton==button)

{

clearSonar();

boostX = row;

boostY = col;

if (revealed[row, col])

{

int chance = rnd.Next(100);

if (evgen\_player&& chance <= evgen\_chance \* evgen\_level && !evgenTries[row,col])

{

openCloseSafeTask();

evgenTries[row, col] = true;

shoppoints += openCloseSafePrice;

shoppointL.Content = shoppoints;

}

else

{

sonar(row, col);

}

}

doubleDown = false;

disableTimer.Stop();

btnLeft = false;

btnRight = false;

}

}

CheckForWin();

}

}

private void Button\_Click(object sender, RoutedEventArgs e)

{

if (gameOver) return; // Don't allow clicks after the game is over

clearSonar();

Button button = (Button)sender;

Tuple<int, int> position = (Tuple<int, int>)button.Tag;

int row = position.Item1;

int col = position.Item2;

if (!doubleDown)

{

if (!btnLeft)

{

doubleDown = true;

btnLeft = true;

buffButton = button;

disableTimer.Tag = button;

bufREA = e;

boostX = row;

boostY = col;

disableTimer.Start();

}

else

{

steps++;

if (minefield[row, col])

{

if (!flagged[row, col])

{

int chance = rnd.Next(100);

if (zakhar\_player&& chance <= zakhar\_chance\*zakhar\_level)

{

minesLeft.Content = (Convert.ToInt32(minesLeft.Content) - 1).ToString();

Image flag = new Image();

BitmapImage bitmap = new BitmapImage(new Uri("pack://application:,,,/saperdun;component/Images/333.png"));

flag.Source = bitmap;

flag.Width = 20;

flag.Height = 20;

button.Content = flag;

button.Style = new Style(typeof(Button), buttonStyle) { Setters = { new Setter(Button.BackgroundProperty, (Brush)buttonFlaggedBackgroundBrush) } };

flagged[row, col] = true;

if(zakhar\_level==5)

{

int chance3 = rnd.Next(100);

if (chance3 <= 50)

{

openRandomMineLabel\_Click(null, null);

shoppoints += 250;

shoppointL.Content = shoppoints;

}

else

{

openRandomSafeLabel\_Click(null, null);

shoppoints += 150;

shoppointL.Content = shoppoints;

}

}

}

else

{

// Game Over - Mine Hit!

Image bomb = new Image();

BitmapImage bitmap = new BitmapImage(new Uri("pack://application:,,,/saperdun;component/Images/bimba.png"));

bomb.Source = bitmap;

bomb.Width = 20;

bomb.Height = 20;

button.Content = bomb;

button.Style = new Style(typeof(Button), buttonStyle) { Setters = { new Setter(Button.BackgroundProperty, (Brush)buttonMineBackgroundBrush) } };

if (needToCountShoppoints) EndGame(false);

}

}

}

else

{

// Display Adjacent Mines Count

if (!flagged[row,col])

{

tryToReveal(row, col);

CheckForWin();

}

}

btnLeft = false;

}

}

else

{

//some double click code

if(buffButton==button)

{

clearSonar();

boostX = row;

boostY = col;

if (revealed[row, col])

{

int chance = rnd.Next(100);

if (evgen\_player && chance <= evgen\_chance \* evgen\_level && !evgenTries[row, col])

{

openCloseSafeTask();

evgenTries[row, col] = true;

shoppoints += openCloseSafePrice;

shoppointL.Content = shoppoints;

}

else

{

sonar(row, col);

}

}

doubleDown = false;

disableTimer.Stop();

btnLeft = false;

btnRight = false;

}

}

CheckForWin();

}

private void tryToReveal(int row, int col)

{

if (row < 0 || row >= rows || col < 0 || col >= columns) return;

if (revealed[row, col]) return;

if (minefield[row, col]) return;

if (adjacentMines[row, col] > 0)

{

if (buttons[row, col].Content.ToString() == "!")

