Министерство образования Республики Беларусь

Учреждение образования

БелорусскиЙ государственный университет

информатики и радиоэлектроники

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему

**Программное игровое средство «САПЁР»**

|  |  |  |
| --- | --- | --- |
| Студент |  | А. Ю. Белявский |
| Руководитель |  | Д. С. Шулицкий |

Минск 2019

**ВВЕДЕНИЕ**

Игра "Сапер" появилась в Windows тоже не просто так. Как сообщает компания, эта игра была нужна для того, чтобы научить игроков пользоваться двумя кнопками мыши. А поиск заминированных клеток оказался самым простым способом донести до пользователей разницу между правой и левой клавишей.

* 1. Анализ предметной области
  2. **Обзор аналогов**

Существуют варианты игры с полем и/или ячейками непрямоугольной формы, в трёхмерном пространстве, многопользовательские варианты. В некоторых разновидностях цифры, обозначающие количество заминированных ячеек, различаются по цвету. В стандартных вариантах «Сапёра» и «Mines» цифра 1 — синяя, 2 — зелёная, 3 — красная, 4 — тёмно-синяя.

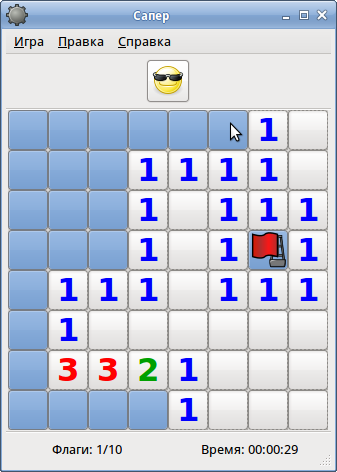
При некоторых расстановках мин вычислить их расположение невозможно, и игрок оказывается перед необходимостью открывать ячейки наугад. Сапёр для Windows имеет интерактивное поле, если все возможные комбинации на поле уже открыты, то взрыва не последует даже при нажатии наугад на любой неоткрытый участок поля.

**1.1.1** «Сапёр» (стандартная игра ОС Windows)

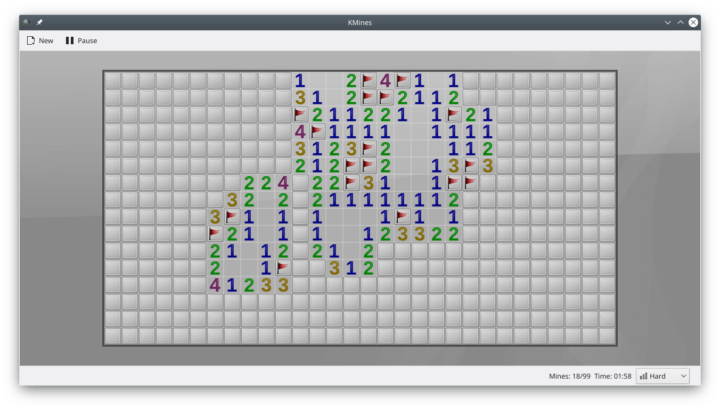


**1.1.2** «Mines» (набор игр от GNOME Games)

Существует современная реализация концепции «Сапёра», представленная в приложении «Mines» из набора игр GNOME Games. Особенностями данного проекта являются новый дизайн и адаптация под современные операционные системы. К достоинствам данного программного продукта также следует отнести поддержку приложения разработчиками, о чем свидетельствуют регулярные обновления, выпускаемые студией. Актуальной версией на сегодняшний день является «Mines 3.12». Внешний вид окна данного приложения представлен на рисунке 1.2.



**1.1.3** «KMines» (набор игр KDE Games)



* 1. **Постановка задачи**

В рамках данного курсового проекта планируется разработка игрового программного средства «Сапёр». В результате исследования соответствующей предметной области были поставлены следующие задачи:

- разработка алгоритмов:

* размещения мин на игровом поле;
* открытия ячейки;
* расстановка флагов;
* отслеживания победного победных и проигрышных состояний;

- реализация функций:

* подсчёта использованных флагов;
* подсчёта игрового времени;
* выбора уровня сложности;

Для создания программного средства будет использоваться язык программирования C++ и среда разработки Visual Studio 2019.

