Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«Санкт-Петербургский политехнический университет Петра Великого»

—

Институт компьютерных наук и кибербезопасности

**Высшая школа кибербезопасности**

**К У Р С О В А Я Р А Б О Т А**

Space Impact

по дисциплине «Языки программирования»

Выполнили

студенты гр. 5131001/30002 Мишенев Н. С.

Квашенникова В. М.

Руководитель

программист Малышев Е. В.

«\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2024 г.

Санкт-Петербург

2024г.

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 3](#_Toc2118922052)

[ОСНОВНАЯ ЧАСТЬ 3](#_Toc838472128)

[Теоретическая часть 4](#_Toc1887085366)

[Реструктуризация проекта 4](#_Toc1149429073)

[Изменения механик игры 5](#_Toc1240111311)

[Оценка сложности неоптимизированного алгоритма 8](#_Toc1685966535)

[Практическая часть 10](#_Toc962372871)

[Алгоритмическая оптимизация 11](#_Toc1859050084)

[Машинно-независимые оптимизации 11](#_Toc144389837)

[Оценка сложности оптимизированного алгоритма 11](#_Toc1988702212)

[Анализ полученных данных для двух версий алгоритмов 11](#_Toc1040727777)

[ЗАКЛЮЧЕНИЕ 12](#_Toc230536217)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 13](#_Toc1504952596)

[ПРИЛОЖЕНИЕ А 14](#_Toc1930717113)

# ВВЕДЕНИЕ

**Цель**

Провести оптимизацию кода игры для улучшения производительности и увеличения FPS в игре.

**Задачи**

1. Реализовать режим игры, представляющий собой некий бенчмарк, для отслеживания производительности игры при большом количестве объектов.
2. Оценить алгоритмическую сложность программы, сделать замеры FPS.
3. Внести алгоритмическую оптимизацию.
4. Внести не менее двух машинно-независимых оптимизаций.
5. Повторно оценить алгоритмическую сложность программы, сделать замеры FPS.
6. Сопоставить полученные данные, сделать выводы.

# ОСНОВНАЯ ЧАСТЬ

## Теоретическая часть

### Реструктуризация проекта

Так как необходимость реализации бенчмарк режима, где будет создаваться очень большое количество объектов, шла вразрез с возможностями программного кода игры, то было принято решение переработать и реструктуризировать проект.

Для переработки проекта был произведен рефакторинг всего исходного кода из файла **main.c**. Он был разбит на две директории и десяток файлов, для упрощения внесения изменений.

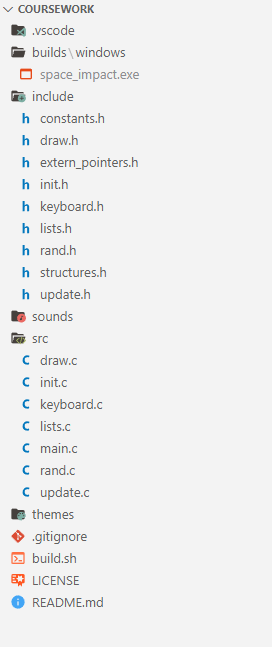


Рис. 1. Структура каталога проекта после внесения изменений.

Также, была сильно изменена система содержания объектов в игре. Вместо задания нескольких объектов указателями и обращения к каждому из них, аллоцируется память для объектов, а затем, они записываются в односвязный линейный список. Таким образом решена проблема ограниченности количества объектов.

### Изменения механик игры

В ходе переработки системы объектов игры, были внесены некоторые правки в основные механики.

* **Количество выпускаемых пуль Игроком и Боссом стали неограничены**

Теперь, одновременно на экране может быть скольугодно много пуль и игрока и босса.



Рис. 2. Битва с боссом.

Босс атакует не без остановки, у него есть определённый интервал аттак, которого у игрока нет, он может атаковать тогда когда захочет.

* **С каждым новым уровнем повышается количество астероидов каждого типа**

Теперь, при переходе на новый уровень, не только увеличивается скорость летящих в игрока камней, но и увеличивается их общее количество.

