

Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное
учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Институт естественных и точных наук
Кафедра прикладной математики и программирования

Программа для моделирования взаимодействия объектов

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ИТОГОВОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
по дополнительной профессиональной программе переподготовки
«Методы программирования в компьютерной графике»

Автор работы,
студент группы КЭ-243
_____ / Г.О. Кузьмин
« ____ » _____ 2024 г.

Руководитель работы,
старший преподаватель
_____ / А.С. Шелудько
« ____ » _____ 2024 г.

Работа защищена с оценкой

« ____ » _____ 2024 г.

Челябинск 2024

АННОТАЦИЯ

Кузьмин Г.О. Программа для моделирования взаимодействия объектов. – Челябинск: ЮУрГУ, КЭ-243, 2024. – 50 с., 7 ил., библиогр. список – 5 наим., 3 прил.

Целью работы является разработка программы для моделирования взаимодействия объектов (далее - частиц). В разделе 1 приведены требования к интерфейсу и функционалу программы, а также схемы программного меню, области симуляции. В разделе 2 рассмотрена формализация задачи и описаны структуры данных, которые используются при программной реализации. В разделе 3 представлены принципы симуляции частиц, алгоритм работы программного меню. В разделе 4 приведено описание программных модулей, структур данных, классов, функций, переменных, констант. Текст программы, руководство пользователя и примеры выполнения программы приведены в приложениях.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ	3
1 ПОСТАНОВКА ЗАДАЧИ	4
2 ФОРМАЛИЗАЦИЯ ЗАДАЧИ	6
3 РАЗРАБОТКА АЛГОРИТМА.....	8
4 ОПИСАНИЕ ПРОГРАММЫ	10
ЗАКЛЮЧЕНИЕ	14
ЛИТЕРАТУРА	15
ПРИЛОЖЕНИЕ 1 Текст программы.....	16
ПРИЛОЖЕНИЕ 2 Руководство пользователя.....	40
ПРИЛОЖЕНИЕ 3 Примеры выполнения программы.....	41

ВВЕДЕНИЕ

Целью работы является разработка программы для моделирования взаимодействия объектов (далее - частиц). Для разработки используется язык программирования C++ (VC++ 22), его стандартная библиотека, а также графическая библиотека SDL2. В качестве системы сборки используется Сmake. Для достижения поставленной цели необходимо выполнить следующие этапы разработки:

- определить требования к интерфейсу и функционалу программы;
- провести формализацию задачи, определить как состояние симуляции будет представлено в цифровом виде;
- определить структуры данных, которые будут использоваться в программной реализации;
- разработать алгоритм обновления всех частиц в рамках области симуляции;
- написать, отладить и протестировать программу, которая реализует игровой процесс;
- реализовать программное меню.

1 ПОСТАНОВКА ЗАДАЧИ

Рассмотрим основные требования к интерфейсу программы, а также функциональные возможности, которые должны быть реализованы при разработке.

После запуска программы заполняется и начинается симуляция конечного автомата по законам Джона Конуэя в левой части экрана, интерактивные объекты интерфейса в правой части экрана содержат пункты «start», «exit» (рисунок 1.1). Выбор пункта меню осуществляется нажатием левой кнопки мыши.

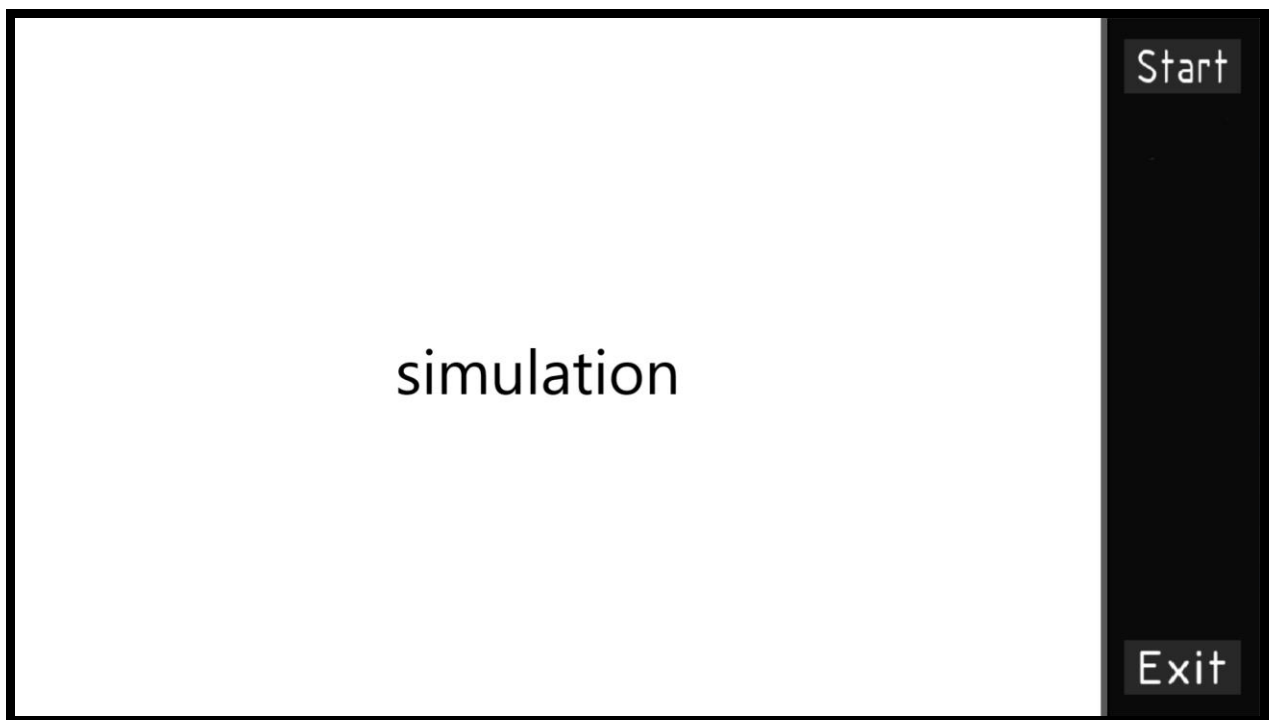


Рисунок 1.1 – Схема главного меню

При выборе пользователем пункта «start» появляется окно симуляции с возможностью взаимодействия (рисунок 1.2). В левой части окна производится симуляция.

Справа от поля симуляции расположены кнопки выбора типа частицы, «exit», «menu». При нажатии на кнопку с названием типа частицы происходит замена активной кисти рисования частиц. При нажатии на кнопку «pause» симуляция останавливается, кнопка становится «unpause». При нажатии на кнопку «menu» происходит возврат в главное меню.

Для разработки запланирована симуляция следующих частиц:

- Дерево. Воспламеняемо, не реагирует на гравитацию;
- Вода. Реагирует на гравитацию, в случае меньшей плотности нижней клетки, меняется с ней местами;
- Огонь. Поджигает воспламеняемые объекты, заменяет воду на пар;
- Сталь. Не реагирует на гравитацию, заменяет воду на пар в случае высокой температуры;
- Песок. Реагирует на гравитацию, рассыпается.

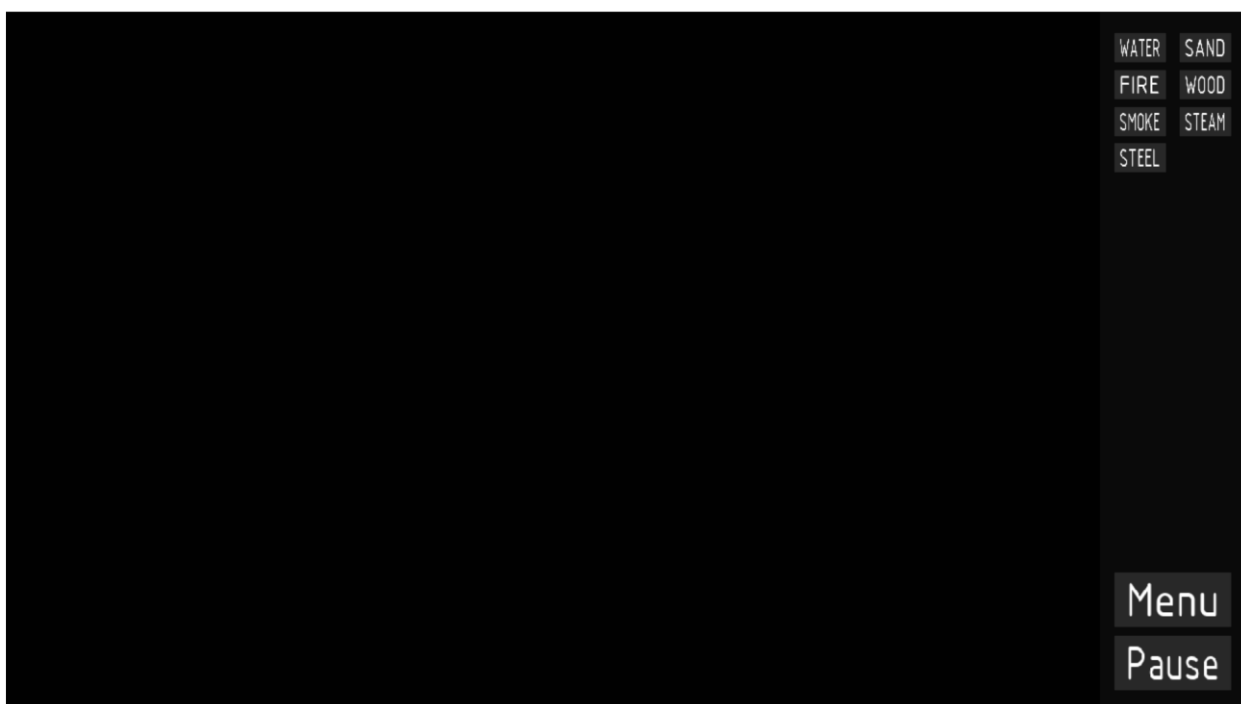


Рисунок 1.2 – Схема окна симуляции с возможностью взаимодействия

2 ФОРМАЛИЗАЦИЯ ЗАДАЧИ

Поле симуляции будет представлять собой двумерную матрицу ячеек, размера $x*y$, содержащих в себе информацию о своем состоянии, типе, и другой информации, необходимой для просчета алгоритма обновления.

