1. Министерство образования и науки Российской Федерации
2. Санкт-Петербургский Политехнический Университет Петра Великого
3. —
4. **Институт кибербезопасности и защиты информации**

**КУРСОВАЯ РАБОТА**

**Разработка игры «Сапер»**

По дисциплине «Структуры данных»

|  |  |  |  |
| --- | --- | --- | --- |
| Выполнили студенты гр. 5151003/30001 | <*подпись*> | Вторушина Д. А.  Шамратова Р. И. |  |
| Руководитель асс.преподавателя | <*подпись*> | Пахомов М. А. |  |

« » 2024 г.

Санкт-Петербург

2024

**СОДЕРЖАНИЕ**

|  |  |
| --- | --- |
| Введение. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 3 |
| Цель работы . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 3 |
| Постановка задачи . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 3 |
| 1. Теоретическая часть . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 3 |
| 1.1. OpenGL. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 4 |
| 2. Практическая часть . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 7 |
| 2.1. Описание работы каждой функции . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 7 |
| 2.2. Описание работы бота . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 17 |
| 2.3. Запуск игры . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 18 |
| Заключение . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 23 |
| Список использованной литературы . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 24 |
| Приложение. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 25 |

**ВВЕДЕНИЕ**

**Цель работы**

Разработать игру «Сапёр» на языке программирования Си.

**Постановка задачи**

1) получить знания и навыки работы с библиотекой OpenGL для разработки 2D графики, необходимых для реализации игры «Сапёр». Ознакомиться с библиотекой GLUT для создания и управления окнами и рисования в них, обработки событий ввода. Разработать интерфейс игры.

2) реализовать возможность выбора уровня сложности (легкий, средний, сложный), изменяющего размеры поля и количество бомб.

3) разработать бота, который поможет в случае затруднений игрока.

4) разработать алгоритмы установки бомб, открытия ячеек, заполнения игрового поля числовой информацией.

**1. Теоретическая часть**

**1.1. OpenGL**

OpenGL (OpenGraphicsLibrary) — спецификация, определяющая платформонезависимый (независимый от языка программирования) программный интерфейс для написания приложений, использующих двумерную и трёхмерную компьютерную графику.

В разработанной работе основная библиотека для работы с графикой — это GLUT, которая является оболочкой для OpenGL.GLUT (OpenGL Utility Toolkit) — это библиотека, которая предоставляет набор функций для создания и управления окнами, а также обработки ввода, что облегчает разработку приложений на OpenGL. GLUT является кроссплатформенной библиотекой, что позволяет писать графические приложения, которые будут работать на различных операционных системах без необходимости модификации кода.

Список используемых функций из этой библиотеки:

1) glutInit: инициализация GLUT и обработка аргументов командной строки.

2) glutInitDisplayMode(GLUT\_DEPTH | GLUT\_SINGLE | GLUT\_RGBA): задает режим отображения окна. В данном случае используются следующие флаги: GLUT\_DEPTH: включает буфер глубины, который используется для определения видимости пикселей в трехмерной графике. GLUT\_SINGLE: включает использование одинарной буферизации (один буфер для отображения). Обычно используется для простых приложений. GLUT\_RGBA: включает режим отображения с использованием цветовой модели RGBA (красный, зеленый, синий, альфа-канал).

3) glutInitWindowPosition(250, 100): эта функция задает начальную позицию окна на экране. В данном случае окно будет создано с верхним левым углом в точке (250, 100) относительно верхнего левого угла экрана.

4) glutInitWindowSize(500, 600): эта функция задает начальный размер окна. В данном случае окно будет иметь ширину 500 пикселей и высоту 600 пикселей.

5) glutCreateWindow("Minesweep"): функция создает окно с указанным заголовком ("Minesweep") и устанавливает его текущим окном для последующих операций OpenGL.

6) glutDisplayFunc(starting): Эта функция регистрирует функцию обратного вызова для отрисовки (displaycallback). Это основная функция для выполнения всех операций рисования. Она будет вызвана каждый раз, когда требуется перерисовка окна, например, при его обновлении.

7) glutMouseFunc(mouse1): регистрирует функцию для обработки событий мыши. Функция, указанная в аргументе (mouse1/mouse2), будет вызвана при нажатии или отпускании кнопки мыши. В функции определяется, какая кнопка была нажата и какие координаты курсора, чтобы соответствующим образом отреагировать на событие.

8) glutMainLoop(): функция запускает основной цикл обработки событий GLUT. Она передает управление библиотеке GLUT и не возвращается до завершения программы. В этом цикле обрабатываются все зарегистрированные события, такие как отрисовка, нажатия клавиш, движения мыши и т. д.

9)glClear(GL\_COLOR\_BUFFER\_BIT):дляочисткибуферакадра. GL\_COLOR\_BUFFER\_BITуказывает на буфер цвета.

10) glColor3f(floatred, floatgreen, floatblue):устанавливает текущий цвет рисования.Параметры определяют красную, зеленую и синюю компоненты цвета соответственно. Значения компонентов в пределах от 0 до 1.

1. glBegin(GL\_POINTS) : функция рисования точки
2. glBegin(GL\_POLYGON): задает рисование многоугольника с количеством углов равным количеству заданных вершин.
3. glVertex2f(): функция задания вершин.число параметров =2 + тип параметров- f (float).
4. glutDisplayFunc(display): устанавливает функциюdisplay, которая будет вызываться для перерисовки окна GLUT.
5. glutTimerFunc(): функция регистрирует функцию обратного вызова, которая будет вызвана через указанное количество миллисекунд.
6. glutPostRedisplay(): помечает текущее окно как нуждающееся в перерисовке. Это не вызывает немедленную перерисовку окна, а ставит флаг, который сообщает GLUT, что нужно вызвать функцию отображения при следующем удобном случае. Когда очередь дойдет до этого события, GLUT вызовет функцию, установленную с помощью glutDisplayFunc(). Она позволяет объединить несколько изменений в одну перерисовку, что оптимизирует работу приложения.

**2. Практическая часть**

**2.1. Описание работы каждой функции**

Создана функция bitmap (рис.1), которая предназначена для отображения текстовых строк в заданных координатах на экране. В ней функция glRasterPos2f(x, y) устанавливает позицию начала отображения текста на экране. Для каждого символа в строке вызывается функция glutBitmapCharacter, которая отображает этот символ на текущей позиции.