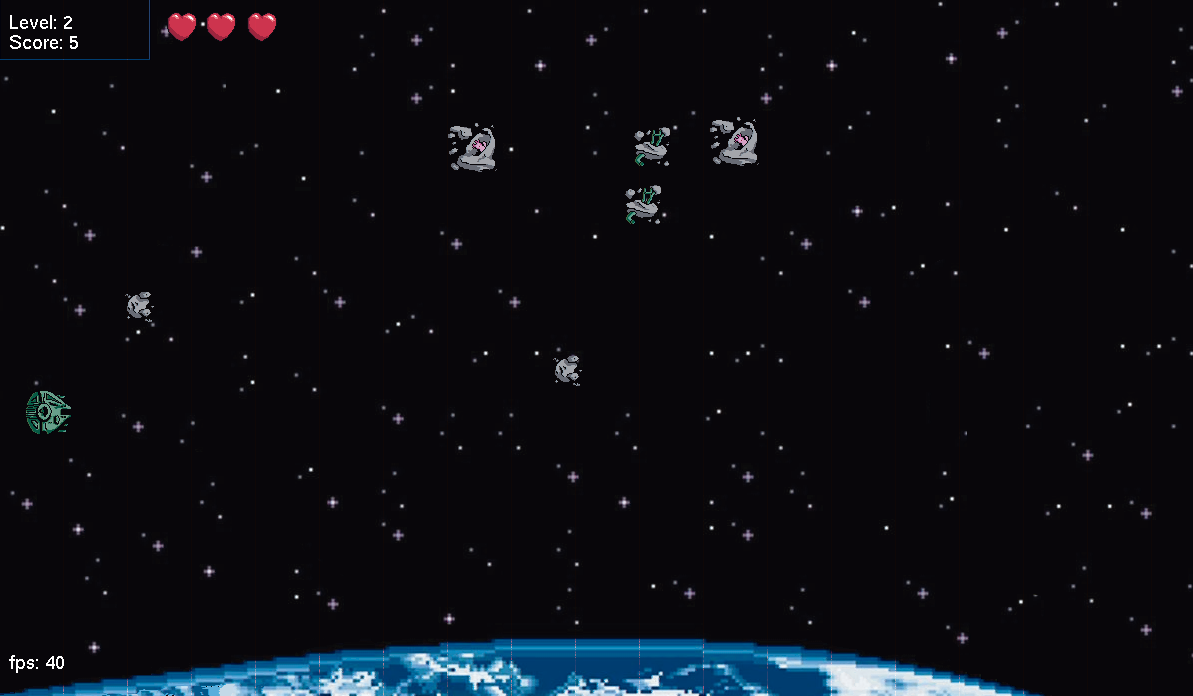


Рис. 3. Демонстрация второго уровня.

Так, увеличивается сложность прохождения уровня за счёт появления дополниельных астероидов, по 1 каждого типа на новый уровень.

В итоге, на I уровне игрок противостоит 3-м астероидам, на II уровне игроку уже придется держаться против 6-ти астероидов, и на III уровне, против 9-ти.

* **Изменены требования по набору очков для перехода на новый уровень**

В связи с увеличением количества выпускаемых пуль игроком, а также увеличением количества “источников очков”, было увеличено количество очков, требуемое для перехода на новй уровень.



Рис. 4. Демонстрация перехода на третий уровень.

Теперь для перехода на 2й уровень нужно все также 5 очков, для перехода на 3й уровень нужно 15 очков, а для перехода на уровень с боссом, 35 очков.

* **Изменено количество очков здоровья у босса**

Так как было увеличено количество выпускаемых пуль, то и количество очков здоровья босса необходимо увеличить, так как теперь по нему проще попасть за счёт упрощенной механики стрельбы.



Рис. 5. Демонстрация уровня с боссом.

Теперь босс имеет 10 очков здоровья вместо прежних трёх. Это позволяет уравновесить шансы игрока на победу и поражение, оставив сложность прохождения игры на прежнем уровне.

* **Добавлен режим “бенчмарк”**

Для того чтобы проверять производительность игры был реализолизован режим “бенчмарк”. Данный режим активируется при нажатии на кнопку “a”