Поле рисуется из квадратов, окрашенных в цвет, принадлежащий типу частицы. Ось X направлена вправо. Ось Y направлена вниз. Отсчет ведется с точки с координатами $(0;0)$ из верхнего левого угла.

Размер клеток неизменен и составляет 5 пикселей.

Координаты верхнего левого угла поля $(0; 0)$, соответственно координаты правого нижнего угла (ширина поля; высота поля).

Исходное состояние поля – пустое (рисунок 2.1).

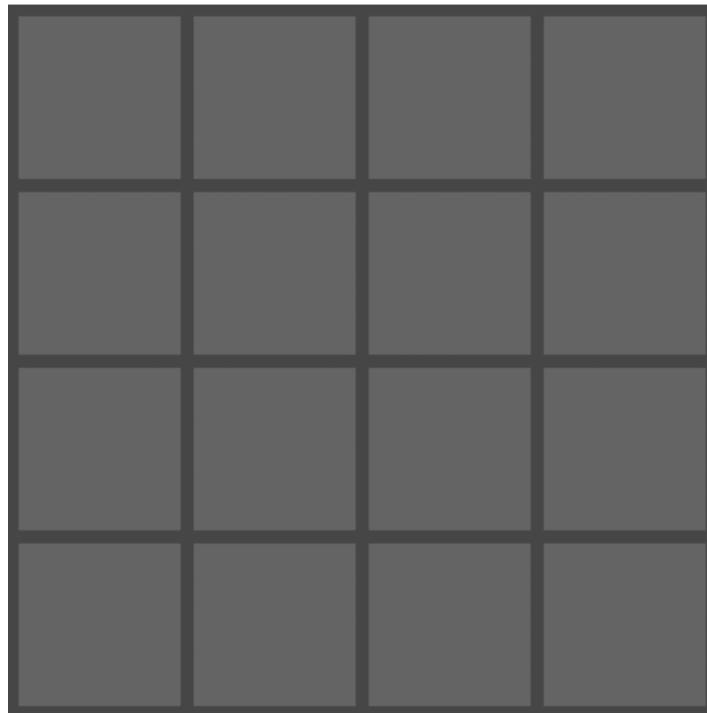


Рисунок 2.1 – Пример исходного поля

В случае нахождения частицы внутри поля следует отрисовать квадрат с нужным цветом(рисунок 2.2).

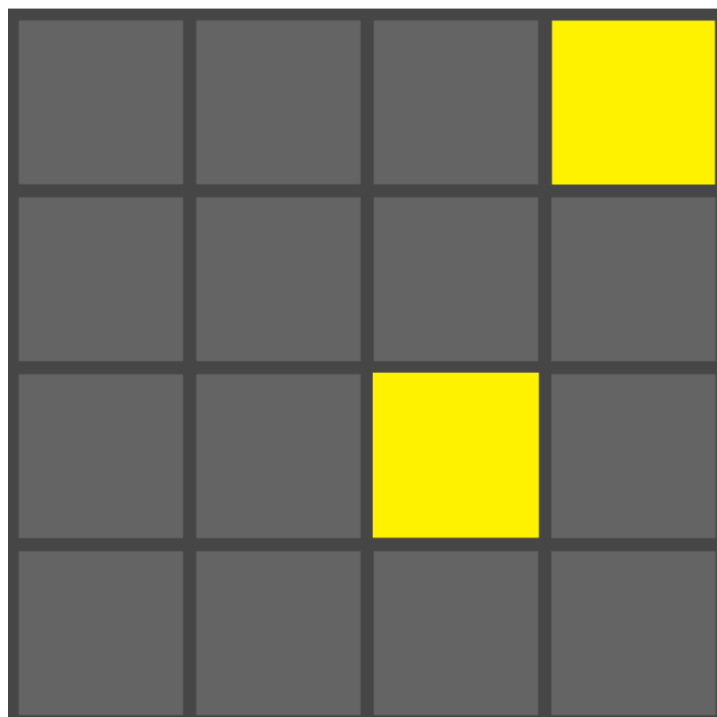


Рисунок 2.2 – Пример заполненного поля

Если частица подразумевает взаимодействие с окружением, например, падение песка под действием гравитации, следует произвести обновление состояние ячейки(рисунок 2.3).