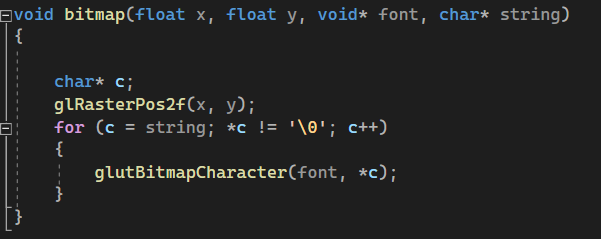
****

Рисунок 1 – Функция для отображения строк

Функция wait() (рис.2) используется для создания задержки выполнения программы на полсекунды. В переменную start1 сохраняется начальное время, с которого начинается ожидание. Цикл продолжается, пока разница между текущим временем и начальным временем, деленная на количество тактов в секунду (CLOCKS\_PER\_SEC), меньше 0.5 секунды. Поскольку clock() возвращает количество прошедших тактов процессора с начала выполнения программы, это позволяет создать паузу в выполнении программы на 0.5 секунды. Функция используется, например, в момент вывода на экран всех бомб в случае подрыва.

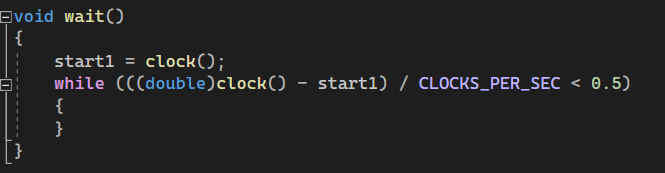


Рисунок 2 - Функция wait

Функция draw(int x, int y) отвечает за отрисовку флага в игре. Размеры и координаты флага и флагштока зависят от размера игрового поля (size\_field), что позволяет флагу сохранять пропорции независимо от уровня сложности игры (рис. 3). Аналогичным образом создана функция bomb() для создания рисунка бомбы.

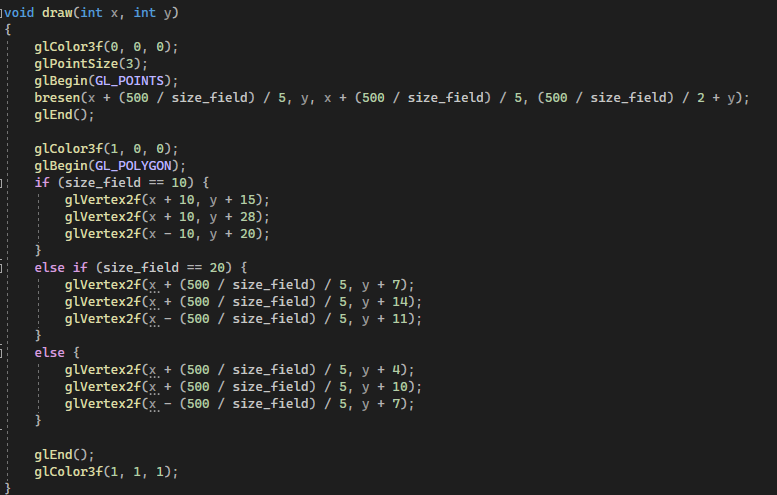
****

Рисунок 3 - Функция рисунка флага

Функция fill (рис. 4) заполняет массив arr числовыми значениями, которые указывают количество бомб, находящихся в соседних ячейках для каждой клетки игрового поля. Если в клетке находится бомба, то эта клетка не изменяется (остается значением -1).Для каждой клетки, которая не содержит бомбу, функция проверяет все восемь соседних клеток. Для каждой соседней клетки проверяется, находится ли она в пределах игрового поля и содержит ли она бомбу. Если обе проверки проходят, счетчик a увеличивается на 1.После проверки всех соседних клеток для текущей клетки, в массив arr записывается значение a, которое указывает, сколько бомб находится в соседних клетках. Это ключевая часть игры Сапер, где игрок видит числа на поле, помогающие ему определить местоположение бомб.

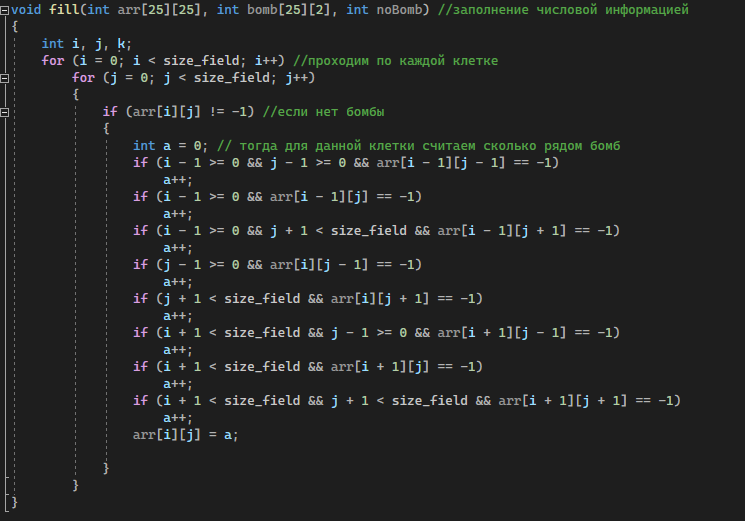
****

Рисунок 4– Функция для заполнения поля числовой информацией

Функция spread (рис. 5) используется для открытия пустых клеток на игровом поле. Когда игрок открывает клетку, которая не содержит бомбу(значение не -1) и еще не была открыта (значение не 1 в массиве visible), то клетка открывается. Но если в ней не храниться числа о количестве бомб:

(if (arr[i][j] == 0)), то функция рекурсивно вызывает саму себя для всех восьми соседних клеток, продолжая до тех пор, пока не встретит клетки с ненулевыми значениями или края игрового поля. Таким образом, Функция spreadочень важна и используется для автоматического открытия всех смежных пустых клеток на игровом поле.

