

МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ  
М. В. ЛОМОНОСОВА  
Факультет вычислительной математики и кибернетики  
Кафедра системного анализа

Отчёт и документация по заданию

# «Компьютерная графика: Интерактивная 2D-графика на языке C++ »

Вариант Б3

*Студент 315 группы*  
А. Т. Айтеев

Москва, 2020

## Содержание

<b>1</b>	<b>Предварительная подготовка для написания кода</b>	<b>3</b>
1.1	Скачать и установить шаблон . . . . .	3
1.2	Установка необходимых пакетов для работы . . . . .	3
1.3	Компиляция . . . . .	4
<b>2</b>	<b>Набор необходимых ресурсов</b>	<b>8</b>
2.1	GLFW User Guide . . . . .	8
<b>3</b>	<b>Как компилировать и запускать готовое задание</b>	<b>9</b>
3.1	Как запускать программу . . . . .	9
<b>4</b>	<b>Реализованные механики</b>	<b>10</b>
<b>5</b>	<b>Используемые алгоритмы</b>	<b>12</b>
5.1	Размеры блоков и игрока . . . . .	12
5.2	Границы перемещения игрока . . . . .	12

# 1 Предварительная подготовка для написания кода

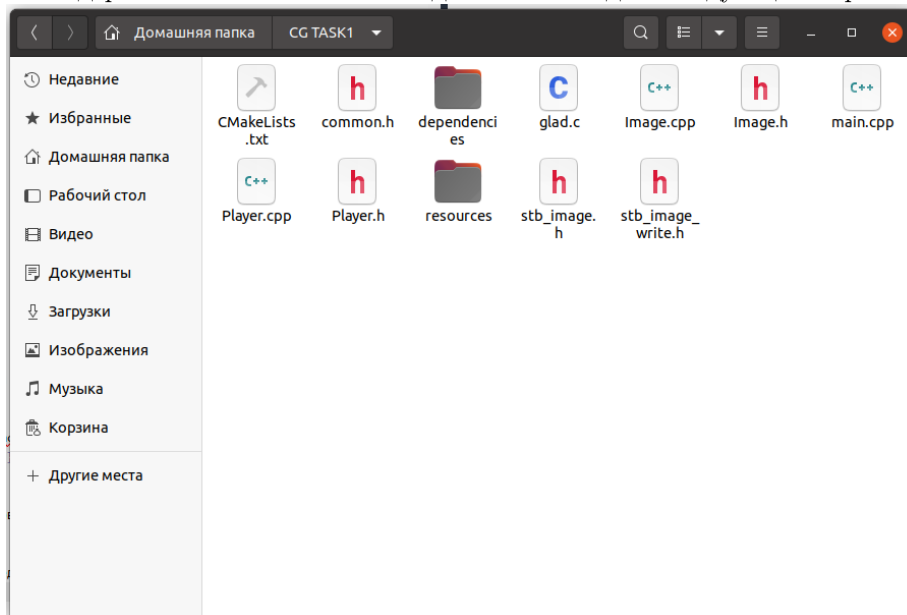
В данной главе мы рассмотрим как подготовиться к написанию проекта, а так же пройдем подготовительный этап для написания кода.

## 1.1 Скачать и установить шаблон

Прежде всего необходимо скачать архив "msu\_cmc\_cg\_2021-master.zip" по следующей ссылке: [ссылка на архив](#).

Содержимое папки "template1\_cpp" разархивировать в удобное для вас место (в нашем случае это папка "CG TASK1").

Содержимое папки CG TASK1 должно выглядеть следующим образом:



## 1.2 Установка необходимых пакетов для работы

Далее необходимо ввести следующие команды в терминал:

```
sudo apt-get update  
sudo apt install build-essential
```

Они установят основные компоненты GCC и базовые пакеты для работы с языками «C» и «C++»

(Подробнее смотрите по ссылке [Установка GCC в Ubuntu](#))

Так же установим пакеты для компиляции make файлов:

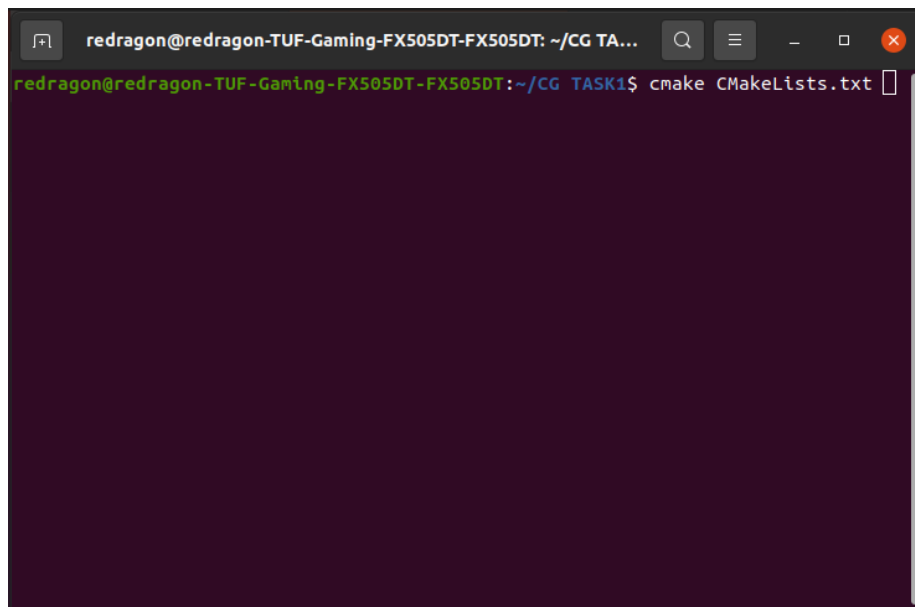
```
sudo apt-get install cmake
sudo apt-get install libglfw3-dev
```

Теперь мы полностью готовы к написанию программы.

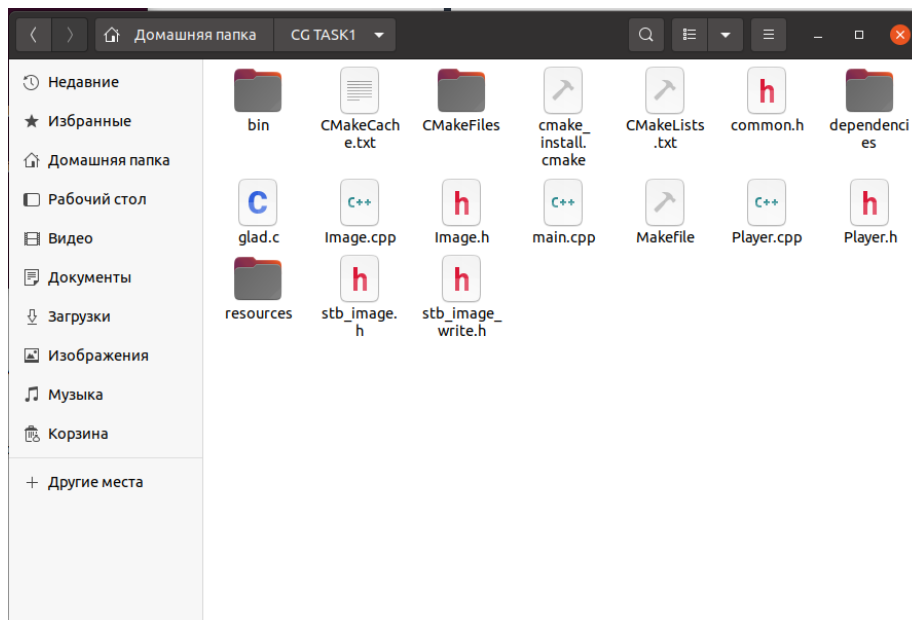
### 1.3 Компиляция

Откройте в терминале корень проекта (в папке CG TASK1) и введите команду

```
cmake CMakeLists.txt
```

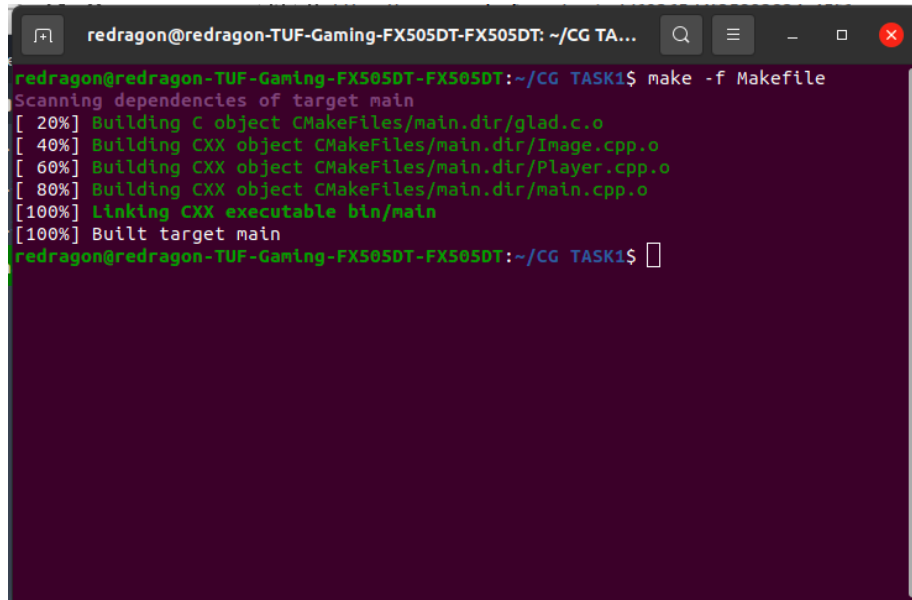


Содержимое папки изменится следующим образом:

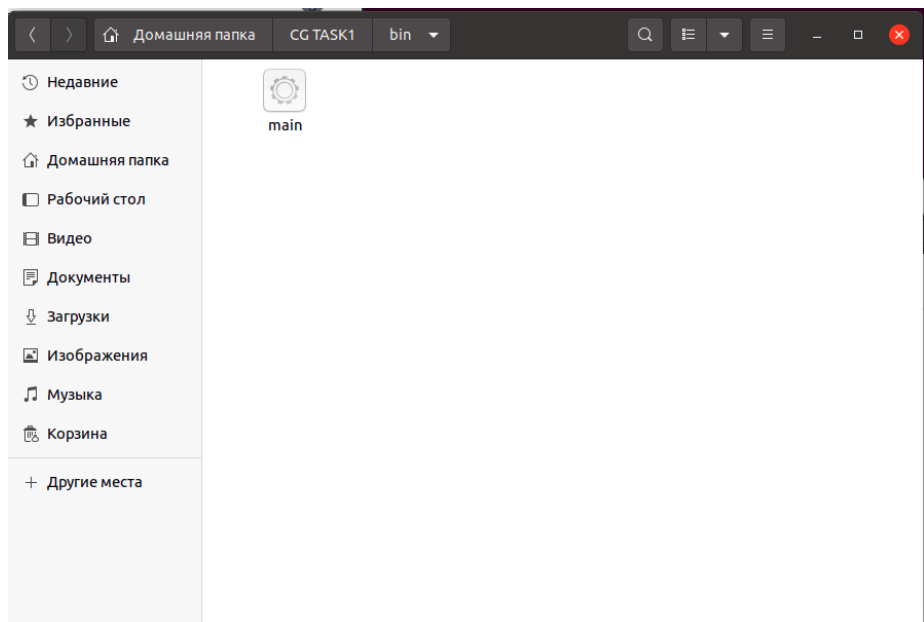


Для компиляции проекта будем использовать следующую команду:

```
make -f Makefile
```

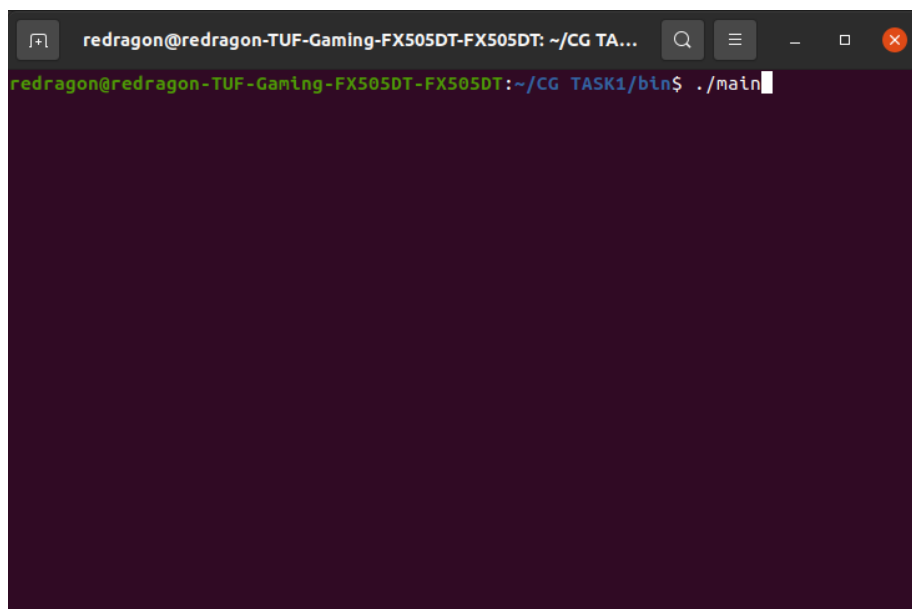


Теперь в папке `bin` появился исполняемый файл `main`



Который через эту папку без труда можно запустить через команду

`./main`

A terminal window with a dark background. The title bar shows the user 'redragon' on a machine named 'redragon-TUF-Gaming-FX505DT-FX505DT' in the directory '~/CG TA...'. The prompt is 'redragon@redragon-TUF-Gaming-FX505DT-FX505DT:~/CG TASK1/bin\$'. The command './main' has been entered and is followed by a cursor. The rest of the terminal is empty.

```
redragon@redragon-TUF-Gaming-FX505DT-FX505DT:~/CG TASK1/bin$ ./main
```

Теперь мы полностью готовы к написанию кода

## 2 Набор необходимых ресурсов

В данной главе указаны все необходимые интернет-ресурсы для работы с библиотекой GLFW и ресурсы для создания игры.