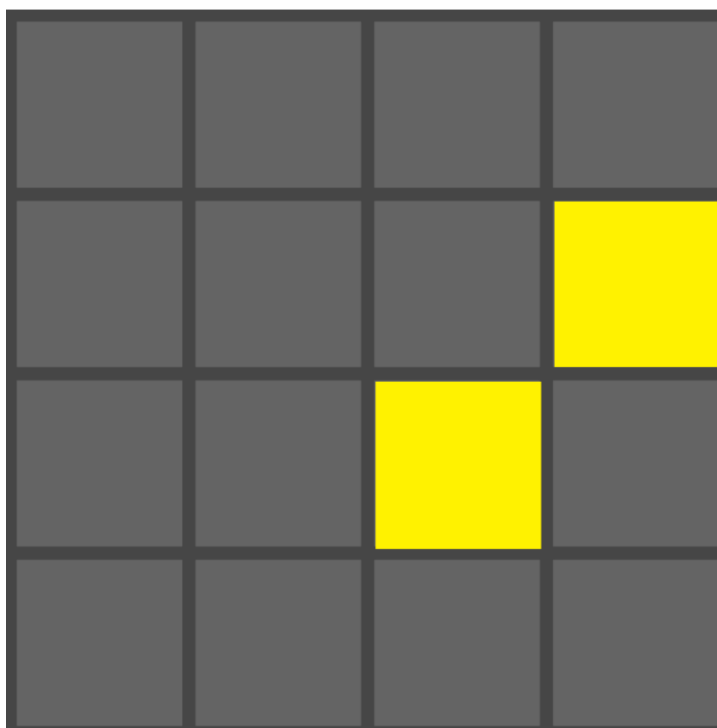


Рисунок 2.2 – Пример обновленного поля

3 РАЗРАБОТКА АЛГОРИТМА

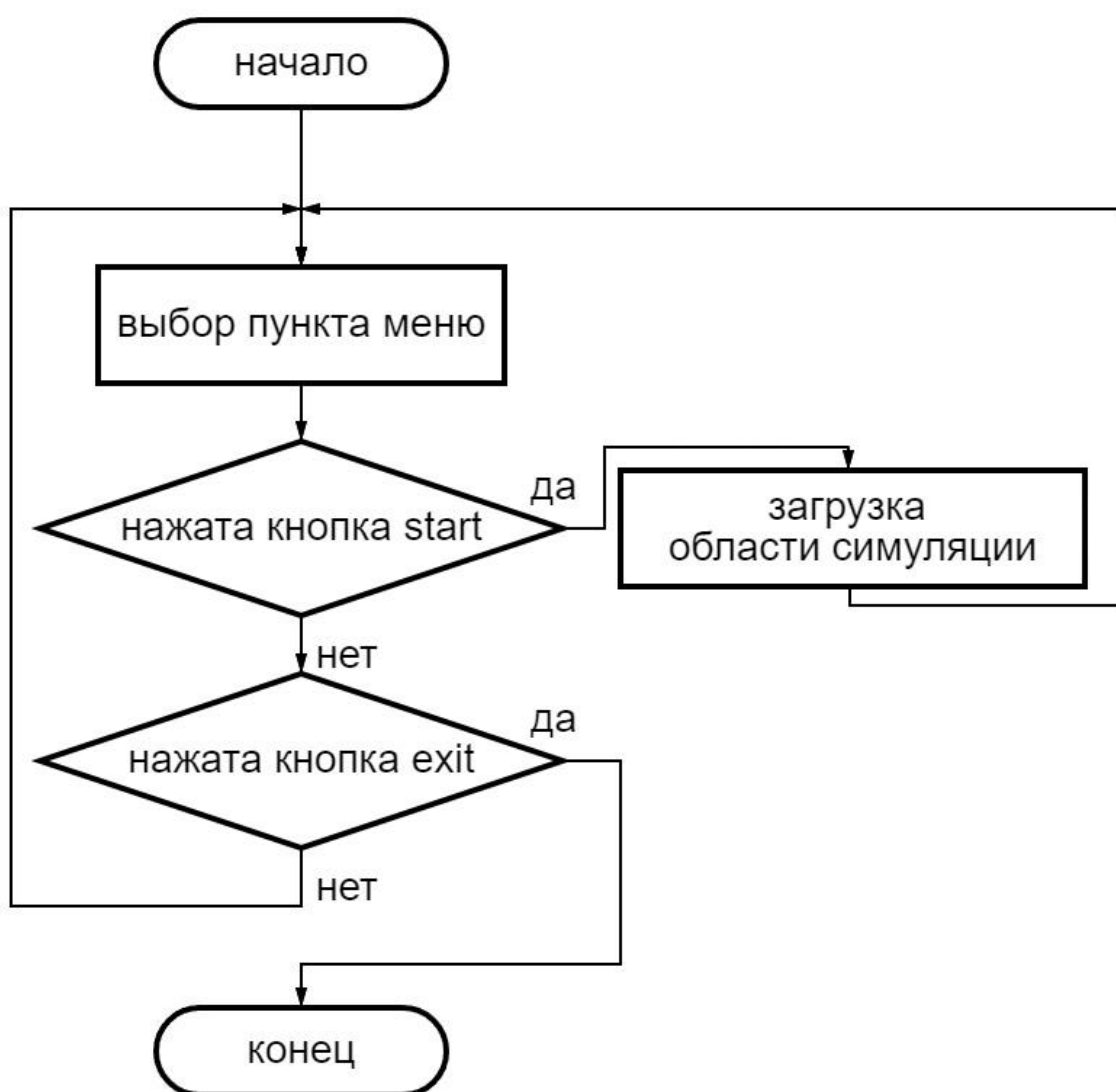


Рисунок 3.1 – алгоритм работы программного меню

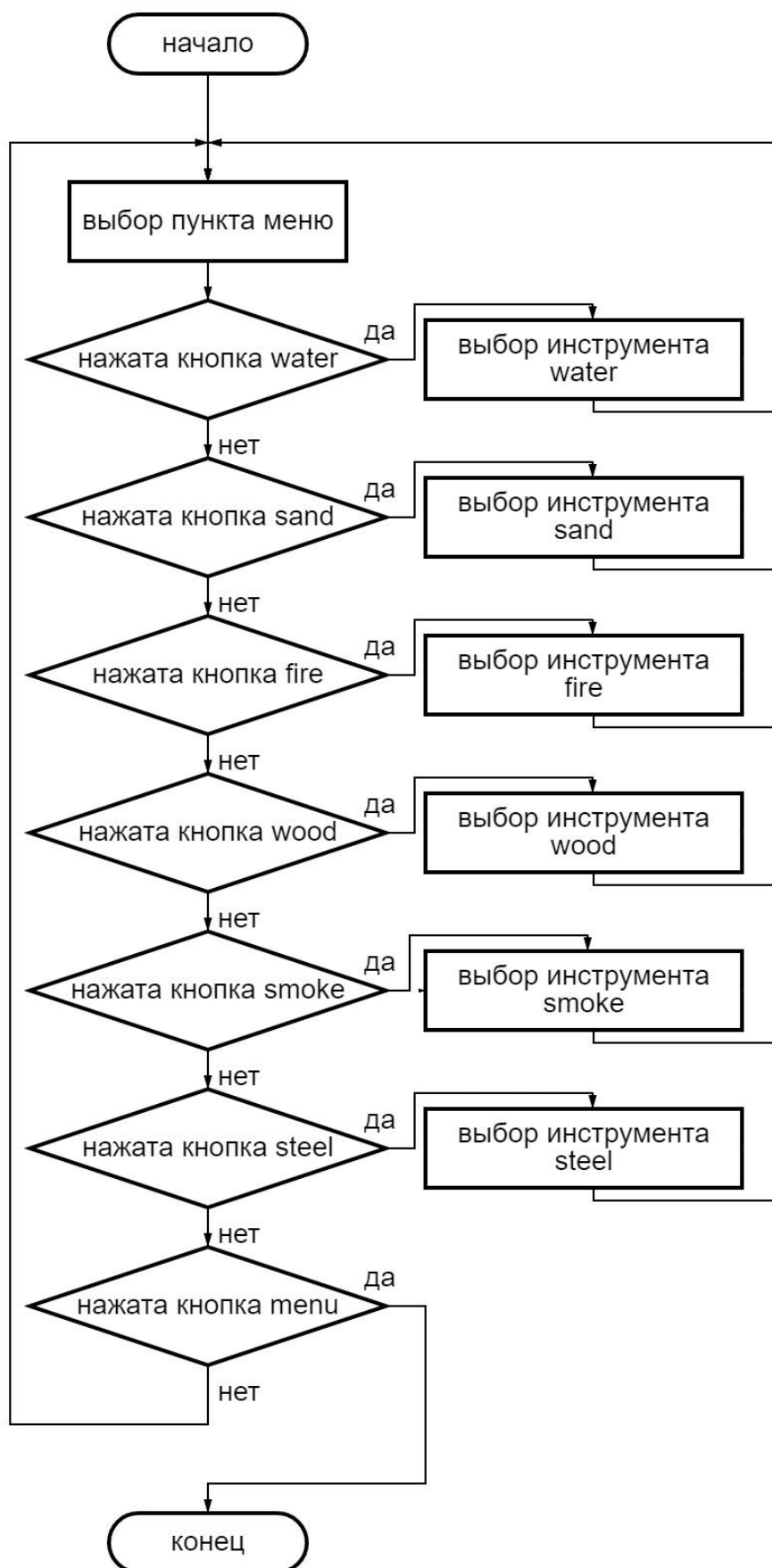


Рисунок 3.1 – алгоритм работы в процессе симуляции

4 ОПИСАНИЕ ПРОГРАММЫ

В реализации, программа разбита на 5 основных модулей: main.cpp, engine.cpp, world.cpp, cell.cpp, level.cpp. Каждый из модулей отвечает за определенную часть в программе.

Модуль main.cpp содержит точку входа, вызывает функции создания окна из модуля engine.cpp, поочередно, в бесконечном цикле обновления и отрисовки, вызывает соответствующие функции, в случае прерывания цикла завершает работу программы.

Модуль engine.cpp отвечает за инициализацию данных, Высвобождения памяти после завершения работы программы. отрисовывает объекты внутри окна, такие как кнопки и частицы с помощью метода render, который последовательно рисует квадраты размером 5 пикселей на экране, прямоугольники, кнопки. реагирует на действия пользователя с помощью метода handleEvents, который может среагировать лишь на нажатие ЛКМ или закрытие окна, остальные методы взаимодействия с программой не предусмотрены в случае регистрации нажатия на кнопку, выполняет соответствующие ей действия, в другом случае, наведения курсора на рабочую область, устанавливает в ней выбранную частицу (по умолчанию песок).

```
class game
{
    private:
        SDL_Window* window;
        SDL_Renderer* renderer;
        SDL_Event event;
        Level level;
        int windowHeight, windowWidth;
        int _currentPaintID;
    public:
        game(const char* p_title, int p_width, int p_height);
        void init();
        void handleEvents(bool &gameRunning);
        void update();
        void render();
        void cleanUp();
};
```

Модуль level.cpp отвечает за добавление, удаление, изменение объектов окна, загрузку другого окна, предоставляет другим модулям информацию об этих объектах.

```
class Level
{
private:
World _world;
vector<UIprimitive> _plains;
vector<UIprimitive> _buttons;
int _level;
public:
Level(int p_windowX, int p_windowY);
int getButtonCount();
int getPlainCount();
UIprimitive* getPlain(int p_i);
UIprimitive* getButton(int p_i);
World* getWorld();
void setColorFromVec(int p_i, int p_r, int p_g, int p_b, int
p_a);
void updateWorld();
void loadLevel(int levelID);
};
```

Модуль world.cpp отвечает за обновление симуляции как в главном меню, так и внутри рабочей области.

Каждое обновление симуляции, исходя из типа частицы, обрабатывается отдельный набор правил:

Для пустого пространства ничего не происходит;

Частицы песка стремятся упасть под силами гравитации.

Частица воды такая же как и песок, но более текучая.

Частица огня не падает, каждые 25мс пытается создать своего более теплого двойника выше себя, в случае затухания превращается в дым, стремятся поджечь все горящие частицы вокруг себя, при контакте с водой превращается в пар.

Частица дерева лишь воспламеняема, не подвержена гравитации.

Частица стали стремятся передать тепло своим соседям-сталям, при определенной температуре раскаляется и может испарять воду.