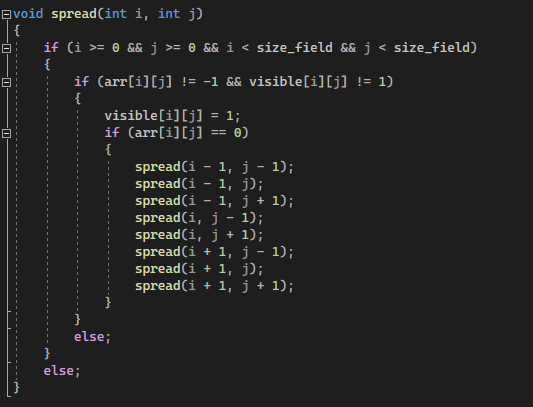
****

Рисунок 5–Рекурсивное открытие пустых клеток

Функция outline (рис. 6) используется для рисования сетки, которая делит игровое поле на клетки.Первый цикл рисует вертикальные линии (меняется координата х), а второй – горизонтальные (меняется координата у).Координата х/у вычисляется путем деления ширины игрового поля (500) на количество клеток (size\_field - разное для разного уровня сложности игры) и умножения на индекс i. Линии рисуются с помщью функцииbresen по алгоритму Брезенхема. Она определена в заголовочном файле line.hАлгоритм Брезенхемаэффективно рисует прямые линии с использованием целочисленных вычислений, что делает его быстрым и удобным для реализации.

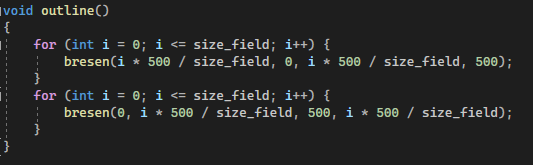
****

Рисунок 6 - Генерация сетки поля

Функция setbomb() (рис. 7) отвечает за случайную установку бомб на игровом поле в игре "Сапёр".srand(time(NULL));: инициализация генератора случайных чисел текущим временем, чтобы каждый запуск программы генерировал уникальную последовательность случайных чисел. В цикле происходит генерация случайного числа от 0 до количества клеток на поле, вычисляется строка и столбец на игровом поле. Запускается цикл, который проверяет, совпадают ли новые координаты с уже существующими. Если координаты совпадают, происходит переход к метке manb для генерации нового случайного числа. Так алгоритм гарантирует, что каждая бомба будет установлена на уникальную клетку. Далее сохраняем координаты бомбы и проходимся по всем установленным бомбам: помечаем клетки, содержащие бомбы, значением -1 в массиве arr. После установки бомб вызываем функциюfill(), которая заполняет поле информацией о количестве соседних бомб для каждой клетки.

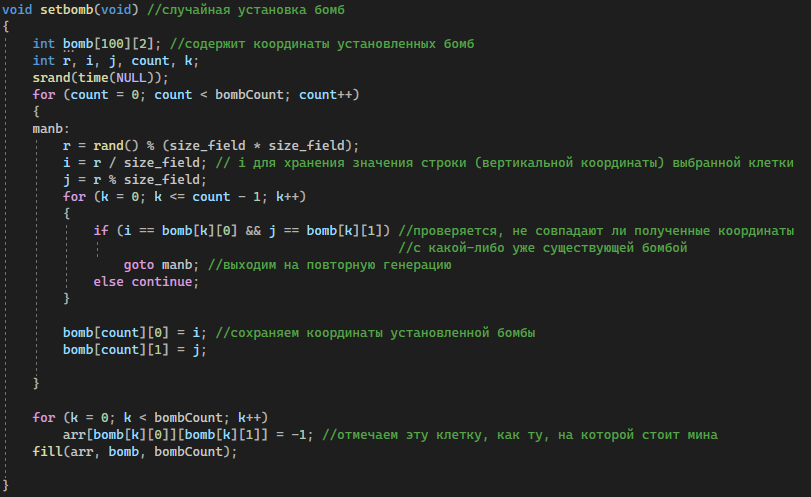


Рисунок 7 - Случайная расстановка бомб на поле

Далее разберем функцию display(). Она обновляется и вызывается периодически, чтобы отобразить текущее состояние игры.glClear(GL\_COLOR\_BUFFER\_BIT): очищает буфер цвета перед рисованием нового изображения, чтобы избежать размытости или "остаточных" изображений прошлых кадров. Первые два цикла отвечают за работу бота, они будут описаны подробно далее.

out - переменная, которая указывает, был ли подрыв на бомбе. Если out == true, то игрок подорвался на бомбе: проигрывается звуковой сигнал. Отображаются открытые клетки (всеоткрытые клетки становятся белыми) и отрисовываются все оставшиеся бомбы на игровом поле, отображается сообщение "YOU LOSE", с музыкальным сопровождением в виде взрыва бомбы.Игра завершается.

Иначе, если игрок нажал на клетку без бомбы:

Для каждой клетки на поле проверяется ее видимость: если клетка открыта (visible[i][j] == 1), то рисуется белый квадрат, представляющий открытую клетку. Если в клетке содержится число (arr[i][j] != 0), оно отображается в центре клетки. Если на клетке установлен флаг (visible[i][j] == 2), то рисуется флаг с помощью функции draw(). Далее проверка на завершение игры: если условия победы выполнены, то игра завершается и выводится сообщение "YOU WON".

Для проверки победы реализована функция finish() (рис. 8) , которая возвращает значение 1 (победа), если открыты все клетки, не содержащие бомбы, а также проверяется, нет ли ячейки с флагом, но без бомбы. Она проходит по всем клеткам поля, если какая-то клетка не содержит бомбы, но при этом она закрыта (visible[i][j] == 0) или есть ли ячейка с флагом, но без бомбы, то игра продолжается (return 0), так как не все клетки без бомбы открыты или не все оставшиеся клетки с флагом – бомбы, также проверяется условие закрытой клетки с бомбой, чтобы игра не завершалась заранее.

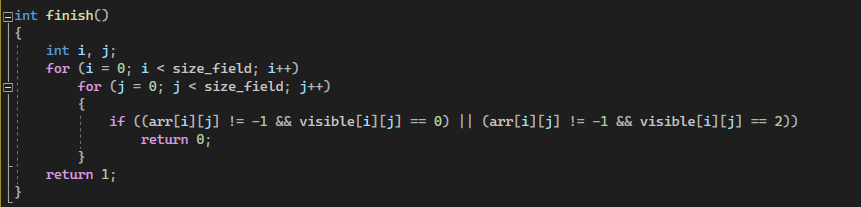


Рисунок 8 - Функция finish

Функция mouse2 (рис. 9) отвечает за обработку событий мыши уже в моменте самой игры, а не выбора уровня сложности, в отличие от функции mouse1.