### 2.1 GLFW User Guide

GLFW User Guide - это документ, который поможет лучше освоить работу с библиотекой GLFW. Найти её можно по следующей [ссылке](#).



### 3 Как компилировать и запускать готовое задание

В моем случае задание было создано и добавлено в архив под названием 315\_Aiteyev\_B3

1) разархивируем его в любое удобное место 2) выполним команду

```
make -f Makefile
```

Если не выполняется, то нужно зайти в файл CMakeCache.txt и заменить путь "/home/redragon/CG TASK1" на путь где вы разархивировали папку (везде в файле)

#### 3.1 Как запускать программу

выполним команду

```
./bin/main
```

в корне папки

## 4 Реализованные механики

Прежде всего полностью выполнены все пункты базовой части, перейдем к реализации дополнительных

\* Анимация статических объектов - например, сокровища и шипы ловушек блестят

В нашем случае анимирована подсказка (заметно при запуске)

\* Плавная спрайтовая анимация динамических объектов- походка игрока и врагов, открытие дверей и.т.д. (от 2 до 5 баллов)

В нашем случае анимирован персонаж при передвижении

\* Реализовать графический эффект перехода между уровнями(вариант А) и комнатами (вариант Б) - постепенное “угасание” и появление игровой карты(fade out / fade in), эффект “мозаики”, плавное “перетекание” одного изображения в другое и т.п. (3 балла)

Уровни подгружаются при пересечении специальных областей, без анимаций

\* Эффекты пост-обработки всего изображения - “дрожание” воздуха (heat haze), размытие/туман и т.п. (3 балла)

В моем случае анимированный туман активируется через функцию

```
fogObj.Draw(screenBuffer, fog, deltaTime);
```

внутри кода

\* Реализация и графическое отображение инвентаря (3 балла)

В качестве инвентаря используется правая часть экрана, что выводит здоровье и служебные переменные на экран с помощью следующих команд:

```
str = "Input message here";  
symbol.DrawWord(screenBuffer, deltaTime, 780, 450, str );
```

Данная команда выведет строку str на координаты 780 и 450

\* Графическое отображение характеристик игрока и соответствующие игровые механики - например, если выводится здоровье, то игрок может его

потерять(ловушки, враги) и, возможно, восстановить. (2 балла)  
Здоровье рисуется спрайтом и сопровождается графическим текстом, а так же теряется когда игрок наступает на лаву (пустоту)

## 5 Используемые алгоритмы

Данная глава посвящена алгоритмам программы и реализациям базовых концепций построения игрового движка

ВНИМАНИЕ! Код, написанный в данной главе может отличаться от конечного в приложенном архиве, так как написание отчета шло постепенно с разработкой

### 5.1 Размеры блоков и игрока

Прежде всего обратим внимание на переменную `tileSize` из файла "Image.h"

```
1 || constexpr int tileSize = 64;
```

Это размер нашего игрока (а так же размер единичных игровых блоков, которые образуются из символов текстовой карты)

### 5.2 Границы перемещения игрока

Так же обратим внимание на переменную `move_speed` из структуры `Player` из файла "Player.h"

```
1 || struct Player
2 || {
3 ||     //...
4 || private:
5 ||     //...
6 ||     int move_speed = 1;
7 ||
8 || };
```

Она равна единице, а так же переменная `move_dist` из файла "Player.cpp"

```
1 || void Player::ProcessInput(MovementDir dir)
2 || {
3 ||     int move_dist = move_speed * 1;
4 ||     switch(dir)
5 ||     {
6 ||         case MovementDir::UP:
7 ||
8 ||             old_coords.y = coords.y;
9 ||             coords.y += move_dist;
10 ||
11 ||             if (coords.y >= ymax - tileSize ){ coords.y = old_coords.y
12 ||             ;}
13 ||             break;
14 ||         case MovementDir::DOWN:
15 ||
16 ||             old_coords.y = coords.y;
```

```

17         coords.y -= move_dist;
18
19         if(coords.y < 0){ coords.y = old_coords.y;}
20         break;
21
22     case MovementDir::LEFT:
23
24         old_coords.x = coords.x;
25         coords.x -= move_dist;
26
27         if(coords.x < 0 ){coords.x = old_coords.x;}
28         break;
29
30     case MovementDir::RIGHT:
31
32         old_coords.x = coords.x;
33         coords.x += move_dist;
34
35         if(coords.x >= xmax - tileSize){ coords.x = old_coords.x;}
36         break;
37
38     default:
39         break;
40 }
41 }

```

move\_dist - это количество пикселей пройденное за один шаг. В случае нашей программы мы оставили эту переменную равной единице.

И кроме того, в коде выше видно, как задается ограничение перемещения при скорости равной единице