{

minesLeft.Content = (Convert.ToInt32(minesLeft.Content) + 1).ToString();

}

buttons[row, col].Content = adjacentMines[row, col].ToString();

buttons[row, col].Style = new Style(typeof(Button), buttonStyle) { Setters = { new Setter(Button.ForegroundProperty, (Brush)buttonForegroundBrush) } };

buttons[row, col].Style = new Style(typeof(Button), buttonStyle) { Setters = { new Setter(Button.BackgroundProperty, (Brush)buttonAdjacentBackgroundBrush) } };

revealed[row, col] = true;

if(!countNoMoreScore)score += 5;

CheckForWin();

return;

}

if (buttons[row, col].Content.ToString() == "!")

{

minesLeft.Content = (Convert.ToInt32(minesLeft.Content) + 1).ToString();

}

buttons[row, col].Content = "";

buttons[row, col].IsEnabled = false;

buttons[row, col].Style = new Style(typeof(Button), buttonStyle) { Setters = { new Setter(Button.BackgroundProperty, (Brush)buttonRevealedBackgroundBrush) } };

if(!countNoMoreScore)score += 1;

revealed[row, col] = true;

for (int i = -1; i <= 1; i++)

{

for (int j = -1; j <= 1; j++)

{

if (i == 0 && j == 0) continue;

tryToReveal(row + i, col + j);

}

}

CheckForWin();

}

private void sonar(int row,int col)

{

int countFlagged = 0;

int countAllFlagged = 0;

for(int i = row-1; i<=row+1;i++)

{

for(int j = col-1; j<=col+1;j++)

{

if (i >= 0&& i< rows && j>=0 &&j<columns)

{

if ((i != row || j != col) && !revealed[i, j] && !flagged[i,j])

{

buttons[i, j].Style = new Style(typeof(Button), buttonStyle) { Setters = { new Setter(Button.BackgroundProperty, (Brush)buttonSonarBackgroundBrush) } };

bufRevx = row;

bufRevy = col;

sonarWorked = true;

}

}

}

}

for (int i = row - 1; i <= row + 1; i++)

{

for (int j = col - 1; j <= col + 1; j++)

{

if (i >= 0 && i < rows && j >= 0 && j < columns)

{

if ((i != row || j != col) && (minefield[i,j]&&flagged[i,j]))

{

countFlagged++;

}

if ((i != row || j != col) && (flagged[i, j]))

{

countAllFlagged++;

}

}

}

}

if (countAllFlagged == adjacentMines[row,col])

{

if (adjacentMines[row, col] == countFlagged)

{

for (int i = row - 1; i <= row + 1; i++)

{

for (int j = col - 1; j <= col + 1; j++)

{

if (i >= 0 && i < rows && j >= 0 && j < columns)

{

if ((i != row || j != col) && !revealed[i, j] && !flagged[i, j])

{

tryToReveal(i, j);

}

}

}

}

}

else

{

for (int i = row - 1; i <= row + 1; i++)

{

for (int j = col - 1; j <= col + 1; j++)

{

if (i >= 0 && i < rows && j >= 0 && j < columns)

{

if ((i != row || j != col) && minefield[i, j] && !flagged[i, j])

{

// Game Over - Mine Hit!

Image bomb = new Image();

BitmapImage bitmap = new BitmapImage(new Uri("pack://application:,,,/saperdun;component/Images/bimba.png"));

bomb.Source = bitmap;

bomb.Width = 20;

bomb.Height = 20;

buttons[i, j].Content = bomb;

buttons[i, j].Style = new Style(typeof(Button), buttonStyle) { Setters = { new Setter(Button.BackgroundProperty, (Brush)buttonMineBackgroundBrush) } };

if(needToCountShoppoints)EndGame(false);

break;

}

}

}

}

}

}

else if (countAllFlagged > adjacentMines[row,col])

{

for (int i = row - 1; i <= row + 1; i++)

{

for (int j = col - 1; j <= col + 1; j++)

{

if (i >= 0 && i < rows && j >= 0 && j < columns)

{

if ((i != row || j != col) && minefield[i, j] && !flagged[i, j])

{

// Game Over - Mine Hit!