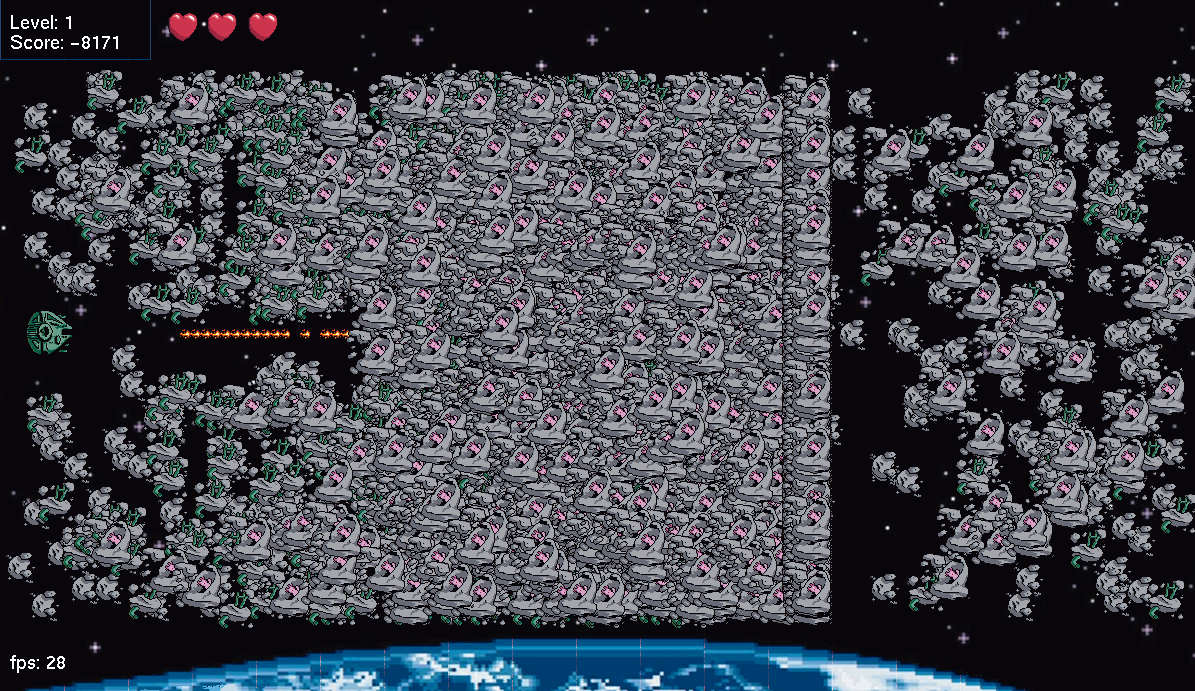


Рис. 6. Демонстрация режима “бенчмарк”.

В этом режиме игрок наделяется бессмертием, у него не отнимаются очки здоровья за столкновения с астероидами, и одновременно запускается заранее заданное количество астероидов, которое достаточно велико.

Главной метрикой производительности игры является счётчик FPS (Frames Per Second), который выведен в левом нижнем углу. Отслеживая его изменения мы и будем судить о произодительности игры.

### Оценка сложности неоптимизированного алгоритма

Самое “сложное” место в коде программы это проверка коллизии объектов, а именно пули и астероидов. Так как мы не знаем положение астероида в пространстве, то мы проверяем коллизию каждой пули с каждым астероидом на экране каждое обновление сцены.