* 1. **РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА**

**2.1 Структура программы**

Приложение состоит из двух окон, которые представлены следующими структурными блоками (классами):

* StartWindow – стартовое окно, отображаемое пользователю после запуска игры с возможностью начать игру с определённым уровнем сложности;
* GameWindow – игровое окно, предоставляющее пользователю возможность взаимодействия с отображаемыми на экране игровыми объектами;

Также, вышеперечисленные классы StartWindow и GameWindow являются наследниками класса CustomWindow, который предоставляет дочерним классам виртуальные методы для определения собственной реализации. Такая структура хороша тем, позволяет ускорить разработку и избежать дублирования кода.

Далее представлены классы, также необходимые для реализации программного средства:

* App – модель, представляющая собой все приложение;
* Game – модель, представляющая собой отдельную сессию игры;
* Field – модель игрового поля данной игровой сессии;

Применительно к данным сущностям применяется композиционный подход, что вполне соответствует логической структуре приложения.

Помимо классов, специфических для конкретного программного средства, должны быть созданы:

* main – модуль, отвечающий за инициализацию приложения;
* Utils – модуль, содержащий вспомогательные функции;
* enums – модуль, содержащий определения перечислений;
* constants – модуль, содержащий определения основных констант приложения;

**2.2 Интерфейс программного средства**

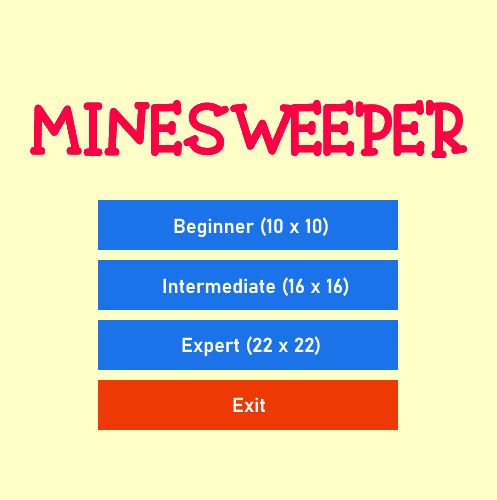
Внешний вид, удобство и доступность играют довольно важную роль в формировании опыта использования приложения. Однако в программных средствах игровой направленности пользовательский интерфейс является одной из наиболее критически важных для пользователя характеристик. Программное обеспечение с непонятным и сложным пользовательским интерфейсом редко вызывает одобрение у тех, кто им впоследствии пользуется. В связи с этим необходимо разработать его наиболее простым и понятным.

Меню стартового окна приложения состоит из четырёх кнопок:

* «Beginner (10 x 10)»;
* «Intermediate (16 x 16)»;
* «Expert (22 x 22)»;
* «Exit»;

Первые три кнопки используются для начала игровой сессии с определенными уровнем сложности и количеством мин (эти параметры указаны). Кнопка «Exit» предоставляет возможность закрыть приложение. Соответствующее действие происходит по нажатию на левую кнопку мыши.

Все элементы данного окна создаются при помощи sf::Sprite в совокупности с sf::Texture.



В верхней части игрового окна изображено закрытое минное поле. По мере открытия ячеек изображение в каждой из них сменяется либо на число мин, находящихся рядом, либо на картинку бомбы. В нижней части экрана располагается информационная панель. В её левой части, указано количество флагов, расставленное пользователем в течение данной игровой сессии, а справа – количество секунд, прошедшее с начала игры.

Для отрисовки каждой ячейки поля используется sf::RectangleShape. Для отображения мин и флагов применяется подход, аналогичный созданию кнопок в стартовом окне (с sf::Texture и sf::Sprite). Вывод на экран текста реализуется с помощью sf::Text.



**2.3 Основная логика программного средства**

Структурные блоки Game и Field ответственны за логику игры. В классе Game хранится экземпляр объекта Field, что непосредственно связывает игровую сессию с полем определенной конфигурации (размер и количество мин). Код конструктора класса Game представлен ниже.