Image bomb = new Image();

BitmapImage bitmap = new BitmapImage(new Uri("pack://application:,,,/saperdun;component/Images/bimba.png"));

bomb.Source = bitmap;

bomb.Width = 20;

bomb.Height = 20;

buttons[i, j].Content = bomb;

buttons[i, j].Style = new Style(typeof(Button), buttonStyle) { Setters = { new Setter(Button.BackgroundProperty, (Brush)buttonMineBackgroundBrush) } };

if(needToCountShoppoints)EndGame(false);

break;

}

}

}

}

}

CheckForWin();

}

private void clearSonar()

{

if(sonarWorked)

{

int row = bufRevx;

int col = bufRevy;

for (int i = row - 1; i <= row + 1; i++)

{

for (int j = col - 1; j <= col + 1; j++)

{

if (i >= 0 && i < rows && j >= 0 && j < columns)

{

if ((i != row || j != col) && !revealed[i,j] && !flagged[i, j])

{

if (ThemeManager.CurrentTheme == "CyberpunkTheme" && cbGlow) buttons[i,j].Style = buttonGlowStyle;

else buttons[i, j].Style = new Style(typeof(Button), buttonStyle) { Setters = { new Setter(Button.BackgroundProperty, (Brush)buttonBackgroundBrush) } };

}

}

}

}

sonarWorked = !sonarWorked;

}

}

private async void EndGame(bool win)

{

if (worker\_player)

{

score += (score / 100) \* 3;

}

if(leha\_player)

{

int chance = rnd.Next(0-leha\_chance,leha\_chance);

if(chance+leha\_level <leha\_chance)

{

score += (score / 100) \* chance + leha\_level;

}

else if(chance+leha\_level>leha\_chance)

{

score += (score / 100) \* chance + leha\_level;

}

}

speedGameToggleButton.IsEnabled = true;

time.Content = elapsedTime.ToString(@"mm\:ss");

ThemeButtonFromWindows();

if (win)

{

dayLoginCheck();

TimeSpan ts2 = TimeSpan.ParseExact(elapsedTime.ToString(@"mm\:ss"), "mm\\:ss", null);

TimeSpan ts1 = TimeSpan.ParseExact("00:01", "mm\\:ss", null); ;

switch (selectedDiff)

{

case difficulties.beginner:

ts1 = TimeSpan.ParseExact("00:30", "mm\\:ss", null);

break;

case difficulties.intermediate:

ts1 = TimeSpan.ParseExact("02:00", "mm\\:ss", null);

break;

case difficulties.expert:

ts1 = TimeSpan.ParseExact("03:30", "mm\\:ss", null);

break;

case difficulties.professional:

ts1 = TimeSpan.ParseExact("09:00", "mm\\:ss", null);

break;

}

if(ts2<ts1)

{

checkAchievments(achievments.speedster);

}

gameResult = "victory";

switch (selectedDiff)

{

case difficulties.beginner:

if (speedBoost)

{

int mins = Convert.ToInt32(elapsedTime.ToString(@"mm\:ss").Split(':')[0]);

int secs= Convert.ToInt32(elapsedTime.ToString(@"mm\:ss").Split(':')[1]);

secs += mins \* 60;

if(secs<=60)

{

float result = (float)secs / 60;

float timeBooster = (float)1-result;

score += Convert.ToInt32(score \* timeBooster);

Console.WriteLine(score);

scrap\_count += 2;

powder\_count += 3;

}

else if(secs<=90)

{

float result = (float)secs / 90;

float timeBooster = (float)1 - result;

score = Convert.ToInt32(score \* timeBooster);

Console.WriteLine(score);

}

else

{

score = 0;

}

}

shoppoints += score / 4;

shoppoints += passiveBonus;

checkAchievments(achievments.newJourney);

scrap\_count += rnd.Next(3,7);

powder\_count += rnd.Next(7,13);

if(leha\_player&&leha\_level==5)

{

scrap\_count++;

powder\_count += 2;

}

break;

case difficulties.intermediate:

if(speedBoost)

{

int mins = Convert.ToInt32(elapsedTime.ToString(@"mm\:ss").Split(':')[0]);

int secs = Convert.ToInt32(elapsedTime.ToString(@"mm\:ss").Split(':')[1]);

secs += mins \* 60;

if (secs <= 180)