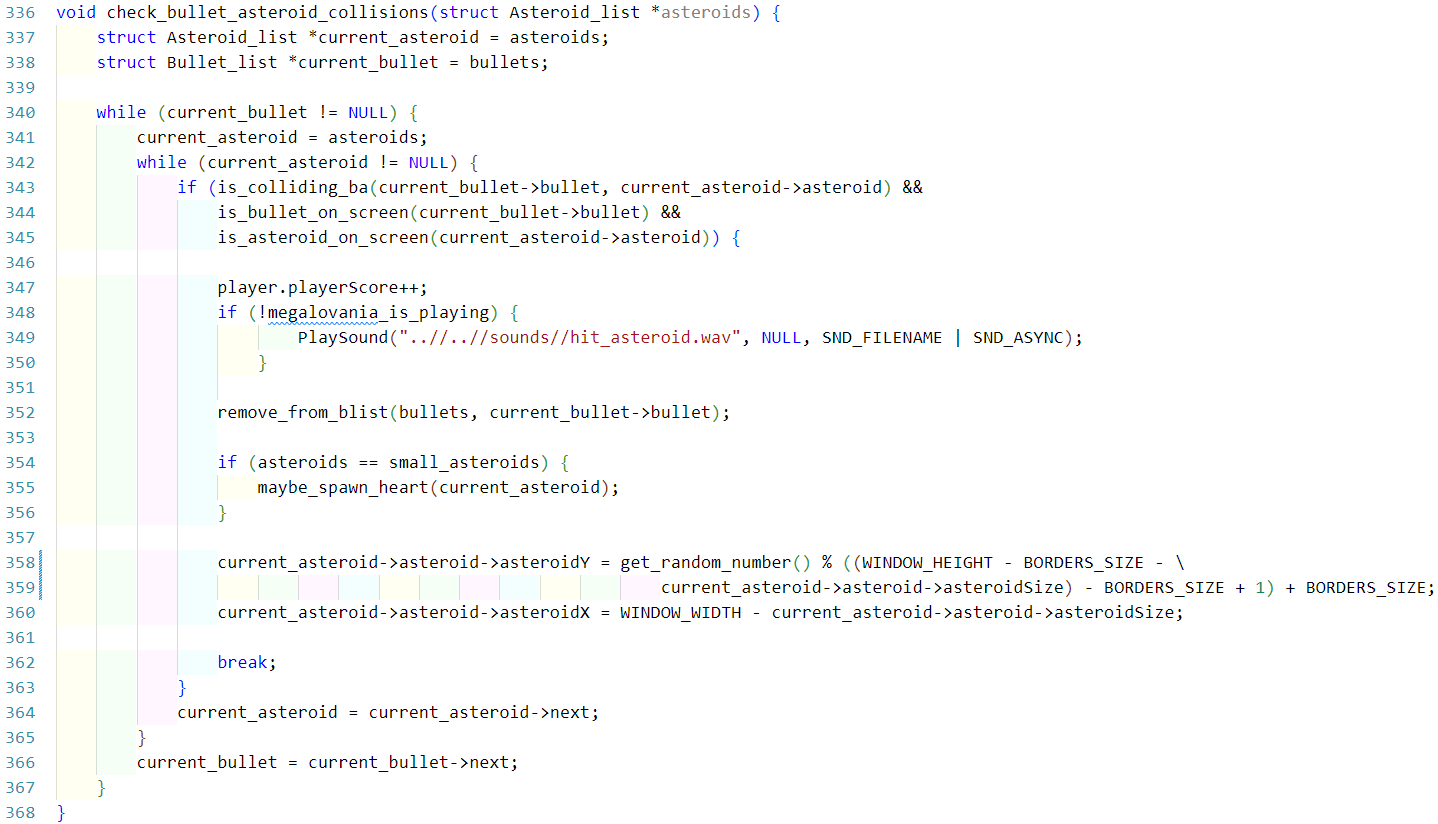


Рис. 7. Функция проверки коллизии пуль с астероидами.

Эта операция довольно ресурсоёмкая, так как внутри каждой проверки осуществляется вызов функции сравнения координат, и весь этот перебор проиходит внутри вложенного цикла. Сложность данного алгоритма сравнения квадратичная - O(n^2).

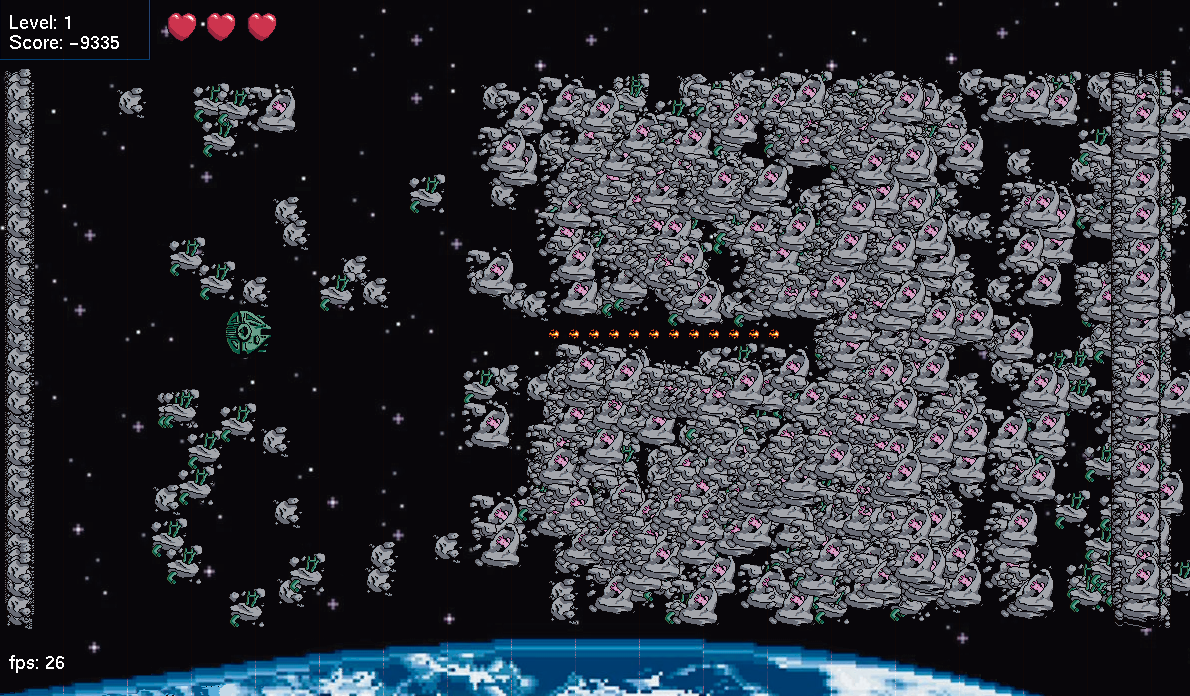


Рис. 8. Оценка FPS для неоптимизированного алгоритма в режиме “бенчмарк” для 50000 астероидов.