Координаты x и y мыши преобразуются в индексы posi и posj в массиве игрового поля, чтобы определить, в какую клетку произошло нажатие. Если клетка уже открыта, то gotoman; (команда бездействовать). При нажатии правой кнопки мыши (mouse == 2) проверяется, стоит ли уже флаг (visible[posi][posj] == 2) и было ли это повторное нажатие (проверяется через mouseVisit % 2 == 1). Если флаг уже установлен, он снимается, иначе он устанавливается.При нажатии колеса мыши (mouse == 1) происходит работа бота (подробнее ниже). При нажатии левой кнопки мыши

(mouse == 0): Если в клетке находится бомба (arr[posi][posj] == -1), устанавливается флаг out = 1, указывающий на поражение игрока, и запоминаются координаты клетки (a[0] и a[1]). Если клетка пустая (arr[posi][posj] == 0), вызывается функция spread() для открытия смежных пустых клеток. В противном случае (arr[posi][posj] != 0), клетка отмечается как открытая (visible[posi][posj] = 1).

Метка wrngpos используется, если клик произошел в недопустимой области (например, выше поля игры).

После обработки события мыши вызывается функция glutDisplayFunc(display) для обновления изображения и glutPostRedisplay() для перерисовки окна.

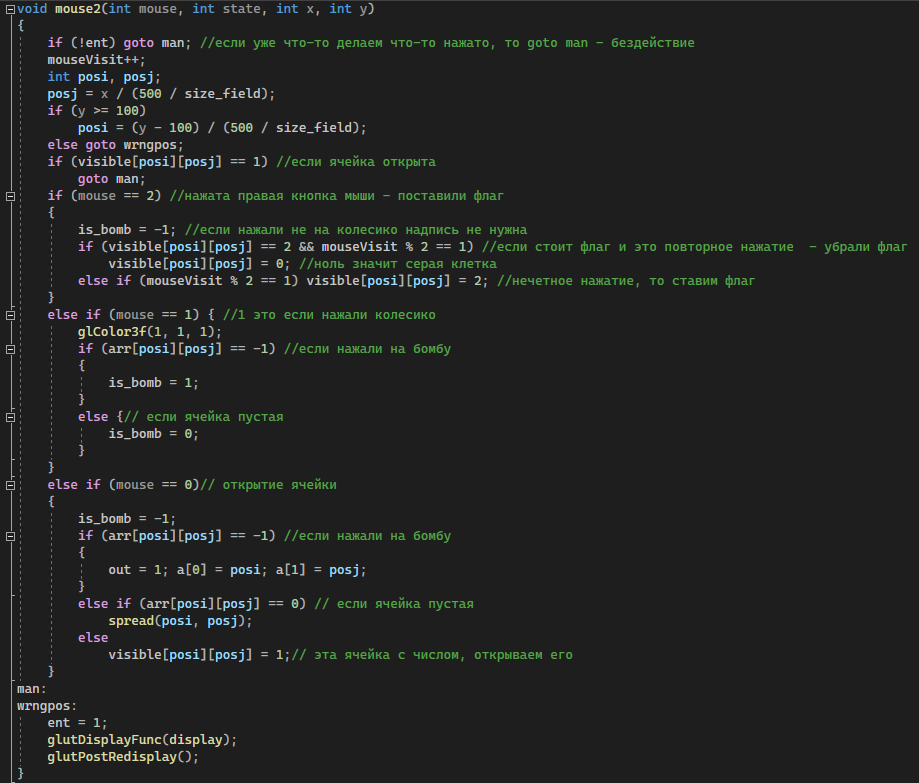


Рисунок 9 - Функция mouse2

Функция mouse1 (рис. 10) отвечает за выбор уровня сложности игры при клике мышью. Если клик произошел в определенной области, соответствующей уровню сложности игры, то устанавливаются соответствующие параметры:

Дляуровня "легкий": size\_field = 10, bombCount = 8.

Дляуровня "средний": size\_field = 20, bombCount = 40.

Дляуровня "сложный": size\_field = 25, bombCount = 90.

После установки параметров вызывается функция setbomb() для установки бомб на поле, и начинается отсчет времени с помощью функции start = clock(). Далее устанавливаются функции обработки событий мыши (glutMouseFunc(mouse2)) и обновления экрана (glutDisplayFunc(display)), а также запускается таймер для обновления игры (glutTimerFunc(500, update, 0)). Наконец, вызывается glutPostRedisplay() для перерисовки окна и отображения выбранного уровня сложности.

Если клик произошел вне области выбора уровня, вызывается функция starting(), которая отображает начальный экран игры.

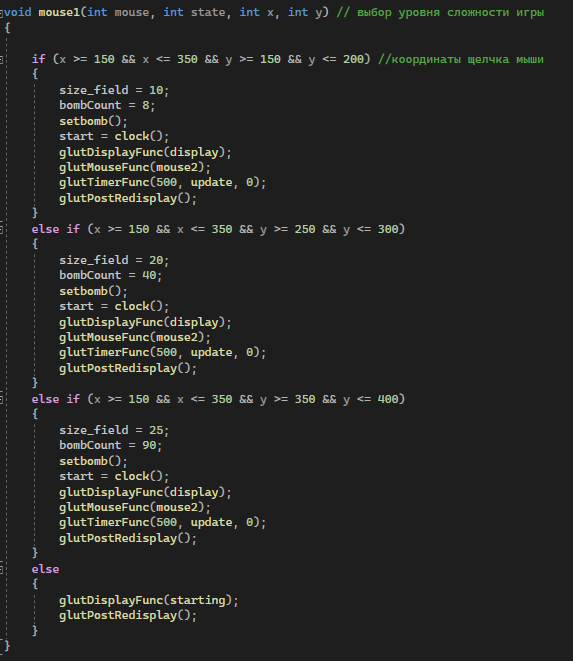
****

Рисунок 10 - Функция mouse1

Функция starting() (рис. 11) отображает начальный экран игры, где игроку предлагается выбрать уровень сложности. На экране выводятся три варианта уровней сложности: "Easy", "Medium" и "Hard".