{

float result = (float)secs / 180;

float timeBooster = (float)1 - result;

score += Convert.ToInt32(score \* timeBooster);

Console.WriteLine(score);

scrap\_count += 4;

powder\_count +=6;

flags\_count += 1;

checkAchievments(achievments.worthy);

}

else if (secs <= 240)

{

float result = (float)secs / 240;

float timeBooster = (float)1 - result;

score = Convert.ToInt32(score \* timeBooster);

Console.WriteLine(score);

}

else

{

score = 0;

}

}

shoppoints += score / 3;

shoppoints += passiveBonus;

scrap\_count += rnd.Next(6, 11);

powder\_count += rnd.Next(10, 17);

checkAchievments(achievments.localLegend);

if (leha\_player && leha\_level == 5)

{

scrap\_count+=3;

powder\_count += 5;

}

break;

case difficulties.expert:

if(speedBoost)

{

int mins = Convert.ToInt32(elapsedTime.ToString(@"mm\:ss").Split(':')[0]);

int secs = Convert.ToInt32(elapsedTime.ToString(@"mm\:ss").Split(':')[1]);

secs += mins \* 60;

if (secs <= 300)

{

float result = (float)secs / 300;

float timeBooster = (float)1 - result;

score += Convert.ToInt32(score \* timeBooster);

Console.WriteLine(score);

scrap\_count += 5;

powder\_count += 7;

flags\_count += 1;

if (leha\_player && leha\_level == 5)

{

flags\_count++;

}

}

else if (secs <= 360)

{

float result = (float)secs / 360;

float timeBooster = (float)1 - result;

score = Convert.ToInt32(score \* timeBooster);

Console.WriteLine(score);

}

else

{

score = 0;

}

}

shoppoints += score / 2;

scrap\_count += rnd.Next(16, 24);

powder\_count += rnd.Next(16, 30);

shoppoints += passiveBonus;

flags\_count += 1;

checkAchievments(achievments.heroOfTheKingdom);

if (leha\_player && leha\_level == 5)

{

scrap\_count += 4;

powder\_count += 9;

}

break;

case difficulties.professional:

if(speedBoost)

{

int mins = Convert.ToInt32(elapsedTime.ToString(@"mm\:ss").Split(':')[0]);

int secs = Convert.ToInt32(elapsedTime.ToString(@"mm\:ss").Split(':')[1]);

secs += mins \* 60;

if (secs <= 630)

{

float result = (float)secs / 630;

float timeBooster = (float)1 - result;

score += Convert.ToInt32(score \* timeBooster);

Console.WriteLine(score);

scrap\_count += 8;

powder\_count += 10;

flags\_count += 2;

if (leha\_player && leha\_level == 5)

{

scrap\_count += 5;

powder\_count += 14;

}

}

else if (secs <= 720)

{

float result = (float)secs / 720;

float timeBooster = (float)1 - result;

score = Convert.ToInt32(score \* timeBooster);

Console.WriteLine(score);

}

else

{

score = 0;

}

}

shoppoints += score;

shoppoints += passiveBonus;

scrap\_count += rnd.Next(25, 35);

powder\_count += rnd.Next(32, 47);

flags\_count += 2;

if (leha\_player && leha\_level == 5)

{

scrap\_count += 9;

powder\_count += 18;

flags\_count++;

}

checkAchievments(achievments.hopeOfHumanity);

if (!bonusCodesStr.Contains("киберпсих"))

{

MSGBox.Show("Доступен новый бонус-код! \n\"киберпсих\"");

}

break;

case difficulties.custom:

scoreLabel.Content = 1000;

score = 1000;

shoppoints += passiveBonus;

break;

}

scrapTxt.Text = "Част.: "+scrap\_count;

powderTxt.Text = "Пыл.: " + powder\_count;

flagsTxt.Text = "Флаж.: " + flags\_count;

}

else

{

if (autoLose) score = 0;

switch (selectedDiff)