Game::Game(Difficulty difficulty)

// инициализация членов класса (через список инициализации)

: field(getDifficultyRows(difficulty), getDifficultyCols(difficulty), getDifficultyMines(difficulty)),

gameDifficulty(difficulty) {

// присваивание количества мин (количество нерасставленных флагов)

flagsLeft = getDifficultyMines(difficulty);

// присваивание текущего игрового времени

resultGameTime = 0;

}

В коде выше стоит обратить внимание на следующее. Как уже говорилось, класс Field является членом класса Game. Поэтому, при создании объекта Game необходимо создавать (инициализировать) и объект Field. В данном примере для этих целей используется список инициализации членов класса. Это гораздо удобнее и правильнее, чем присваивать значение Field в теле конструктора, по причине того, что тогда необходимо:

* либо инициализировать объект Field при объявлении в классе Game;
* либо реализовывать ещё один конструктор (без параметров);

Эти два варианта сильно схожи, но ни один из них не предоставляет такой гибкости, как список инициализации членов класса, ведь сложность игры определяется динамически (по нажатии на соответствующую кнопку в окне StartWindow), а не при объявлении класса.

К тому же, с точки зрения логики, такая характеристика, как сложность, является атрибутом игровой сессии, но не поля. Для поля в данном случае более уместны такие понятия, как количество строк, столбцов и мин. Именно поэтому обработка поступившего значения сложности игры находится именно здесь. В вышеупомянутом примере, инициализация объекта Field производится с помощью функций, объявленных в модуле Utils. Его содержание выглядит следующим образом:

/\* директива препроцессора – включение данного файла в конечную сборку только 1 раз \*/

#pragma once

// включение файлов enums.h и constants.h

#include "../Config/enums.h"

#include "../Config/constants.h"

// объявления вспомогательных функций

int getDifficultyRows(Difficulty difficulty);

int getDifficultyCols(Difficulty difficulty);

int getDifficultyMines(Difficulty difficulty);

int getDifficultyCellSize(Difficulty difficulty);

float getDifficultyScaleFactor(Difficulty difficulty);

**2.4. Генерация игрового поля**

В данном приложении поле представлено классом Field. Создание объекта Field происходит следующим образом:

Field::Field(int rowsNumber, int colsNumber, int minesNumber) {

setFieldConfiguration(rowsNumber, colsNumber, minesNumber);

}

void Field::setFieldConfiguration(int rowsNumber, int colsNumber, int minesNumber) {

/\* присваивание значений из параметров соответствующим полям данного объекта \*/

this->rowsNumber = rowsNumber;

this->colsNumber = colsNumber;

this->minesNumber = minesNumber;

// добавление необходимого количества строк

opened.resize(this->rowsNumber);

mines.resize(this->rowsNumber);

flags.resize(this->rowsNumber);

/\* добавление каждой строке необходимого количества столбцов, при этом инициализируя каждую ячейку \*/

for (int i = 0; i < this->rowsNumber; i++) {

opened.at(i).resize(this->colsNumber, false);

mines.at(i).resize(this->colsNumber, 0);

flags.at(i).resize(this->colsNumber, false);

}

}

Ссылка this на текущий объект класса Field используется здесь по причине того, что имена формальных параметров функции Field::setFieldConfiguration идентичны названиям полей класса Field.

В отличие от метода, который был применен при создании объекта Game (список инициализации членов), здесь, при создании объекта Field, просто изменяется размер соответствующих векторов (с инициализацией элементов вектора нужным значением). Как видно из вышеприведенного кода, использование векторов привносит значительную ясность в код и является более удобным, чем использование обычных массивов.

В самых ранних реализациях игры «Сапёр» существовала проблема следующего содержания: пользователь в начале игры открывал ячейку, в которой оказывалась мина. Таким образом, игра завершалась, не успев начаться. Более поздние версии были лишены данного недостатка. Решается эта проблема следующим способом.