У всех частиц, подверженных гравитации есть общее свойство: если частица под текущей имеет меньшую плотность относительно текущей, то они меняются местами. Также все частицы не имеют права покидать рамки окна.

```
class World
{
    private:
        int height;
        int width;
        vector<vector<cell>> World_grid;
        vector<vector<cell>> World_1;
        vector<pair<int, int>> chekedPoints;
    public:
        World(unsigned int p_width, int p_height);
        World();
        cell getCell(int p_x, int p_y);
        void setCell(int p_x, int p_y, cell p_cell);
        void updateMenu();
        void updateGame();
        void randomFill();
        void erease();
};
```

Модуль cell.cpp передает другим модулям информацию о частицах и их поведении.

```
class cell
{
    private:
        int entityID;
        int lifeTime;
        int density;
        bool flamable = false;
        SDL_Color color;
    public:
        bool hasBeenUpdated = false;
        bool isMetal = false;
        int temp;
        cell(int p_entityID);
        int getDens();
        void reduceLifeTime(int p_lifeTime);
        void setID(int p_entityID);
        void setColor(SDL_Color p_color);
        int getEntityID();
        int getlifeTime();
        bool hasGravity();
        bool isFlamable();
        SDL_Color getColor();
};
```

ЗАКЛЮЧЕНИЕ

Поставленная задача для проекта была выполнена, однако из-за сжатых сроков пришлось отказаться от нескольких функций.

В ходе работы я изучил устройство конечных автоматов для создания симуляции игры в жизнь Джона Конуэя. Также изучил возможности библиотеки SDL2 и SDL2_TTF. SDL2_TTF является крайне неоптимизированной библиотекой, так как после ее добавления в проект и последующего использования производительность упала в 6 раз. Есть несколько методов оптимизации использования этой библиотеки, однако на это не хватило времени.

Удалось применить несколько принципов ООП, таких как: абстракция (модули имеют четкую иерархию, и вызов `main.game.init()` вызывает функции из всех модулей), сокрытие (классы из одного модуля не могут напрямую получить доступ к памяти к другим классам).

Алгоритмы обработки и хранения данных также стали полезными в ходе работы. Первые реализации алгоритмов симуляции были медленными, после оптимизации уже знакомыми мне методами симуляция стала выполняться с удовлетворительной скоростью на моей машине.

Программа хоть и является готовой, в нее еще можно добавить много функционала, такого как: реализация электропроводимости частиц, связывание частиц между собой, добавление столько типов частиц, сколько можно представить (разница в производительности между полным отсутствием симуляции и симуляции 8-и типов частиц минимальна, около 2%).

ЛИТЕРАТУРА

1. Керниган, Б. Практика программирования / Б. Керниган, Р. Пайк. – Москва : Вильямс, 2004. – 288 с.
2. Левитин, А. В. Алгоритмы: введение в разработку и анализ / А. В. Левитин. – Москва : Вильямс, 2006. – 576 с.
3. Липпман, С. Язык программирования C++. Полное руководство / С. Липпман, Ж. Лажоие. – Москва : ДМК Пресс, 2006. – 1105 с.
4. Никулин, Е. А. Компьютерная графика. Модели и алгоритмы : учебное пособие / Е. А. Никулин. – Санкт-Петербург : Лань, 2022. – 708 с.
5. Шилдт, Г. Полный справочник по C++ / Г. Шилдт. – Москва : Вильямс, 2006. – 800 с.

ПРИЛОЖЕНИЕ 1

Текст программы

Main.cpp

```
#include "src/engine.hpp"

using namespace std;

int main(int argc, char ** args)
{
    if (SDL_Init(SDL_INIT_EVERYTHING) != 0)
    { cout << "SDL_ERROR: " << SDL_GetError() << endl; }
    if (!(IMG_Init(IMG_INIT_PNG)))
    { cout << "IMG_init has failed. Error: " << SDL_GetError() << endl; }
    if ((TTF_Init() != 0))
    { cout << "TTF_init has failed. Error: " << SDL_GetError() << endl; }

    game game("game V0.1.1", 1280, 720);

    bool gameRunning = true;

    game.init();
    while(gameRunning)
    {
        Uint64 start = SDL_GetPerformanceCounter();
        game.update();
        game.handleEvents(gameRunning);
        game.render();
        Uint64 end = SDL_GetPerformanceCounter();
        //SDL_Delay(100);
        float elapsed=(end-start)/ (float)SDL_GetPerformanceFrequency();
        //cout << "Current FPS: " << to_string(1.0f / elapsed) << endl;
    }
    game.cleanUp();
    return 0;
}
```

Level.cpp

```
#include "level.hpp"

void pushPlain(vector<UIprimitive> *p_vector, int c_r, int c_g,
int c_b, int c_a, int r_x, int r_y, int r_w, int r_h)
{
    SDL_Color color;
    SDL_Rect rect;
    UIprimitive primitive;
    color.a = c_a;
    color.r = c_r;
    color.g = c_g;
    color.b = c_b;
    rect.h = r_h;
    rect.w = r_w;
    rect.x = r_x;
    rect.y = r_y;
    primitive.color = color;
    primitive.rect = rect;
    p_vector->push_back(primitive);
}

void pushButton(vector<UIprimitive> *p_vector, int c_r, int c_g,
int c_b, int c_a, int r_x, int r_y, int r_w, int r_h, string text)
{
    SDL_Color color;
    SDL_Rect rect;
    UIprimitive primitive;
    color.a = c_a;
    color.r = c_r;
    color.g = c_g;
    color.b = c_b;
    rect.h = r_h;
    rect.w = r_w;
    rect.x = r_x;
    rect.y = r_y;
    primitive.color = color;
    primitive.rect = rect;
    primitive.textButton = text;
    p_vector->push_back(primitive);
}

Level::Level(int p_windowX, int p_windowY)
    : _windowX(p_windowX), _windowY(p_windowY), _world(((p_windowX
- p_windowX/8)/5)+1, (p_windowY/5)+1)
{
    loadLevel(0);
};

void Level::loadLevel(int levelID)
{
    _world.erase();
    _buttons.clear();
    _plains.clear();
    pushPlain(&_amp;plains, 0, 0, 0, 255, 0, 0, _windowX, _windowY);
    // background
```

```

switch (levelID)
{
    case 0: // main menu
    {
        _world.randomFill();
        pushPlain(&_plains, 10, 10, 10, 255, _windowX -
(_windowX/8), 0, _windowX/8, _windowY); // right side bar
        pushButton(&_buttons, 40, 40, 40, 255, _windowX -
(( _windowX/8) - _windowX/64), _windowY/32, (_windowX/16) +
_windowX/32, _windowY/16 + _windowY/64, "Start"); // start button
        pushButton(&_buttons, 40, 40, 40, 255, _windowX -
(( _windowX/8) - _windowX/64), _windowY/8, (_windowX/16) + _windowX/32,
_windowY/16 + _windowY/64, "Settings"); // settings button
        pushButton(&_buttons, 40, 40, 40, 255, _windowX -
(( _windowX/8) - _windowX/64), _windowY - _windowY/8 + _windowY/64,
(_windowX/16) + _windowX/32, _windowY/16 + _windowY/64, "Exit"); //
exit button
        break;
    }
    case 1: // game itself
    {
        pushPlain(&_plains, 10, 10, 10, 255, _windowX -
(_windowX/8), 0, _windowX/8, _windowY); // right side bar
        pushButton(&_buttons, 40, 40, 40, 255, _windowX -
(( _windowX/8) - _windowX/64), _windowY - _windowY/5, (_windowX/16) +
_windowX/32, _windowY/16 + _windowY/64, "Menu"); // menu button
        pushButton(&_buttons, 40, 40, 40, 255, _windowX -
(( _windowX/8) - _windowX/64), _windowY - _windowY/8 + _windowY/64,
(_windowX/16) + _windowX/32, _windowY/16 + _windowY/64, "Pause"); //
exit button
        pushButton(&_buttons, 40, 40, 40, 255, _windowX -
(( _windowX/8) - _windowX/64), _windowY/32, _windowX/24, _windowY/24,
"WATER"); // water selection
        pushButton(&_buttons, 40, 40, 40, 255, _windowX -
(_windowX/17.7), _windowY/32, _windowX/24, _windowY/24, "SAND"); //
sand selection
        pushButton(&_buttons, 40, 40, 40, 255, _windowX -
(( _windowX/8) - _windowX/64), _windowY/12, _windowX/24, _windowY/24,
"FIRE"); // fire selection
        pushButton(&_buttons, 40, 40, 40, 255, _windowX -
(_windowX/17.7), _windowY/12, _windowX/24, _windowY/24, "WOOD"); //
wood selection
        pushButton(&_buttons, 40, 40, 40, 255, _windowX -
(( _windowX/8) - _windowX/64), _windowY/7.3, _windowX/24, _windowY/24,
"SMOKE"); // fire selection
        pushButton(&_buttons, 40, 40, 40, 255, _windowX -
(_windowX/17.7), _windowY/7.3, _windowX/24, _windowY/24, "STEAM"); //
wood selection
        pushButton(&_buttons, 40, 40, 40, 255, _windowX -
(( _windowX/8) - _windowX/64), _windowY/5.3, _windowX/24, _windowY/24,
"STEEL"); // fire selection
        //pushButton(&_buttons, 40, 40, 40, 255, _windowX -
(_windowX/17.7), _windowY/5.3, _windowX/24, _windowY/24, "STEAM"); //
wood selection
        break;
    }
}
}

```

```

        _level = levelID;
    }

void Level::updateWorld()
{
    switch (_level)
    {
        case 0:
        {
            _world.updateMenu();
            break;
        }
        case 1:
        {
            _world.updateGame();
            break;
        }
    }
}

int Level::getPlainCount()
{
    return _plains.size();
}

int Level::getButtonCount()
{
    return _buttons.size();
}

UIprimitive* Level::getPlain(int p_i)
{
    return &_plains[p_i];
}

UIprimitive* Level::getButton(int p_i)
{
    return &_buttons[p_i];
}

World* Level::getWorld()
{
    return &_world;
}