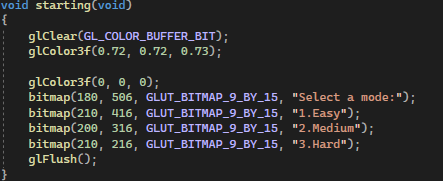


Рисунок 11 - Функция starting

В функции main (рис. 12) инициализируется окно GLUT и устанавливается режим отображения одноканальный буфер цвета и цветовая модель RGBA. Задаются параметры позиции и размера окна. Создается окно с заголовком "Minesweep". Вызывается функция Init() для инициализации игры.Устанавливается начальная функция отображения starting().Устанавливаются функции обработки событий мыши (glutMouseFunc(mouse1)), которая отвечает за выбор уровня сложности, и основной цикл GLUT с помощью glutMainLoop(), который обрабатывает события ввода и управляет выполнением программы до момента ее завершения.

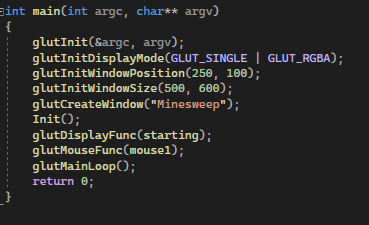


Рисунок 12 - Функция main

**2.2. Описание работы бота**

Помимо реализации самой игры, необходимо было реализовать бота, который как-либо должен был «помогать» в ходе игры (рис. 13). Бот не должен понимать, где находятся бомбы, т.е у него нет доступа к массиву данных, где хранятся местоположения бомб. Условие is\_bomb указывает на наличие бомбы, если она есть, то на экране появляется надпись «There is bomb», иначе «There is no bomb», или «Probably the bomb is here», когда достоверно не известно. Программа активирует бота, с помощью нажатия на колесо мышки.

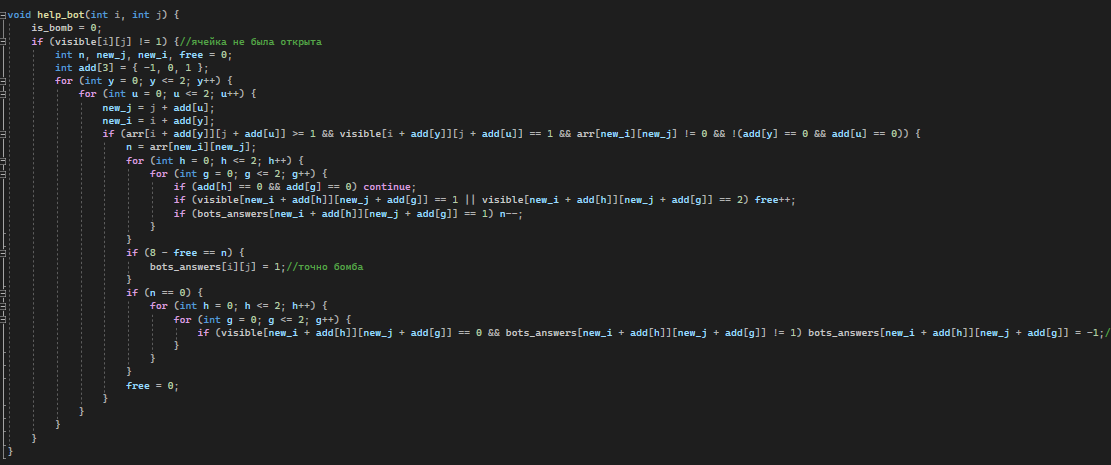


Рисунок 13 – Алгоритм работы бота

Сначала мы выбираем клетку на поле, далее бот анализирует все клетки вокруг нее и все клетки вокруг этих клеток. Далее, учитывая значение клетки и количество открытых клеток, бот понимает, есть ли там бомба, возможно она там есть или ее точно нет.

Например, в случае как на рисунке 14, там, где стоит флаг, бот указывал, что бомба точно там, так как бот, анализируя все клетки вокруг, понял что у клетки, расположенной по диагонали сверху слева(справа) открыто 8 клеток из 9 возможных, следовательно, бот понимал, что бомба точно там. На месте ячейки, выделенной на рисунке, бот указывает, что в ней точно нет бомбы, так как у единички по диагонали сверху уже поставлена своя бомба, следовательно, в ее окружении бомб больше быть не может.

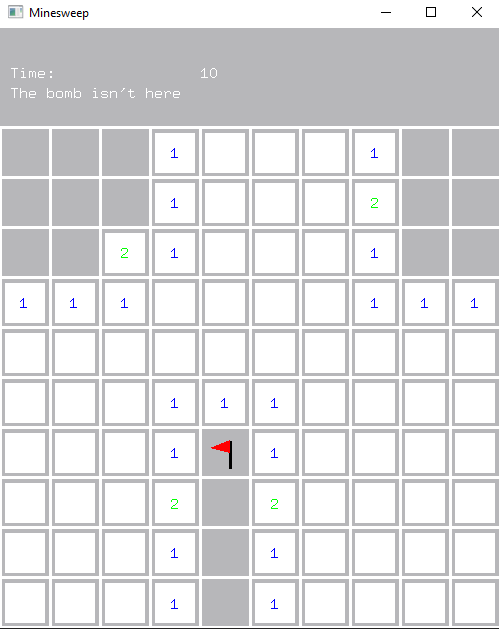


Рисунок 14 – Анализ работы бота

**2.3. Запуск игры**

Начальное отображение игры представлено на рисунке 15. Пользователь должен выбрать один из уровней, в который хочет сыграть.

**Изображение выглядит как текст, снимок экрана

Автоматически созданное описание**

Рисунок 15 – Выбор режима игры

На рисунке 16 представлен один из этапов игры. Продемонстрирована корректность работы открытия ячеек, установления флага.

**Изображение выглядит как снимок экрана, прямоугольный, Прямоугольник, линия

Автоматически созданное описание**

Рисунок 16 – Режим игры «Easy», открытые клеток поля, флаг

Рисунок 17 – случай, когда пользователь проиграл. Программа завершается соответствующей фразой.

Изображение выглядит как снимок экрана, текст, компьютер, программное обеспечение

Автоматически созданное описание

Рисунок 17 – Проигрыш

На рисунке 18 представлен режим работы «Medium», который отличается количеством клеток и расставленных бомб.

**Изображение выглядит как снимок экрана, текст, дизайн

Автоматически созданное описание**

Рисунок 18 – Режим игры «Medium»

Аналогично, уровень «Hard» (рис. 19).