В конструкторе класса Field не расставляются мины. Генерация расстановки мин происходит после первого нажатия пользователя мышкой по игровому полю. В таком случае возможность открыть ячейку с миной на первом ходе устраняется. Код, отвечающий за такую логику, приведен ниже.

// если игра еще не началась

if (currentGameStatus == NOT\_STARTED) {

// если клик был произведен на игровом поле

if (isClickOnField(hoverMousePos)) {

unhighlightAll();

// подсветка ячейки, над которой находится мышка

highlightCell(xHover, yHover);

// начало игры

game.restart();

// если нажата левая кнопка мыши

if (leftButtonState == PRESSED) {

/\* инициализация мин таким образом, чтобы в ячейке с индексами xHover, yHover не было мины (а также в соседних ячейках в радиусе 1 клетка) \*/

game.getField()->initializeMines(xHover, yHover);

/\* расстановка значений количества мин в соседних клетках \*/

game.getField()->initializeDigits();

// попытка открыть ячейку с индексами xHover, yHover

game.getField()->openCell(xHover, yHover);

}

// если нажата правая кнопка мыши

else if (rightButtonState == PRESSED) {

// попытка поставить флаг на выбранную ячейку

game.setFlag(xHover, yHover);

}

}

}

void Field::initializeMines(int x, int y) {

/\* расстановка количества мин, заданного при создании объекта класса Field \*/

int minesLeft = minesNumber;

// радиус, в котором будет исключена расстановка мин

int const xOffset = 1, yOffset = 1;

/\* каждый раз выбираем новую «базу» для псевдослучайной расстановки мин \*/

srand(time(0));

// расставляем мины, пока они не закончились

while (minesLeft) {

// вычисляем значение будущих индексов ячейки с миной

int xRand = rand() % (rowsNumber - 1);

int yRand = rand() % (colsNumber - 1);

/\* если новые координаты не лежат в пределах радиуса, в котором запрещена генерация мин \*/

if (!(x - xOffset <= xRand && xRand <= x + xOffset) ||

!(y - yOffset <= yRand && yRand <= y + yOffset)) {

// не устанавливаем мину на место другой мины

if (!hasMineAt(xRand, yRand)) {

mines.at(xRand).at(yRand) = MINE;

minesLeft--;

}

}

}

}

**2.5 Взаимодействие пользователя с полем**

На поле пользователь может производить следующие действия:

* открывать ячейки;
* расставлять флаги;

Установка флага на игровое поле представляет собой довольно простые операции:

void Game::setFlag(int x, int y) {

/\* если флаг был в ячейке до нажатия на нее правой кнопки мыши, то увеличиваем количество непоставленных флагов, иначе уменьшаем \*/

field.hasFlagAt(x, y) ? flagsLeft++ : flagsLeft--;

// если флаг был, то убираем, если не было, то ставим

field.toggleFlag(x, y);

}

void Field::toggleFlag(int x, int y) {

// изменяем признак присутствия флага в ячейке на обратный

flags.at(x).at(y) = !flags.at(x).at(y);

}

Логика открытия ячейки является более комплексной: если в открытой клетке нет ни мины, ни цифры (абсолютно пустая ячейка), то, в случае если рядом есть пустые клетки (нет мины), необходимо открывать целую область на поле, в которую входит нажатая ячейка. При этом эта область ограничена цифрами (нажатая ячейка может входить в эту границу). Приведенный ниже код отражает логику открытия ячеек.

void Field::openCell(int x, int y) {

// если открывается ячейка, на которой нет флага

if (!hasFlagAt(x, y)) {

// открываем саму ячейку

opened.at(x).at(y) = true;

// если в ячейке пусто

if (mines.at(x).at(y) == EMPTY) {

// итерация по столбцам выделенной области

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

// итерация по строкам выделенной области

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

// если индексы лежат в пределах поля

if (hasCell(i, j)) {

// если в ячейке цифра

if (mines.at(i).at(j) != EMPTY && !hasMineAt(i, j)) {

// открываем её

opened.at(i).at(j) = true;

}

// если в ячейке абсолютно пусто

else if (mines.at(i).at(j) == EMPTY) {

// если она еще закрыта

if (!opened.at(i).at(j)) {

// повторяем ту же процедуру

openCell(i, j);

}

}

}

}

}

}

}

}

**2.6 Работа окон приложения**

Логика работы с окнами приложения содержится в классах CustomWindow, StartWindow, GameWindow. Рассмотрим жизненный цикл окна на примере GameWindow.