```

engine.cpp

```
#include "engine.hpp"

int mouseX, mouseY;
int g_windowWidth, g_windowHeight;
bool gamePaused = false;

enum string_code {
    eStart,
    eSettings,
    eMenu,
    ePauseSim,
    eUnPauseSim,
    eWATER,
    eWOOD,
    eSAND,
    eFIRE,
    eSMOKE,
    eSTEAM,
    eSTEEL,
    eXXX,
    eExit
};

string_code hashit (std::string const& inString) {
    if (inString == "Start") return eStart;
    if (inString == "Settings") return eSettings;
    if (inString == "Exit") return eExit;
    if (inString == "Pause") return ePauseSim;
    if (inString == "Unpause") return eUnPauseSim;
    if (inString == "SAND") return eSAND;
    if (inString == "WATER") return eWATER;
    if (inString == "WOOD") return eWOOD;
    if (inString == "FIRE") return eFIRE;
    if (inString == "SMOKE") return eSMOKE;
    if (inString == "STEAM") return eSTEAM;
    if (inString == "STEEL") return eSTEEL;
    if (inString == "XXX") return eXXX;
    if (inString == "Menu") return eMenu;
}

bool insideRect(SDL_Rect* rect, int x, int y)
{
    if ((rect->x < x && rect->w + rect->x > x) && (rect->y < y &&
rect->h + rect->y > y))
    {
        return true;
    }
    return false;
}

SDL_Texture* renderText(const std::string &message, const
std::string *fontFile,
    SDL_Color color, int fontSize, SDL_Renderer *renderer)
{
    //Открываем шрифт
```

```

        TTF_Font *font = TTF_OpenFont(fontFile->c_str(), font-
Size);
        //Сначала нужно отобразить на поверхность с помощью
TTF_RenderText,
        //затем загрузить поверхность в текстуру
        SDL_Surface *surf = TTF_RenderText_Blended(font, mes-
sage.c_str(), color);
        SDL_Texture *texture =
SDL_CreateTextureFromSurface(renderer, surf);
        //Очистка поверхности и шрифта
        SDL_FreeSurface(surf);
        TTF_CloseFont(font);
        return texture;
    }

    void renderPlane(SDL_Renderer *renderer, UIprimitive* p_plane)
    {
        SDL_Color color = p_plane->color;
        SDL_SetRenderDrawColor(renderer, color.r, color.g, color.b,
color.a);
        SDL_RenderFillRect(renderer, &p_plane->rect);
    }

    void handleButtonColor(UIprimitive* p_button, int x, int y)
    {
        if (!p_button->mouseOnMe)
        {
            if (insideRect(&p_button->rect, mouseX, mouseY))
            {
                p_button->mouseOnMe = true;
                p_button->color.r += 15;
                p_button->color.g += 15;
                p_button->color.b += 15;
                return;
            }
        }
        else
        {
            if(insideRect(&p_button->rect, mouseX, mouseY))
            {
                p_button->mouseOnMe = false;
                p_button->color.r -= 15;
                p_button->color.g -= 15;
                p_button->color.b -= 15;
                return;
            }
        }
    }

    void renderButton(SDL_Renderer *renderer, UIprimitive* p_button)
    {
        SDL_Color color = p_button->color;
        SDL_Texture* text = renderText(p_button->textButton,
&p_button->fontPath, {255, 255, 255, 255}, 50, renderer);
        SDL_Rect rect = p_button->rect;
        rect.x = rect.x + rect.w * 0.10;
        rect.y = rect.y + rect.h * 0.20;
        rect.h = rect.h*0.80;
    }

```

```

        rect.w = rect.w*0.80;
        handleButtonColor(p_button, mouseX, mouseY);
        SDL_SetRenderDrawColor(renderer, color.r, color.g, color.b,
color.a);
        SDL_RenderFillRect(renderer, &p_button->rect);
        SDL_RenderCopy(renderer, text, NULL, &rect);
        SDL_DestroyTexture(text);
    }

void handleExit(SDL_Event &event, bool &gameRunning)
{
    if (event.type == SDL_QUIT)
    {
        gameRunning = false;
    }
}

void handleButtons(SDL_Event &event, Level &level, bool
&gameRunning, int &tool)
{
    if(event.type == SDL_MOUSEBUTTONDOWN)
    {
        for(int i = 0; i < level.getButtonCount(); i++){
            switch (hashit(level.getButton(i)->textButton))
            {
                case eStart:
                {
                    if      (insideRect(&level.getButton(i)->rect,
mouseX, mouseY)){
                        level.loadLevel(1);}
                    break;
                }
                case eSettings:
                {
                    if      (insideRect(&level.getButton(i)->rect,
mouseX, mouseY)){
                        cout << "settings not implemented" <<
endl;}
                    break;
                }
                case ePauseSim:
                {
                    if      (insideRect(&level.getButton(i)->rect,
mouseX, mouseY)){
                        level.getButton(i)->textButton    =    "Un-
pause";
                        gamePaused = true;}
                    break;
                }
                case eUnPauseSim:
                {
                    if      (insideRect(&level.getButton(i)->rect,
mouseX, mouseY)){
                        level.getButton(i)->textButton = "Pause";
                        gamePaused = false;}
                    break;
                }
                case eMenu:

```

```

        {
            if      (insideRect(&level.getButton(i)->rect,
mouseX, mouseY)){
                level.loadLevel(0);
                break;
            }
            case eXXX:
            {
                if      (insideRect(&level.getButton(i)->rect,
mouseX, mouseY)){
                    tool = 1;}
                break;
            }
            case eSAND:
            {
                if      (insideRect(&level.getButton(i)->rect,
mouseX, mouseY)){
                    tool = 2;}
                break;
            }
            case eWATER:
            {
                if      (insideRect(&level.getButton(i)->rect,
mouseX, mouseY)){
                    tool = 3;}
                break;
            }
            case eWOOD:
            {
                if      (insideRect(&level.getButton(i)->rect,
mouseX, mouseY)){
                    tool = 4;}
                break;
            }
            case eFIRE:
            {
                if      (insideRect(&level.getButton(i)->rect,
mouseX, mouseY)){
                    tool = 5;}
                break;
            }
            case eSMOKE:
            {
                if      (insideRect(&level.getButton(i)->rect,
mouseX, mouseY)){
                    tool = 6;}
                break;
            }
            case eSTEAM:
            {
                if      (insideRect(&level.getButton(i)->rect,
mouseX, mouseY)){
                    tool = 7;}
                break;
            }
            case eSTEEL:
            {

```



```

        if      (insideRect(&level.getButton(i)->rect,
mouseX, mouseY)){
            tool = 8;}
        break;
    }
    case eExit:
    {
        if      (insideRect(&level.getButton(i)->rect,
mouseX, mouseY)){
            gameRunning = false;
        }
        break;
    }
}
}
}

void checkCellInsertion(Level* level, int p_paintID, SDL_Event
&event)
{
    if(SDL_GetMouseState(NULL, NULL) & SDL_BUTTON_LMASK)
    {
        if(mouseX < (g_windowWidth - g_windowWidth/8))
        {
            int cellPixelSizeX = g_windowWidth/level->getWorld()-
>getWidth();
            int    cellPixelSizeY    =    g_windowHeight/level-
>getWorld()->getHeight() + 1;
            int x = mouseX/cellPixelSizeX;
            int y = mouseY/cellPixelSizeY;
            level->getWorld()->setCell(x, y, cell(p_paintID));
            level->getWorld()->setCell(x+1, y, cell(p_paintID));
            level->getWorld()->setCell(x-1, y, cell(p_paintID));
            level->getWorld()->setCell(x, y+1, cell(p_paintID));
            level->getWorld()->setCell(x, y-1, cell(p_paintID));
        }
    }
}

game::game(const char* p_title, int p_width, int p_height)
:window(NULL), renderer(NULL), windowHeight(p_height), win-
dowWidth(p_width), level(p_width, p_height)
{
    _currentPaintID = 2;
    window = SDL_CreateWindow(p_title, SDL_WINDOWPOS_UNDEFINED,
SDL_WINDOWPOS_UNDEFINED, p_width, p_height, SDL_WINDOW_SHOWN);
    if (window == NULL)
    {
        std::cout << "Couldn't create window!" << std::endl;
    }

    renderer = SDL_CreateRenderer(window, -1,
SDL_RENDERER_ACCELERATED | SDL_RENDERER_PRESENTVSYNC);
}

void game::init()
{