**Изображение выглядит как снимок экрана, текст, компьютер, дисплей

Автоматически созданное описание**

Рисунок 19– Режим игры «Hard»

**Заключение**

Таким образом, была разработана игра «Сапёр» на языке программирования Си, а также получены знания и навыки работы с библиотекой OpenGL для разработки 2D графики, необходимые для реализации игры "Сапёр", с библиотекой GLUT для создания и управления окнами и рисования в них, обработки событий ввода. Кроме того, был разработан интерфейс игры, реализована возможность выбора уровня сложности (легкий, средний, сложный), изменяющего размеры поля и количество бомб, разработан бот, который поможет в случае затруднений игрока, разработаны алгоритмы установки бомб, открытия ячеек, заполнения игрового поля числовой информацией. При подрыве на бомбе воспроизводится звуковой эффект, добавляющий элемент неожиданности и реализма. Мы постарались сделать игру не только функциональной, но и визуально привлекательной.

**Список использованной литературы**

1)OpenGL - Урок 1 - Первая программа // YouTube URL: https://www.youtube.com/watch?v=09GRZ0c0LlA&list=PLBOPkQsFLCR0XuXoCf6PMhVmD2ukHXIBK&ab\_channel=BeginEnd (дата обращения: 15.04.2024).

2) Графическая библиотека OpenGL. Методическое пособие // URL: https://www.rsdn.org/article/opengl/ogltut2.xml#EDIAI (дата обращения: 15.04.2024).

3) Как использовать шрифты в GLUT? // URL: https://programmingcpp.narod.ru/font\_in\_GLUT.htm (дата обращения: 15.04.2024).

**Приложение**

Листинг программы «Сапер».

#define \_CRT\_SECURE\_NO\_WARNINGS

#include<stdlib.h>

#include<windows.h>

#include<glut.h>

#include<stdio.h>

#include<time.h>

#include<math.h>

#include"line.h"

#ifdef \_UNICODE

#define T\_STRING(x) L##x

#else

#define T\_STRING(x) x

#endif

clock\_t start1;

clock\_t start;

int visible[25][25];

int arr[25][25];

int out = 0, bombCount;

int mouseVisit = 0, ent = 0;

int a[2];

int over = 0;

int size\_field;

int is\_bomb = -2;

int bots\_answers[25][25];

void Init()

{

glClearColor(0.72, 0.72, 0.73, 0.0);

glEnable(GL\_DEPTH);

glShadeModel(GL\_SMOOTH);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

glOrtho(0, 500, 0, 600, 0, 500);

for (int i = 0; i < size\_field; i++) {

for (int j = 0; j < size\_field; j++) {

bots\_answers[i][j] = 0;

}

}

}

void bitmap(float x, float y, void\* font, char\* string)

{

char\* c;

glRasterPos2f(x, y);

for (c = string; \*c != '\0'; c++)

{

glutBitmapCharacter(font, \*c);

}

}

void wait()

{

start1 = clock();

while (((double)clock() - start1) / CLOCKS\_PER\_SEC < 0.5)

{

}

}

void draw(int x, int y)

{

glColor3f(0, 0, 0);

glPointSize(3);

glBegin(GL\_POINTS);

bresen(x + (500 / size\_field) / 5, y, x + (500 / size\_field) / 5, (500 / size\_field) / 2 + y);

glEnd();

glColor3f(1, 0, 0);

glBegin(GL\_POLYGON);

if (size\_field == 10) {

glVertex2f(x + 10, y + 15);

glVertex2f(x + 10, y + 28);

glVertex2f(x - 10, y + 20);

}

else if (size\_field == 20) {

glVertex2f(x + (500 / size\_field) / 5, y + 7);

glVertex2f(x + (500 / size\_field) / 5, y + 14);

glVertex2f(x - (500 / size\_field) / 5, y + 11);

}

else {

glVertex2f(x + (500 / size\_field) / 5, y + 4);

glVertex2f(x + (500 / size\_field) / 5, y + 10);

glVertex2f(x - (500 / size\_field) / 5, y + 7);

}

glEnd();

glColor3f(1, 1, 1);

}

void fill(int arr[25][25], int bomb[25][2], int noBomb)

{

int i, j, k;

for (i = 0; i < size\_field; i++)

for (j = 0; j < size\_field; j++)

{

if (arr[i][j] != -1)

{

int a = 0;

if (i - 1 >= 0 && j - 1 >= 0 && arr[i - 1][j - 1] == -1)

a++;

if (i - 1 >= 0 && arr[i - 1][j] == -1)

a++;

if (i - 1 >= 0 && j + 1 < size\_field && arr[i - 1][j + 1] == -1)

a++;

if (j - 1 >= 0 && arr[i][j - 1] == -1)

a++;

if (j + 1 < size\_field && arr[i][j + 1] == -1)

a++;

if (i + 1 < size\_field && j - 1 >= 0 && arr[i + 1][j - 1] == -1)

a++;

if (i + 1 < size\_field && arr[i + 1][j] == -1)

a++;

if (i + 1 < size\_field && j + 1 < size\_field && arr[i + 1][j + 1] == -1)

a++;

arr[i][j] = a;

}

}

}

void spread(int i, int j)

{

if (i >= 0 && j >= 0 && i < size\_field && j < size\_field)

{

if (arr[i][j] != -1 && visible[i][j] != 1)

{

visible[i][j] = 1;

if (arr[i][j] == 0)

{

spread(i - 1, j - 1);

spread(i - 1, j);

spread(i - 1, j + 1);

spread(i, j - 1);

spread(i, j + 1);

spread(i + 1, j - 1);

spread(i + 1, j);

spread(i + 1, j + 1);

}

}

else;

}

else;

}

int finish()

{

int i, j;

for (i = 0; i < size\_field; i++)

for (j = 0; j < size\_field; j++)

{

if ((arr[i][j] != -1 && visible[i][j] == 0) || (arr[i][j] != -1 && visible[i][j] == 2))

return 0;

}

return 1;

}

void outline()

{

for (int i = 0; i <= size\_field; i++) {

bresen(i \* 500 / size\_field, 0, i \* 500 / size\_field, 500);

}

for (int i = 0; i <= size\_field; i++) {

bresen(0, i \* 500 / size\_field, 500, i \* 500 / size\_field);

}

}

void bomb(int x, int y)