GameWindow является наследником класса CustomWindow (преимущества такой структуры были описаны выше). При этом реализует три метода класса-родителя: initResources, checkActions и render.

Метод initResources вызывается в конструкторе окна. Его задача – подготовить необходимые ресурсы окна для того, чтобы его отобразить. Эта процедура включает в себя загрузку текстур, спрайтов, задание положения на экране некоторых объектов, которые на протяжении всего цикла отрисовки будут неподвижны, вычисление некоторых значений, которые тем или иным образом влияют на конечный вид элементов на экране. Например, значение переменной scaleFactor отличается для игровых окон различных размеров, и обозначает, во сколько раз нужно масштабировать элемент. Для вычисления этого значения служит процедура getDifficultyScaleFactor:

float getDifficultyScaleFactor(Difficulty difficulty) {

// выбираем нужный масштаб в зависимости от сложности

switch (difficulty) {

case BEGINNER: return (float)BEGINNER\_CELL\_SIZE / BEGINNER\_CELL\_SIZE;

case INTERMEDIATE: return (float)INTERMEDIATE\_CELL\_SIZE / BEGINNER\_CELL\_SIZE;

case EXPERT: return (float)EXPERT\_CELL\_SIZE / BEGINNER\_CELL\_SIZE;

default: return (float)BEGINNER\_CELL\_SIZE / BEGINNER\_CELL\_SIZE;

};

}

От уровня сложности зависит количество ячеек игрового поля. Чем больше ячеек, тем больше необходимо уменьшить отображаемые элементы, чтобы они помещались на экране. Вышеприведенная формула соответствует данным требованиям.

Вернемся к жизненному циклу окна. Он находится в методе render и выглядит следующим образом:

// пока окно открыто

while (window.isOpen()) {

/\* заливаем окно указанным цветом, тем самым фактически очищая его \*/

window.clear(GAME\_BG\_COLOR);

// отрисовка поля

drawField();

// проверка нажатий

checkActions();

/\* отображение того, что подготовлено к отрисовке (double buffering) \*/

window.display();

}

Сначала происходит заливка окна указанным цветом, затем отрисовка поля, что делает процедура drawField. Блок-схема подпрограммы drawField представлена далее.



После этого отрабатывает процедура checkActions. Она проверят текущее состояние мышки, а именно то, какие кнопки нажаты и в каком состоянии они находились до этого. На основании полученных данных определяется действие, которое необходимо выполнить. В checkActions может происходить дополнительная отрисовка некоторых элементов, поэтому этот метод вызывается после drawField и перед window.display. Фрагмент кода, дающий представление о том, как работает метод checkActions, приведен ниже.

**//привести фрагмент кода**

* 1. **ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА**
  2. **РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ**

**4.1 Правила игры**

Правила данной игры предельно просты и понятны. К тому же, они полностью идентичны правилам «Сапёра», поставляемого вместе с Windows.

Плоское игровое поле разделено на смежные ячейки (квадраты), некоторые из которых «заминированы». Количество «заминированных» ячеек известно. Целью игры является открытие всех ячеек, не содержащих мины.

Игрок открывает ячейки, стараясь не открыть ячейку с миной. Открыв ячейку с миной, он проигрывает. Мины расставляются после первого хода, поэтому проиграть на первом же ходу невозможно. Если под открытой ячейкой мины нет, то в ней появляется число, показывающее, сколько ячеек, соседствующих с только что открытой, «заминировано».

Используя эти числа, игрок пытается рассчитать расположение мин, однако иногда даже в середине и в конце игры некоторые ячейки всё же приходится открывать наугад. Если под соседними ячейками тоже нет мин, то открывается некоторая «не заминированная» область до ячеек, в которых есть цифры. «Заминированные» ячейки игрок может пометить, чтобы случайно не открыть их. Открыв все «не заминированные» ячейки, игрок выигрывает.