```

```

    /// loading textures
    g_windowHeight = windowHeight;
    g_windowWidth = windowWidth;
    string texture_path;
    int texturesCount = 0;
    for(int i = 0; i < size(textures); i++)
    {
        texture_path = "res\\img\\" + to_string(i + 1) + ".jpg";
        textures[i] = IMG_LoadTexture(renderer, texture_path.c_str());
        if (textures[i] == NULL)
        {
            std::cout << "Failed to load texture. Error: " <<
SDL_GetError() << std::endl;
        }
        else { texturesCount++; }
    }
    cout << "Loaded " << texturesCount << " textures." << endl;

    SDL_GetWindowSize(window, &windowWidth, &windowHeight);
    srand(time(NULL));
}

void game::handleEvents(bool &gameRunning)
{
    SDL_GetMouseState(&mouseX, &mouseY);
    checkCellInsertion(&level, _currentPaintID, event);
    while(SDL_PollEvent(&event))
    {
        handleExit(event, gameRunning);
        handleButtons(event, level, gameRunning,
_currentPaintID);
    }
}

void game::update()
{
    _currentTick += 1;
    if (!gamePaused)
    {
        level.updateWorld();
    }
}

void game::render()
{
    SDL_RenderClear(renderer);

    SDL_Rect dst;
    dst.w = 5;
    dst.h = 5;
    for(int i = 0; i < level.getPlainCount(); i++){
        renderPlane(renderer, level.getPlain(i));
    }

    for(int i = 0; i < level.getButtonCount(); i++){

```

```

        handleButtonColor(level.getButton(i), mouseX, mouseY);
        renderButton(renderer, level.getButton(i));
    }

    SDL_Color pixelColor;
    for (int i = 0; i < level.getWorld()->getHeight() - 1; i++)
    {
        for (int j = 0; j < level.getWorld()->getWidth() - 1;
j++)
        {
            dst.x = 5 * j;
            dst.y = 5 * i;
            pixelColor = level.getWorld()-
>getCell(j,i).getColor();
            int id = level.getWorld()-
>getCell(j,i).getEntityID();

            SDL_SetRenderDrawColor(renderer, pixelColor.r, pix-
elColor.g, pixelColor.b, pixelColor.a);
            SDL_RenderFillRect(renderer, &dst);
            //SDL_RenderCopy(renderer, textures[id], NULL, &dst);
        }

        SDL_RenderPresent(renderer);
    }

    void game::cleanUp()
    {
        SDL_DestroyWindow(window);
        SDL_Quit();
    }

```

world.cpp

```
#include "world.hpp"

World::World(unsigned int p_width, int p_height)
{
    height = p_height + 1;
    width = p_width + 1;
    World_grid = {p_width, vector<cell>(height, cell(0))};
    World_l = {p_width, vector<cell>(width, cell(0))};
}

cell World::getCell(int p_x, int p_y)
{
    return World_grid[p_x][p_y];
}

void World::setCell(int p_x, int p_y, cell p_cell)
{
    World_grid[p_x][p_y] = p_cell;
}

int World::getHeight()
{
    return height;
}

int World::getWidth()
{
    return width;
}

void World::randomFill()
{
    for (int x = 0; x < width - 1; x++)
    {
        for (int y = 0; y < height - 1; y++)
        {
            World_grid[x][y] = cell((rand() % 2));
        }
    }
}

void World::updateMenu()
{
    bool C, R, L, D, U, RU, LU, RD, LD, flag;
    int count;
    for (int x = 1; x < World_grid.size() - 1; x++)
    {
        for (int y = 1; y < World_grid[0].size() - 1; y++)
        {
            U = World_grid[x][y+1].getEntityID();
            D = World_grid[x][y-1].getEntityID();
            R = World_grid[x+1][y].getEntityID();
            RU = World_grid[x+1][y+1].getEntityID();
            RD = World_grid[x+1][y-1].getEntityID();
            L = World_grid[x-1][y].getEntityID();
            LU = World_grid[x-1][y+1].getEntityID();
        }
    }
}
```

```

        LD = World_grid[x-1][y-1].getEntityID();
        count = R + RU + RD + L + LU + LD + D + U;
        if ((count == 3) && (World_grid[x][y].getEntityID()
== 0)) { World_1[x][y] = cell(1); }
        else if ((count == 3) &&
(World_grid[x][y].getEntityID() == 1)) { World_1[x][y] = cell(1); }
        else if ((count == 2) &&
(World_grid[x][y].getEntityID() == 1)) { World_1[x][y] = cell(1); }
        else { World_1[x][y] = cell(0); }
    }
}
for (int x = 0; x < World_grid.size(); x++)
{
    for (int y = 0; y < World_grid[0].size(); y++)
    {
        World_grid[x][y] = World_1[x][y];
    }
}

void World::updateGame()
{
    for (int x = 1; x < World_grid.size() - 1; x++)
    {
        for (int y = 0; y < World_grid[0].size() - 3; y++)
        {
            switch (World_grid[x][y].getEntityID())
            {
                //space
                case 0:
                {
                    World_grid[x][y].hasBeenUpdated = true;
                    break;
                }

                //white dot
                case 1:
                {
                    World_grid[x][y].hasBeenUpdated = true;
                    break;
                }

                //sand
                case 2:
                {
                    if (World_grid[x][y].hasBeenUpdated) { break;

World_grid[x][y].hasBeenUpdated = true;

//fall
if(World_grid[x][y+1].getEntityID() == 0){
    World_grid[x][y+1] = cell(2);
    World_grid[x][y+1].hasBeenUpdated = true;
    World_grid[x][y] = cell(0);
}
else if(World_grid[x + 1][y+1].getEntityID()
== 0){
    World_grid[x+1][y+1] = cell(2);

```

```

World_grid[x+1][y+1].hasBeenUpdated =
true;
World_grid[x][y] = cell(0);
}
else if(World_grid[x - 1][y+1].getEntityID()
== 0){
World_grid[x-1][y+1] = cell(2);
World_grid[x-1][y+1].hasBeenUpdated =
true;
World_grid[x][y] = cell(0);
}
//swap
else if(World_grid[x][y].getDens() >
World_grid[x][y+1].getDens()){
cell tempCell = World_grid[x][y+1];
World_grid[x][y+1] = World_grid[x][y];
World_grid[x][y+1].hasBeenUpdated = true;
World_grid[x][y] = tempCell;
}
else if(World_grid[x][y].getDens() >
World_grid[x+1][y+1].getDens()){
cell tempCell = World_grid[x+1][y+1];
World_grid[x+1][y+1] = World_grid[x][y];
World_grid[x+1][y+1].hasBeenUpdated =
true;
World_grid[x][y] = tempCell;
}
else if(World_grid[x][y].getDens() >
World_grid[x-1][y+1].getDens()){
cell tempCell = World_grid[x-1][y+1];
World_grid[x-1][y+1] = World_grid[x][y];
World_grid[x-1][y+1].hasBeenUpdated =
true;
World_grid[x][y] = tempCell;
}
}
break;
}
//water
case 3:
{
if (World_grid[x][y].hasBeenUpdated) { break;
}
World_grid[x][y].hasBeenUpdated = true;
if ((World_grid[x][y + 1].getEntityID() ==
0)){
World_grid[x][y + 1] = cell(3);
World_grid[x][y+1].hasBeenUpdated = true;
World_grid[x][y] = cell(0);
}
else if ((World_grid[x + 1][y +
1].getEntityID() == 0)){
World_grid[x + 1][y + 1] = cell(3);
World_grid[x+1][y+1].hasBeenUpdated =
true;
World_grid[x][y] = cell(0);
}
}

```

```

else if ((World_grid[x - 1][y +
1].getEntityID() == 0)){
    World_grid[x - 1][y + 1] = cell(3);
    World_grid[x-1][y+1].hasBeenUpdated =
true;

    World_grid[x][y] = cell(0);
}
else if (rand() % 2 == 0)
{
    if (World_grid[x + 1][y].getEntityID() ==
0){
        World_grid[x + 1][y] = cell(3);
        World_grid[x+1][y].hasBeenUpdated =
true;

        World_grid[x][y] = cell(0);
    }
}
else if (World_grid[x - 1][y].getEntityID()
== 0){
    World_grid[x - 1][y] = cell(3);
    World_grid[x-1][y].hasBeenUpdated = true;
    World_grid[x][y] = cell(0);
}

//swap
else if (World_grid[x][y].getDens() >
World_grid[x][y+1].getDens()){
    cell tempCell = World_grid[x][y+1];
    World_grid[x][y+1] = World_grid[x][y];
    World_grid[x][y+1].hasBeenUpdated = true;
    World_grid[x][y] = tempCell;
}
else if (World_grid[x][y].getDens() >
World_grid[x+1][y+1].getDens()){
    cell tempCell = World_grid[x+1][y+1];
    World_grid[x+1][y+1] = World_grid[x][y];
    World_grid[x+1][y+1].hasBeenUpdated =
true;

    World_grid[x][y] = tempCell;
}
else if (World_grid[x][y].getDens() >
World_grid[x-1][y+1].getDens()){
    cell tempCell = World_grid[x-1][y+1];
    World_grid[x-1][y+1] = World_grid[x][y];
    World_grid[x-1][y+1].hasBeenUpdated =
true;

    World_grid[x][y] = tempCell;
}
break;
}

//wood
case 4:
{
    World_grid[x][y].hasBeenUpdated = true;
    break;
}