{

glColor3f(0, 0, 0);

glBegin(GL\_POLYGON);

circle((500 / size\_field) / 10 \* 3, x, y);

glEnd();

glColor3f(0, 0, 0);

glBegin(GL\_POLYGON);

glVertex2f(x - (500 / size\_field) / 10, y + (500 / size\_field) / 5 + 5);

glVertex2f(x + (500 / size\_field) / 10, y + (500 / size\_field) / 5 + 5);

glVertex2f(x, y + (500 / size\_field) / 5 \* 2);

glEnd();

glBegin(GL\_POLYGON);

glVertex2f(x - (500 / size\_field) / 5 + 5, y - (500 / size\_field) / 10);

glVertex2f(x - (500 / size\_field) / 5 + 5, y + (500 / size\_field) / 10);

glVertex2f(x - (500 / size\_field) / 5 \* 2, y);

glEnd();

glBegin(GL\_POLYGON);

glVertex2f(x - (500 / size\_field) / 10, y - (500 / size\_field) / 5 + 5);

glVertex2f(x + (500 / size\_field) / 10, y - (500 / size\_field) / 5 + 5);

glVertex2f(x, y - (500 / size\_field) / 5 \* 2);

glEnd();

glBegin(GL\_POLYGON);

glVertex2f(x + (500 / size\_field) / 5 + 5, y - (500 / size\_field) / 10);

glVertex2f(x + (500 / size\_field) / 5 + 5, y + (500 / size\_field) / 10);

glVertex2f(x + (500 / size\_field) / 5 \* 2, y);

glEnd();

}

void setbomb(void)

{

int bomb[100][2];

int r, i, j, count, k;

srand(time(NULL));

for (count = 0; count < bombCount; count++)

{

manb:

r = rand() % (size\_field \* size\_field);

i = r / size\_field;

j = r % size\_field;

for (k = 0; k <= count - 1; k++)

{

if (i == bomb[k][0] && j == bomb[k][1])

goto manb;

else continue;

}

bomb[count][0] = i;

bomb[count][1] = j;

}

for (k = 0; k < bombCount; k++)

arr[bomb[k][0]][bomb[k][1]] = -1;

fill(arr, bomb, bombCount);

}

void help\_bot(int i, int j) {

is\_bomb = 0;

if (visible[i][j] != 1) {

int n, new\_j, new\_i, free = 0;

int add[3] = { -1, 0, 1 };

for (int y = 0; y <= 2; y++) {

for (int u = 0; u <= 2; u++) {

new\_j = j + add[u];

new\_i = i + add[y];

if (arr[i + add[y]][j + add[u]] >= 1 && visible[i + add[y]][j + add[u]] == 1 && arr[new\_i][new\_j] != 0 && !(add[y] == 0 && add[u] == 0)) {

n = arr[new\_i][new\_j];

for (int h = 0; h <= 2; h++) {

for (int g = 0; g <= 2; g++) {

if (add[h] == 0 && add[g] == 0) continue;

if (visible[new\_i + add[h]][new\_j + add[g]] == 1 || visible[new\_i + add[h]][new\_j + add[g]] == 2) free++;

if (bots\_answers[new\_i + add[h]][new\_j + add[g]] == 1) n--;

}

}

if (8 - free == n) {

bots\_answers[i][j] = 1;

}

if (n == 0) {

for (int h = 0; h <= 2; h++) {

for (int g = 0; g <= 2; g++) {

if (visible[new\_i + add[h]][new\_j + add[g]] == 0 && bots\_answers[new\_i + add[h]][new\_j + add[g]] != 1) bots\_answers[new\_i + add[h]][new\_j + add[g]] = -1;

}

}

}

free = 0;

}

}

}

}

}

void display(void)

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glClearColor(0.72, 0.72, 0.73, 0.0);

if (is\_bomb == 1) {

glColor3f(1, 1, 1);

bitmap(10, 530, GLUT\_BITMAP\_9\_BY\_15, "The bomb is here");

glPointSize(3);

glBegin(GL\_POINTS);

outline();

glEnd();

}

else if (is\_bomb == 0) {

glColor3f(1, 1, 1);

bitmap(10, 530, GLUT\_BITMAP\_9\_BY\_15, "Probably the bomb is here");

glPointSize(3);

glBegin(GL\_POINTS);

outline();

glEnd();

}

else if (is\_bomb == -1) {

glColor3f(1, 1, 1);

bitmap(10, 530, GLUT\_BITMAP\_9\_BY\_15, "The bomb isn't here");

glPointSize(3);

glBegin(GL\_POINTS);

outline();

glEnd();

}

if (out)

{

if (!over)

{

LPCTSTR pszSound1 = T\_STRING("C:\\Users\\Регина\\Downloads\\SOUNDm.wav");

BOOL result1 = PlaySound(pszSound1, NULL, SND\_FILENAME | SND\_ASYNC);

glColor3f(1, 1, 1);

int i, j;

for (i = 0; i < size\_field; i++)

for (j = 0; j < size\_field; j++)

{

if (visible[i][j] == 1)

{

glColor3f(1, 1, 1);

glBegin(GL\_POLYGON);

glVertex2f(j \* (500 / size\_field) + (500 / size\_field) / 10, 500 - (500 / size\_field) - i \* (500 / size\_field) + (500 / size\_field) / 10);

glVertex2f(j \* (500 / size\_field) + (500 / size\_field) \* 0.9, 500 - (500 / size\_field) - i \* (500 / size\_field) + (500 / size\_field) / 10);

glVertex2f(j \* (500 / size\_field) + (500 / size\_field) \* 0.9, 500 - (500 / size\_field) - i \* (500 / size\_field) + (500 / size\_field) \* 0.9);

glVertex2f(j \* (500 / size\_field) + (500 / size\_field) / 10, 500 - (500 / size\_field) - i \* (500 / size\_field) + (500 / size\_field) \* 0.9);

glEnd();

glColor3f(1, 1, 1);

if (arr[i][j] != 0)

{

char str[5];

\_itoa(arr[i][j], str, 10);

bitmap(j \* (500 / size\_field) + (500 / size\_field) / 5 \* 2, 500 - (500 / size\_field) - i \* (500 / size\_field) + (500 / size\_field) / 5 \* 2, GLUT\_BITMAP\_9\_BY\_15, str);

}

}

}

glColor3f(1, 1, 1);

glPointSize(3);

glBegin(GL\_POINTS);

outline();

glEnd();

bomb(a[1] \* (500 / size\_field) + (500 / size\_field) / 2, 500 - (500 / size\_field) - a[0] \* (500 / size\_field) + (500 / size\_field) / 2);

glFlush();

wait();

for (i = 0; i < size\_field; i++)

for (j = 0; j < size\_field; j++)

if (arr[i][j] == -1)

bomb(j \* (500 / size\_field) + (500 / size\_field) / 2, 500 - (500 / size\_field) - i \* (500 / size\_field) + (500 / size\_field) / 2);

glFlush();

wait();