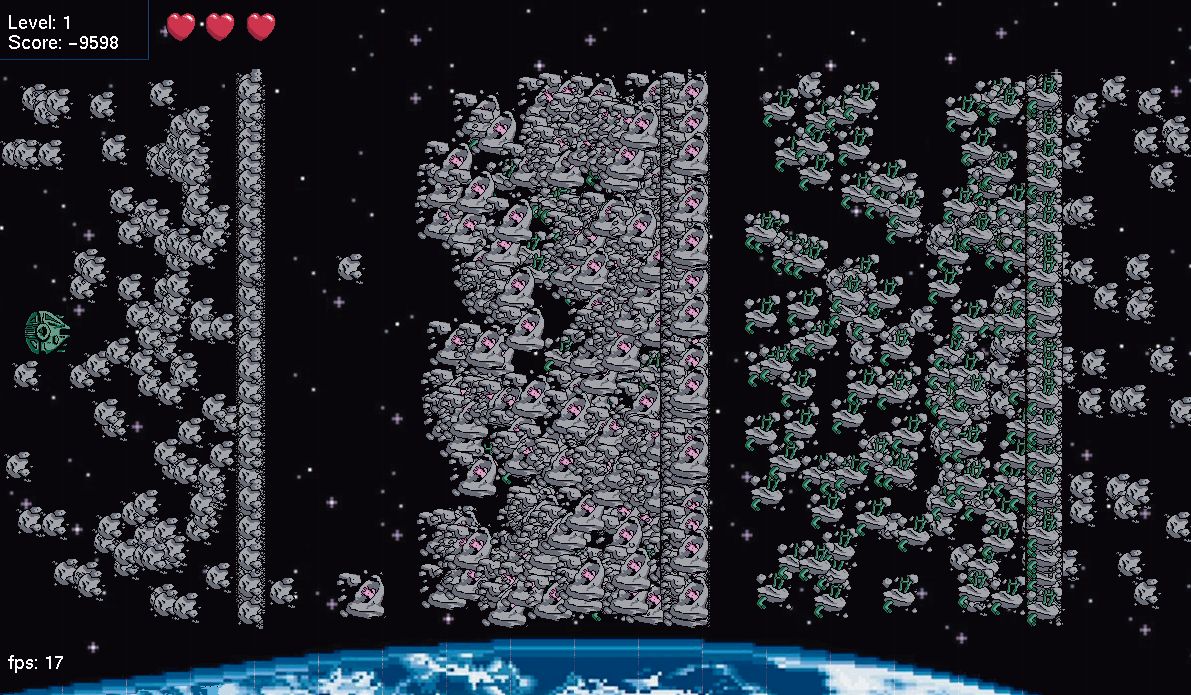


Рис. 9. Оценка FPS для неоптимизированного алгоритма в режиме “бенчмарк” для 100000 астероидов.

В режиме “бенчмарка” для неоптимизированного алгоритма, наблюдается падение FPS до 26 кадров в секунду для 50000 объектов и стабильная работа с 17 кадрами в секунду для 100000 объектов, без повышения.

## Практическая часть

### Алгоритмическая оптимизация

Алгоритмическая оптимизация игрового процесса была направлена на снижение количества проверок коллизий между объектами, такими как пули, астероиды и игрок.

Изначальный алгоритм предполагал сопоставление каждого астероида со всеми пулями на поле или с позицией игрока при каждом обновлении кадра. Это приводило к увеличению количества проверок пропорционально квадрату числа объектов на экране, особенно заметно в интенсивных игровых режимах, таких как "бенчмарк".

Для решения этой проблемы игровое поле было логически разделено на горизонтальные “полосы”. Это разделение позволило ограничить проверки коллизий только областью, в которой объект находится в данный момент. Например:

* Пули проверяются только на столкновение с астероидами в той же полосе.
* Астероиды проверяются на столкновение с игроком только в полосах, где он физически присутствует.

В рамках оптимизации была реализована новая структура данных — массив указателей на ноды линейных списков, организованных для каждой горизонтальной полосы. Каждый элемент этого массива представляет собой указатель на начало линейного списка, в котором хранятся объекты (пули и астероиды), расположенные в соответствующей полосе игрового поля.

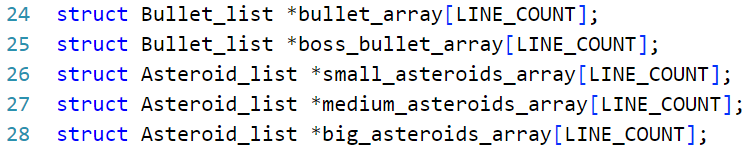


Рис. 10. Новые структуры данных.