```

```

//fire
case 5:
{
    World_grid[x][y].reduceLifeTime(1);
    if (World_grid[x][y].hasBeenUpdated) { break;
}

    World_grid[x][y].hasBeenUpdated = true;

    if(World_grid[x][y].getlifeTime() <= 0){
        if(rand() % 5 == 0){
            World_grid[x][y] = cell(6);
        }
        break;
    }

// fire time
if(World_grid[x][y].getlifeTime() % 20 == 0){
    if(World_grid[x][y-1].getEntityID() == 0
&& (rand() % 3 == 0)){
        World_grid[x][y-1] = cell(5);
        World_grid[x][y-
1].reduceLifeTime(300);
        World_grid[x][y-1].hasBeenUpdated =
true;
    }
}
if(World_grid[x][y].getlifeTime() % 20 == 0){
    if(World_grid[x+1][y-1].getEntityID() ==
0 && (rand() % 3 == 0)){
        World_grid[x+1][y-1] = cell(5);
        World_grid[x+1][y-
1].reduceLifeTime(300);
        World_grid[x+1][y-1].hasBeenUpdated =
true;
    }
}
if(World_grid[x][y].getlifeTime() % 20 == 0){
    if(World_grid[x-1][y-1].getEntityID() ==
0 && (rand() % 3 == 0)){
        World_grid[x-1][y-1] = cell(5);
        World_grid[x-1][y-
1].reduceLifeTime(300);
        World_grid[x-1][y-1].hasBeenUpdated =
true;
    }
}

/*
if(World_grid[x][y].getlifeTime() % 300 ==
0){
    if(World_grid[x][y-1].getEntityID() ==
0){
        World_grid[x][y-1] = cell(6);
        World_grid[x][y-1].hasBeenUpdated =
true;
    }
}

```



```

else if(World_grid[x+1][y].getEntityID()
== 0){
    World_grid[x+1][y] = cell(6);
    World_grid[x+1][y].hasBeenUpdated =
true;
}
else if(World_grid[x-1][y].getEntityID()
== 0){
    World_grid[x-1][y] = cell(6);
    World_grid[x-1][y].hasBeenUpdated =
true;
}
}
*/
if(World_grid[x][y+1].getEntityID() == 3){
    World_grid[x][y] = cell(7);
}
else if(World_grid[x][y-1].getEntityID() ==
3){
    World_grid[x][y] = cell(7);
}
else if(World_grid[x+1][y].getEntityID() ==
3){
    World_grid[x][y] = cell(7);
}
else if(World_grid[x-1][y].getEntityID() ==
3){
    World_grid[x][y] = cell(7);
}

// ignitor
if(World_grid[x][y+1].isFlamable()){
    if(rand() % 15 == 0){
        World_grid[x][y+1] = cell(5);
        World_grid[x][y+1].hasBeenUpdated =
true;
    }
}
if(World_grid[x][y-1].isFlamable()){
    if(rand() % 15 == 0){
        World_grid[x][y+1] = cell(5);
        World_grid[x][y+1].hasBeenUpdated =
true;
    }
}
if(World_grid[x+1][y].isFlamable()){
    if(rand() % 15 == 0){
        World_grid[x+1][y] = cell(5);
        World_grid[x+1][y].hasBeenUpdated =
true;
    }
}
if(World_grid[x-1][y].isFlamable()){
    if(rand() % 15 == 0){
        World_grid[x-1][y] = cell(5);
        World_grid[x-1][y].hasBeenUpdated =
true;
    }
}

```

```

    }

    // hot
    if(World_grid[x][y-1].isMetal){
        World_grid[x][y-1].temp += 20;
    }
    if(World_grid[x+1][y-1].isMetal){
        World_grid[x+1][y-1].temp += 20;
    }
    if(World_grid[x-1][y-1].isMetal){
        World_grid[x-1][y-1].temp += 20;
    }

    break;
}

//smoke
case 6:
{
    if (World_grid[x][y].hasBeenUpdated) { break;
}

    World_grid[x][y].hasBeenUpdated = true;
    if ((World_grid[x][y - 1].getEntityID() == 0
    && (World_grid[x][y - 1].getEntityID() == 0))) {
        World_grid[x][y - 1] = cell(6);
        World_grid[x][y-1].hasBeenUpdated = true;
        World_grid[x][y] = cell(0);
    }
    else if ((World_grid[x + 1][y -
1].getEntityID() == 0 && (World_grid[x + 1][y - 1].getEntityID() ==
0))) {
        World_grid[x + 1][y - 1] = cell(6);
        World_grid[x+1][y-1].hasBeenUpdated =
true;

        World_grid[x][y] = cell(0);
    }
    else if ((World_grid[x - 1][y -
1].getEntityID() == 0 && (World_grid[x - 1][y - 1].getEntityID() ==
0))) {
        World_grid[x - 1][y - 1] = cell(6);
        World_grid[x-1][y-1].hasBeenUpdated =
true;

        World_grid[x][y] = cell(0);
    }
    else if (rand() % 2 == 0)
    {
        if (World_grid[x + 1][y].getEntityID() ==
0 && World_grid[x + 1][y].getEntityID() == 0) {
            World_grid[x + 1][y] = cell(6);
            World_grid[x+1][y].hasBeenUpdated =
true;

            World_grid[x][y] = cell(0);
        }
    }
    else if (World_grid[x - 1][y].getEntityID()
== 0 && World_grid[x - 1][y].getEntityID() == 0) {
        World_grid[x - 1][y] = cell(6);
        World_grid[x-1][y].hasBeenUpdated = true;

```

```

        World_grid[x][y] = cell(0);
    }

    //swap
    else if(World_grid[x][y].getDens() >
World_grid[x][y-1].getDens()){
        cell tempCell = World_grid[x][y-1];
        World_grid[x][y-1] = World_grid[x][y];
        World_grid[x][y-1].hasBeenUpdated = true;
        World_grid[x][y] = tempCell;
    }
    else if(World_grid[x][y].getDens() >
World_grid[x+1][y-1].getDens()){
        cell tempCell = World_grid[x+1][y-1];
        World_grid[x+1][y-1] = World_grid[x][y];
        World_grid[x+1][y-1].hasBeenUpdated =
true;

        World_grid[x][y] = tempCell;
    }
    else if(World_grid[x][y].getDens() >
World_grid[x-1][y-1].getDens()){
        cell tempCell = World_grid[x-1][y-1];
        World_grid[x-1][y-1] = World_grid[x][y];
        World_grid[x-1][y-1].hasBeenUpdated =
true;

        World_grid[x][y] = tempCell;
    }

    // hot
    if(World_grid[x][y-1].isMetal){
        World_grid[x][y-1].temp += 2;
    }
    if(World_grid[x+1][y-1].isMetal){
        World_grid[x+1][y-1].temp += 2;
    }
    if(World_grid[x-1][y-1].isMetal){
        World_grid[x-1][y-1].temp += 2;
    }
    break;
}

//steam
case 7:
{
    World_grid[x][y].reduceLifeTime(1);
    if (World_grid[x][y].hasBeenUpdated) { break;
}

    World_grid[x][y].hasBeenUpdated = true;
    if (World_grid[x][y].getlifeTime() <= 0){
        World_grid[x][y] = cell(3);
    }

    if ((World_grid[x][y - 1].getEntityID() ==
0)){

        World_grid[x][y - 1] = World_grid[x][y];
        World_grid[x][y-1].hasBeenUpdated = true;
        World_grid[x][y] = cell(0);
    }
}

```

```

else if ((World_grid[x + 1][y - 1].getEntityID() == 0)){
    World_grid[x + 1][y - 1] =
    World_grid[x+1][y-1].hasBeenUpdated =
    true;
    World_grid[x][y] = cell(0);
}
else if ((World_grid[x - 1][y - 1].getEntityID() == 0)){
    World_grid[x - 1][y - 1] =
    World_grid[x-1][y-1].hasBeenUpdated =
    true;
    World_grid[x][y] = cell(0);
}
else if (rand() % 2 == 0)
{
    if (World_grid[x + 1][y].getEntityID() ==
0){
        World_grid[x + 1][y] =
        World_grid[x+1][y].hasBeenUpdated =
        true;
        World_grid[x][y] = cell(0);
    }
}
else if (World_grid[x - 1][y].getEntityID()
== 0){
    World_grid[x - 1][y] = World_grid[x][y];
    World_grid[x-1][y].hasBeenUpdated = true;
    World_grid[x][y] = cell(0);
}
//swap
else if (World_grid[x][y].getDens() >
World_grid[x][y-1].getDens()){
    cell tempCell = World_grid[x][y-1];
    World_grid[x][y-1] = World_grid[x][y];
    World_grid[x][y-1].hasBeenUpdated = true;
    World_grid[x][y] = tempCell;
}
else if (World_grid[x][y].getDens() >
World_grid[x+1][y-1].getDens()){
    cell tempCell = World_grid[x+1][y-1];
    World_grid[x+1][y-1] = World_grid[x][y];
    World_grid[x+1][y-1].hasBeenUpdated =
true;
    World_grid[x][y] = tempCell;
}
else if (World_grid[x][y].getDens() >
World_grid[x-1][y-1].getDens()){
    cell tempCell = World_grid[x-1][y-1];
    World_grid[x-1][y-1] = World_grid[x][y];
    World_grid[x-1][y-1].hasBeenUpdated =
true;
    World_grid[x][y] = tempCell;
}
break;