{

case difficulties.beginner:

if (steps <= 3) gameResult = "draw";

else gameResult = "defeat";

shoppoints += score / 8;

if (leha\_player && leha\_level == 5 && gameResult == "defeat")

{

scrap\_count += rnd.Next(1,3);

powder\_count += rnd.Next(2,4);

}

break;

case difficulties.intermediate:

if (steps <= 5) gameResult = "draw";

else gameResult = "defeat";

shoppoints += score / 6;

if (leha\_player && leha\_level == 5 && gameResult == "defeat")

{

scrap\_count += rnd.Next(2, 4);

powder\_count += rnd.Next(3, 5);

}

break;

case difficulties.expert:

if (steps <= 7) gameResult = "draw";

else gameResult = "defeat";

shoppoints += score / 4;

if (leha\_player && leha\_level == 5&&gameResult=="defeat")

{

scrap\_count += rnd.Next(3, 5);

powder\_count += rnd.Next(4, 6);

}

break;

case difficulties.professional:

if (steps <= 10) gameResult = "draw";

else gameResult = "defeat";

shoppoints += score / 2;

if (leha\_player && leha\_level == 5 && gameResult == "defeat")

{

scrap\_count += rnd.Next(4, 6);

powder\_count += rnd.Next(5, 7);

}

break;

case difficulties.custom:

shoppoints += 0;

break;

}

}

if(leha\_player&&leha\_level==5&&gameResult!="draw")

{

int chance = rnd.Next(100);

if(chance <=15)

{

flags\_count++;

}

}

scanBoostActive = false;

shoppointL.Content = shoppoints;

scoreBoardUpdate();

scoreNeedToWrite = false;

gameOver = true;

if (win)

{

timer.Stop();

minesLeft.Visibility = Visibility.Collapsed;

minesLeftLabel.Visibility = Visibility.Collapsed;

scoreLabel.Foreground = Brushes.Green;

scoreLabel.Background = Brushes.LightGreen;

scoreLabel.Content = score.ToString()+" победа";

}

else

{

timer.Stop();

try

{

btnStart.IsEnabled = false;

await revealAllMines();

}

finally

{

btnStart.IsEnabled = true;

minesLeft.Visibility = Visibility.Collapsed;

minesLeftLabel.Visibility = Visibility.Collapsed;

scoreLabel.Foreground = Brushes.Red;

scoreLabel.Background = Brushes.Pink;

scoreLabel.Content = score.ToString() + " поражение";

}

}

if (!achievmentsStr.Contains("worthy"))

{

scrap\_count = 0;

powder\_count = 0;

flags\_count = 0;

}

startButton.Visibility = Visibility.Visible;

startButton.IsEnabled = true;

openRandomMineLabel.IsEnabled = false;

openRandomMineLabel.Style = new Style(typeof(Button), buttonStyle) { Setters = { new Setter(Button.BackgroundProperty, (Brush)buttonRevealedBackgroundBrush) } };

openRandomSafeLabel.Style = new Style(typeof(Button), buttonStyle) { Setters = { new Setter(Button.BackgroundProperty, (Brush)buttonRevealedBackgroundBrush) } };

openCloseSafeLabel.Style = new Style(typeof(Button), buttonStyle) { Setters = { new Setter(Button.BackgroundProperty, (Brush)buttonRevealedBackgroundBrush) } };

openCloseSafeLabel.IsEnabled = false;

openRandomSafeLabel.IsEnabled = false;

needToCountShoppoints = false;

}

private void CheckForWin()

{

int revealedCount = 0;

openedTileCount = 0;

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

{

for (int col = 0; col < columns; col++)

{

if (revealed[row, col])

{

revealedCount++;

openedTileCount++;

}

}

}

int countAllFlagged = 0;

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

{

for (int coll = 0; coll < columns; coll++)

{

if (minefield[roww, coll] && flagged[roww, coll])

{

countAllFlagged++;

}

}

}

if((revealedCount)==(rows\*columns)-mineCount)

{

if(needToCountShoppoints)

{

score += (mineCount - countAllFlagged) \* 20;

EndGame(true);

}

}

}