А также были внесены некоторые изменения в старые структуры данных:

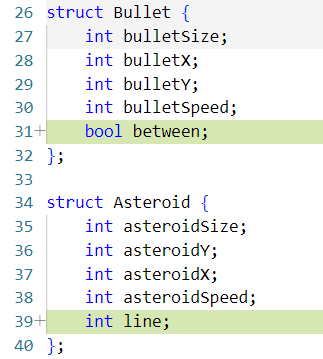


Рис. 11. Изменения в старых структурах данных.

Внедрение новой структуры данных повлекло изменения в функциях добавления объектов. При добавлении новых объектов (пуль или астероидов) теперь учитывается их вертикальная координата, на основе которой вычисляется номер полосы. Этот номер определяет, в какой линейный список будет добавлен объект.

Особое внимание уделено добавлению пуль:

* Если пуля расположена между двумя полосами (например, ее вертикальная координата пересекает границу полос), она добавляется сразу в два соответствующих линейных списка.

Такой подход гарантирует корректное отслеживание коллизий, даже если объект частично находится в соседней полосе.

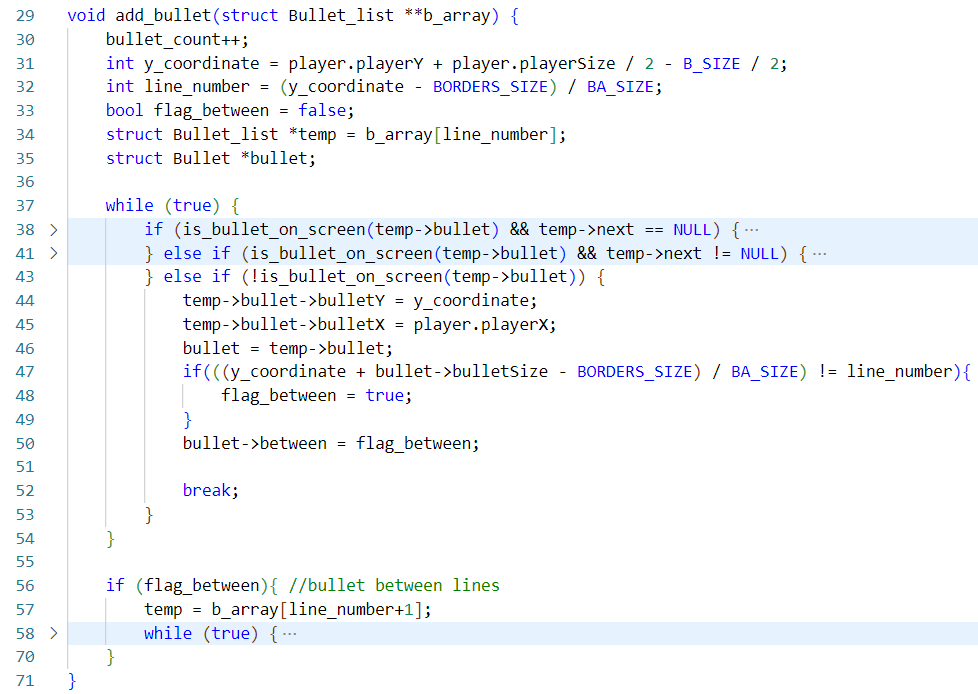


Рис. 12. Разработанная функция добавления пули.

Подобным образом были изменены функции, отвечающие за удаление пуль и астероидов.

Главным преобразованием в работе стало изменение функций проверки коллизий.

* **Функция проверки коллизии пуль с астероидами**

В изначальном варианте проверка каждой пули с каждым астероидом по всему игровому полю выполнялась за квадратичное время. Новый алгоритм значительно улучшил производительность за счёт разделения игрового поля на полосы.

Циклы проверки:

* Внешний цикл проходит по всем полосам (количество полос определяется константой LINE\_COUNT).
* Во вложенных циклах проверяются все пули текущей полосы с астероидами из той же полосы.

Таким образом, несмотря на сохранение квадратичной сложности алгоритма проверки внутри одной полосы из-за вложенных циклов, общее количество переборов значительно сокращается.



Рис. 13. Разработанная функция проверки коллизии пуль с астероидами.

* **Функция проверки коллизии игрока с астероидами**

Функция изначально определяет целевые полосы (или полосу) по вертикальной координате игрока. Это позволяет ограничить проверку коллизий только теми областями, где игрок физически находится, существенно оптимизируя процесс.