```

```

    }
    case 8:
    {
        if(World_grid[x][y].temp > 0 && (rand() % 4
==0)){
            if(World_grid[x][y+1].getEntityID() !=
8){
                World_grid[x][y].temp -= 1;
            }
            else if(World_grid[x][y-1].getEntityID()
!= 8){
                World_grid[x][y].temp -= 1;
            }
            else if(World_grid[x+1][y].getEntityID()
!= 8){
                World_grid[x][y].temp -= 1;
            }
            else if(World_grid[x-1][y].getEntityID()
!= 8){
                World_grid[x][y].temp -= 1;
            }
        }
        if (World_grid[x][y].hasBeenUpdated) { break;
}
        World_grid[x][y].hasBeenUpdated = true;

        // vaporize water
        if(World_grid[x][y].temp > 100){
            World_grid[x][y].setColor({158, 121, 103,
255});
            if(World_grid[x][y-1].getEntityID() ==
3){
                World_grid[x][y-1] = cell(7);
                World_grid[x][y-1].hasBeenUpdated =
true;
            }
            else if(World_grid[x+1][y-
1].getEntityID() == 3){
                World_grid[x+1][y-1] = cell(7);
                World_grid[x+1][y-1].hasBeenUpdated =
true;
            }
            else if(World_grid[x-1][y-
1].getEntityID() == 3){
                World_grid[x-1][y-1] = cell(7);
                World_grid[x-1][y-1].hasBeenUpdated =
true;
            }
        }
        else {World_grid[x][y].setColor({128, 142,
181, 255});}

        // set the limits
        if(World_grid[x][y].temp > 300) {
World_grid[x][y].temp = 300; }

        // share temp

```

```

        if((World_grid[x][y-1].getEntityID() == 8) &&
(World_grid[x][y].temp > World_grid[x][y-1].temp)){
            if(rand() % 4 == 0){
                World_grid[x][y].temp -= 10;
                World_grid[x][y-1].temp += 10;
            }
        }
        if((World_grid[x][y+1].getEntityID() == 8) &&
(World_grid[x][y].temp > World_grid[x][y+1].temp)){
            if(rand() % 4 == 0){
                World_grid[x][y].temp -= 10;
                World_grid[x][y+1].temp += 10;
            }
        }
        if((World_grid[x+1][y].getEntityID() == 8) &&
(World_grid[x][y].temp > World_grid[x+1][y].temp)){
            if(rand() % 4 == 0){
                World_grid[x][y].temp -= 10;
                World_grid[x+1][y].temp += 10;
            }
        }
        if((World_grid[x-1][y].getEntityID() == 8) &&
(World_grid[x][y].temp > World_grid[x-1][y].temp)){
            if(rand() % 4 == 0){
                World_grid[x][y].temp -= 10;
                World_grid[x-1][y].temp += 10;
            }
        }
        }
        break;
    }
}

}

for (int x = 1; x < World_grid.size() - 1; x++)
{
    for (int y = 1; y < World_grid[0].size() - 1; y++)
    {
        World_grid[x][y].hasBeenUpdated = false;
    }
}
//World_grid = World_1;
}

void World::erease()
{
    for (int x = 0; x < World_grid.size(); x++)
    {
        for (int y = 0; y < World_grid[0].size(); y++)
        {
            World_grid[x][y] = cell(0);
        }
    }
    for (int x = 0; x < World_1.size(); x++)
    {
        for (int y = 0; y < World_1[0].size(); y++)
        {
            World_1[x][y] = cell(0);
        }
    }
}

```

cell.cpp

```
#include "cell.hpp"
cell::cell(int p_entityID)
{
    switch (p_entityID)
    {
        case 0: // 0 - space
        {
            entityID = 0;
            density = 0;
            color = {1, 1, 1, 255};
            break;
        }
        case 1: // 1 - white dot
        {
            entityID = 1;
            density = 999;
            color = {255, 255, 255, 255};
            break;
        }
        case 2: // 2 - sand
        {
            entityID = 2;
            density = 30;
            color = {255, 255, 0, 255};
            break;
        }
        case 3: // 3 - water
        {
            entityID = 3;
            density = 10;
            color = {52, 79, 254, 255};
            break;
        }
        case 4: // 4 - wood
        {
            entityID = 4;
            density = 15;
            color = {146, 109, 5, 255};
            flamable = true;
            break;
        }
        case 5: // 5 - fire
        {
            entityID = 5;
            lifeTime = 340;
            density = 0;
            color = {205, 72, 41, 255};
            break;
        }
        case 6: // 6 - smoke
        {
            entityID = 6;
            density = 4;
            color = {92, 92, 92, 255};
        }
    }
}
```

```

        break;
    }
    case 7: // 7 - steam
    {
        entityID = 7;
        density = 5;
        lifeTime = 240;
        color = {74, 81, 101, 255};
        break;
    }
    case 8: // 8 - steel
    {
        entityID = 8;
        density = 40;
        isMetal = true;
        temp = 0;
        lifeTime = 240;
        color = {128, 142, 181, 255};
        break;
    }
}

int cell::getEntityID()
{
    return entityID;
}

int cell::getlifeTime()
{
    return lifeTime;
}

int cell::getDens()
{
    return density;
}

void cell::setID(int p_entityID)
{
    entityID = p_entityID;
}

void cell::reduceLifeTime(int p_lifeTime)
{
    lifeTime -= p_lifeTime;
}

bool cell::isFlamable()
{
    return flamable;
}

void cell::setColor(SDL_Color p_color)
{
    color = p_color;
}

```


ПРИЛОЖЕНИЕ 2

Руководство пользователя

После запуска программы следует нажать кнопку `start`, которая отправит вас в рабочую область, где вы сможете редактировать, наблюдать, останавливать, возобновлять симуляцию частиц.

Чтобы выйти из программы нажмите кнопку `menu->exit`, если вы находитесь в рабочей области, или `exit`, если вы находитесь в главном меню.

Для остановки симуляции нажмите кнопку `pause`. В остановленную симуляцию все еще можно вносить изменения.

Для возобновления симуляции нажмите кнопку `unpause`.

По умолчанию ваш инструмент – песок, наведите курсор в рабочую область и нажмите ЛКМ, чтобы добавить текущий тип объекта в симуляцию.

Чтобы поменять текущий тип объекта, нажмите на одну из кнопок в правом верхнем углу программы.

ПРИЛОЖЕНИЕ 3

Примеры выполнения программы



Рисунок П3.1 – Процесс взаимодействия с симуляцией

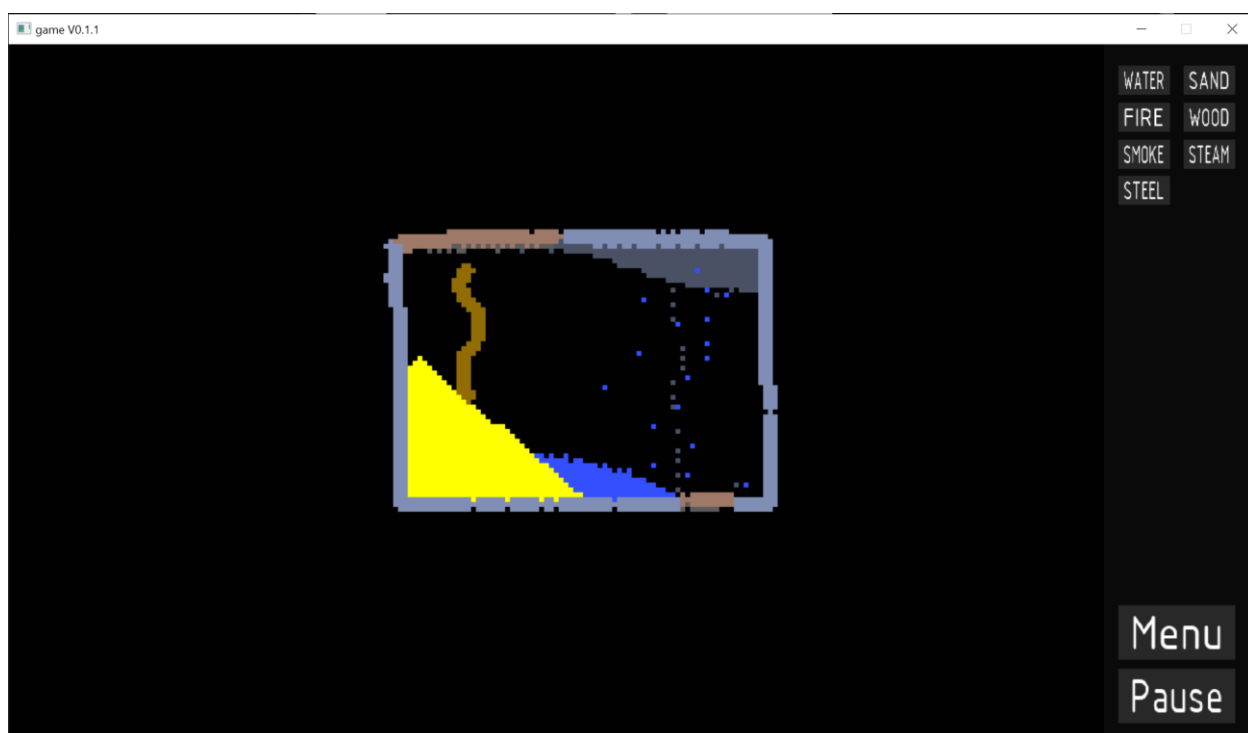


Рисунок П3.2 – Окно ввода данных