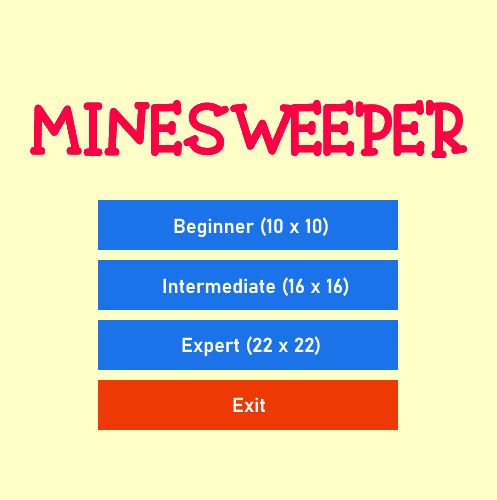
**4.2 Интерфейс программы**

Меню стартового окна приложения состоит из четырёх кнопок:

* «Beginner (10 x 10)»;
* «Intermediate (16 x 16)»;
* «Expert (22 x 22)»;
* «Exit»;

Первые три кнопки используются для начала игровой сессии с определенными уровнем сложности и количеством мин (эти параметры указаны). Кнопка «Exit» предоставляет возможность закрыть приложение. Соответствующее действие происходит по нажатию на левую кнопку мыши.

Все элементы данного окна создаются при помощи sf::Sprite в совокупности с sf::Texture.



В верхней части игрового окна изображено закрытое минное поле. По мере открытия ячеек изображение в каждой из них сменяется либо на число мин, находящихся рядом, либо на картинку бомбы. В нижней части экрана располагается информационная панель. В её левой части, указано количество флагов, расставленное пользователем в течение данной игровой сессии, а справа – количество секунд, прошедшее с начала игры.

Для отрисовки каждой ячейки поля используется sf::RectangleShape. Для отображения мин и флагов применяется подход, аналогичный созданию кнопок в стартовом окне (с sf::Texture и sf::Sprite). Вывод на экран текста реализуется с помощью sf::Text.



Добавить проавил

**ЗАКЛЮЧЕНИЕ**

Сегодня индустрия компьютерных игр не является чем-то незнакомым для рядового пользователя. Каждый из нас знает, что это, но не каждый из нас знает, что за этим стоит. Игры давно перестали быть лишь развлечением. Разработчики вкладывают в сегодняшние игры не только огромные деньги, но и смысл. В контексте последнего данное приложение не исключение. За завесой незамысловатой офисной игры находится то, что способно захватить внимание пользователя на десятки минут, а то и на часы.

В рамках данного курсового проекта было разработано игровое программное средство «Сапёр». Были разработаны все планируемые функции и возможности, а именно:

* размещения мин на игровом поле;
* открытия ячейки;
* расстановка флагов;
* отслеживания победного победных и проигрышных состояний;

Чтобы успешно справиться с поставленными задачами, понадобилось освоить принципы парадигмы объектно-ориентированного программирования, возможности языка программирования С++ в контексте этих принципов и среду разработки Visual Studio 2019.

Несмотря на то, что разработка приложения завершена, и имеется вполне хороший продукт, существует широкий спектр направлений для улучшения данного программного средства. Например, усовершенствование уже разработанных и реализованных в приложении алгоритмов или создание новых возможностей. В последнее можно включить:

* добавление настроек;
* сохранение результатов в таблицу рекордов;
* ведение статистики игр за все время;
* графические улучшения: добавление анимации взрыва мин и открытия «незаминированной» области и так далее;
* добавление звукового сопровождения.

Как уже упоминалось, данная игра способствует развитию логического и тактического мышления. Механизм игры буквально требует, чтобы игрок аккуратно и взвешенно открывал определенные ячейки обходился с имеющимися подсказками, текущей возможностью хода, думал на несколько шагов вперед, чтобы совершить успешный ход и выиграть.