}

glClear(GL\_COLOR\_BUFFER\_BIT);

glClearColor(0.3, 0.3, 1, 0);

glColor3f(1, 1, 1);

bitmap(200, 300, GLUT\_BITMAP\_9\_BY\_15, "YOU LOSE");

over = 1;

}

else

{

if (!over)

{

glColor3f(1, 1, 1);

bitmap(10, 550, GLUT\_BITMAP\_9\_BY\_15, "Time:");

float time = ((double)clock() - start) / CLOCKS\_PER\_SEC;

char str[100000];

\_itoa(time, str, 10);

bitmap(200, 550, GLUT\_BITMAP\_9\_BY\_15, str);

glPointSize(3);

glBegin(GL\_POINTS);

outline();

glEnd();

int i, j;

for (i = 0; i < size\_field; i++)

for (j = 0; j < size\_field; j++)

{

if (visible[i][j] == 1)

{

glColor3f(1, 1, 1);

glBegin(GL\_POLYGON);

glVertex2f(j \* (500 / size\_field) + (500 / size\_field) / 10, 500 - (500 / size\_field) - i \* (500 / size\_field) + (500 / size\_field) / 10);

glVertex2f(j \* (500 / size\_field) + (500 / size\_field) \* 0.9, 500 - (500 / size\_field) - i \* (500 / size\_field) + (500 / size\_field) / 10);

glVertex2f(j \* (500 / size\_field) + (500 / size\_field) \* 0.9, 500 - (500 / size\_field) - i \* (500 / size\_field) + (500 / size\_field) \* 0.9);

glVertex2f(j \* (500 / size\_field) + (500 / size\_field) / 10, 500 - (500 / size\_field) - i \* (500 / size\_field) + (500 / size\_field) \* 0.9);

glEnd();

glColor3f(0, 0, 0);

if (arr[i][j] != 0)

{

if (arr[i][j] == 1)

glColor3f(0, 0, 1);

else if (arr[i][j] == 2)

glColor3f(0, 1, 0);

else if (arr[i][j] == 3)

glColor3f(1, 0, 0);

else

glColor3f(0, 0, 0);

char str[5];

\_itoa(arr[i][j], str, 10);

bitmap(j \* (500 / size\_field) + (500 / size\_field) / 5 \* 2, 500 - (500 / size\_field) - i \* (500 / size\_field) + (500 / size\_field) / 5 \* 2, GLUT\_BITMAP\_9\_BY\_15, str);

}

}

else if (visible[i][j] == 2)

{

draw(j \* (500 / size\_field) + (500 / size\_field) / 5 \* 2, 500 - (500 / size\_field) - i \* (500 / size\_field) + (500 / size\_field) / 5);

}

}

glFlush();

}

if (finish())

{

wait();

glClear(GL\_COLOR\_BUFFER\_BIT);

glClearColor(0.3, 0.3, 1, 0);

glColor3f(1, 1, 1);

bitmap(200, 300, GLUT\_BITMAP\_9\_BY\_15, "YOU WON");

over = 1;

}

}

glFlush();

}

void mouse2(int mouse, int state, int x, int y)

{

if (!ent) goto man;

mouseVisit++;

int posi, posj;

posj = x / (500 / size\_field);

if (y >= 100)

posi = (y - 100) / (500 / size\_field);

else goto wrngpos;

if (visible[posi][posj] == 1)

goto man;

if (mouse == 2)

{

is\_bomb = -2;

if (visible[posi][posj] == 2 && mouseVisit % 2 == 1)

visible[posi][posj] = 0;

else if (mouseVisit % 2 == 1) visible[posi][posj] = 2;

}

else if (mouse == 1) {

help\_bot(posi, posj);

is\_bomb = bots\_answers[posi][posj];

}

else if (mouse == 0)

{

is\_bomb = -2;

if (arr[posi][posj] == -1)

{

out = 1; a[0] = posi; a[1] = posj;

}

else if (arr[posi][posj] == 0)

spread(posi, posj);

else

visible[posi][posj] = 1;

}

man:

wrngpos:

ent = 1;

glutDisplayFunc(display);

glutPostRedisplay();

}

void update(int value)

{

glutDisplayFunc(display);

glutPostRedisplay();

glutTimerFunc(1000, update, value);

}

void starting(void)

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(0.72, 0.72, 0.73);

glColor3f(0, 0, 0);

bitmap(180, 506, GLUT\_BITMAP\_9\_BY\_15, "Select a mode:");

bitmap(210, 416, GLUT\_BITMAP\_9\_BY\_15, "1.Easy");

bitmap(200, 316, GLUT\_BITMAP\_9\_BY\_15, "2.Medium");

bitmap(210, 216, GLUT\_BITMAP\_9\_BY\_15, "3.Hard");

glFlush();

}

void mouse1(int mouse, int state, int x, int y)

{

if (x >= 150 && x <= 350 && y >= 150 && y <= 200)

{

size\_field = 10;

bombCount = 8;

setbomb();

start = clock();

glutDisplayFunc(display);

glutMouseFunc(mouse2);

glutTimerFunc(500, update, 0);

glutPostRedisplay();

}

else if (x >= 150 && x <= 350 && y >= 250 && y <= 300)

{

size\_field = 20;

bombCount = 40;

setbomb();

start = clock();

glutDisplayFunc(display);

glutMouseFunc(mouse2);

glutTimerFunc(500, update, 0);

glutPostRedisplay();

}

else if (x >= 150 && x <= 350 && y >= 350 && y <= 400)

{

size\_field = 25;

bombCount = 90;

setbomb();

start = clock();

glutDisplayFunc(display);

glutMouseFunc(mouse2);

glutTimerFunc(500, update, 0);

glutPostRedisplay();

}

else

{

glutDisplayFunc(starting);

glutPostRedisplay();

}

}

int main(int argc, char\*\* argv)

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DEPTH | GLUT\_SINGLE | GLUT\_RGBA);

glutInitWindowPosition(250, 100);

glutInitWindowSize(500, 600);

glutCreateWindow("Minesweep");

Init();

glutDisplayFunc(starting);

glutMouseFunc(mouse1);

glutMainLoop();

return 0;

}