Циклы проверки:

* Первый цикл - обрабатывает астероиды, находящиеся в текущей полосе игрока.
* Второй цикл (опциональный) - проверяет астероиды в полосе, расположенной выше, если игрок пересекает границу между полосами.

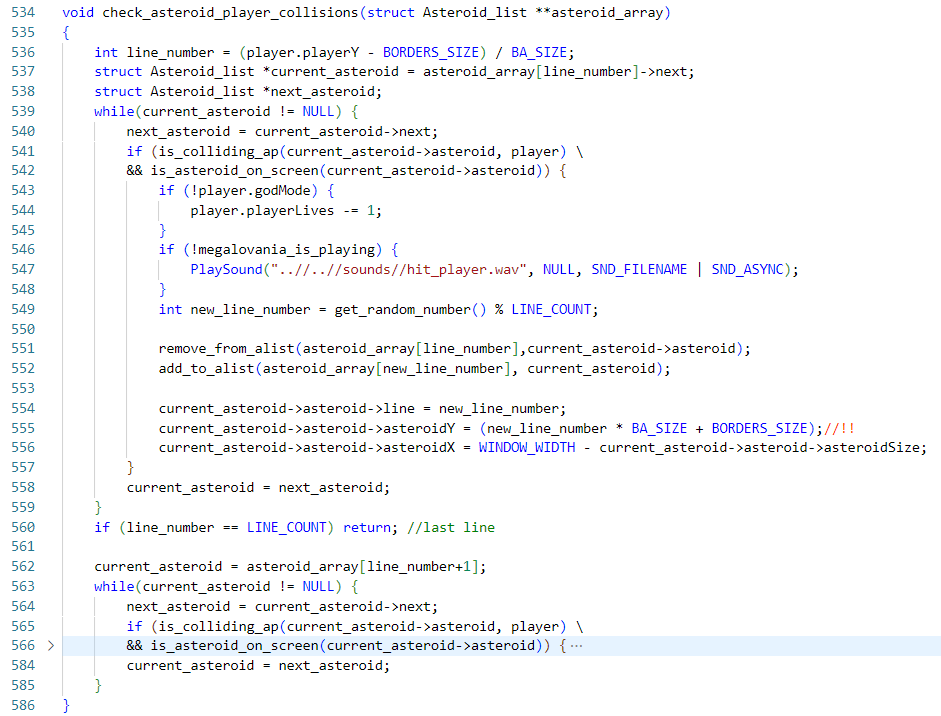


Рис. 14. Разработанная функция проверки коллизии игрока с астероидами.

Таким образом, функции фокусируется только на активных зонах вокруг игрока или пуль, исключая обработку неактуальных областей, что минимизирует количество ненужных вычислений.

### Машинно-независимые оптимизации

Машинно-независимые оптимизации были проведены в нескольких местах и для разных частей кода игры.

* **Оптимизация через предварительную**  **генерацию случайных чисел**

В процессе разработки игры было выявлено, что при создании объектов и задании их координат за одно обновление кадра выполняется до 40 вызовов генерации случайных чисел. Это приводило к значительным накладным расходам при увеличении количества объектов в процессе игры, особенно на более высоких уровнях сложности или в режиме “бенчмарк”.

Для устранения этой проблемы было принято решение заранее создать массив случайных чисел и заполнить его на этапе инициализации игры. Это позволяет уменьшить расходы ресурсов ОС, так как генерация чисел выполняется только один раз при старте игры, а не в каждом кадре. Также ускоряется получение случайных чисел, так как обращение к массиву значительно быстрее, чем вызов функции генерации случайных чисел

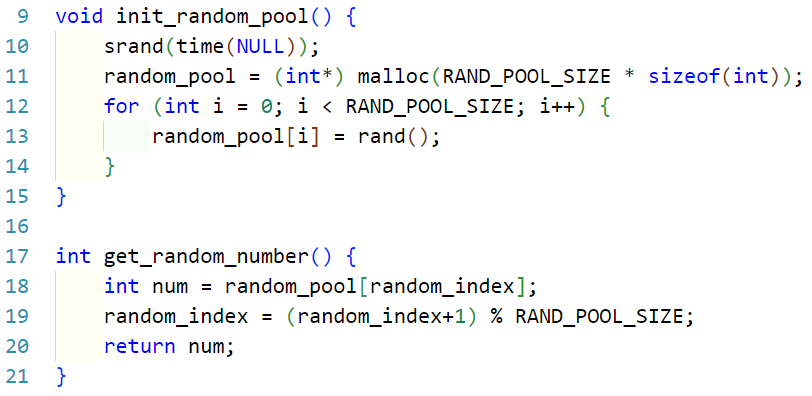


Рис. 15. Генерация случайных чисел.

* **Частота проверки коллизий**

В целях экономии процессорных ресурсов, частота проверки коллизий и обновления состояний у разных объектов была уменьшена. Для этого, в функцию update(...)былодобавлено множество условий, из-за которых частота вызовов функций проверок уменьшается.

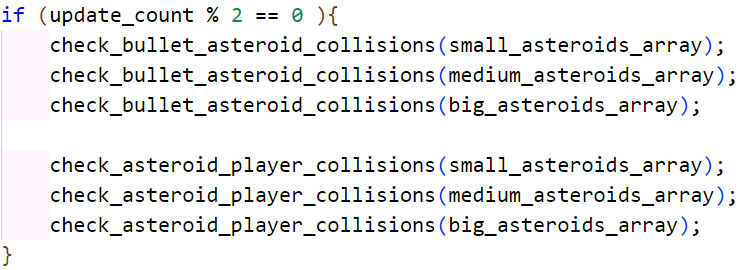


Рис. 16. Уменьшение частоты проверки коллизий.

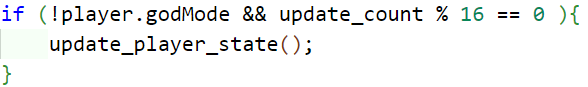


Рис. 17. Уменьшение частоты обновления состояния игрока.

Из-за уменьшения частоты вызовов функций проверок, происходит уменьшение нагрузки на игру, тем самым увеличиваются показания счётчика кадров.

* **Удаление избыточных вызовов**

В изначальной версии игры, вызовы функций glEnable(...) и glDisable(...) для подключения и отключения возможности отрисовки текстур, были расположены крайне неоптимально (при запуске игры и ее окончании) из-за чего ухудшалась производительность игры.

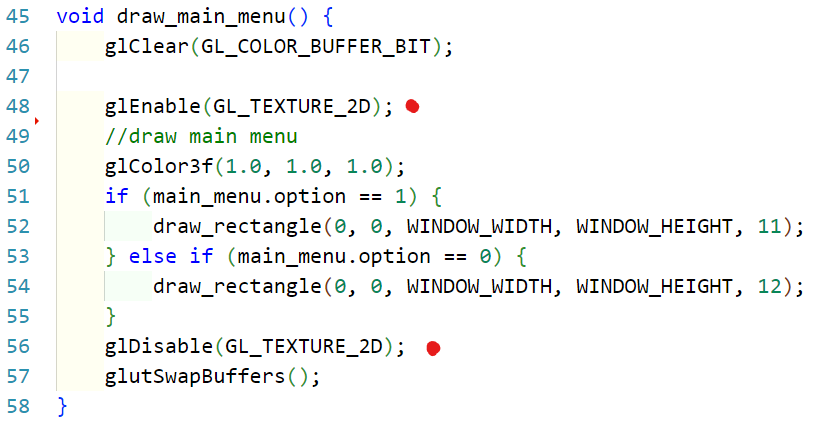


Рис. 18. Добавление вызовов glEnable(...)/glDisable(...).

После проведения оптимизации, было добавлено несколько вызовов пары glEnable(...)/glDisable(...),чтобы отключать двумерное форматирование на то время, когда оно не используется и включать только на время отрисовки объектов.

* **Кеширование текстур**

В изначальной версии игры, каждый раз при отрисовке объекта, в зависимости от его типа подгружалась текстура, которая присуща ему по порядковому номеру положения текстур в памяти.

Было принято решение сохранять порядковый номер последней загруженной текстуры, чтобы при отрисовке нового объекта не загружать ее заново, если она совпадёт с уже установленной.

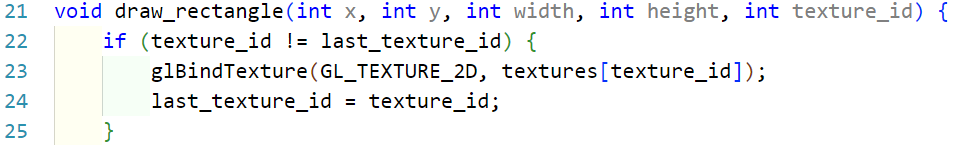


Рис. 19. Кеширование номера текстуры.

Таким образом, при последовательной отрисовке большого количества объектов одного и того же типа, будет уменьшено количество вызовов функции для привязки текстуры и улучшена произодительность.

* **Условие проверки коллизий**

В изначальной версии переделанного алгоритма, функция проверки коллизий пули и другого объекта вызывалась всегда. Это достаточно ресурсоёмкий процесс, поэтому он был ускорен, посредством добавления счётчика существующих пуль.

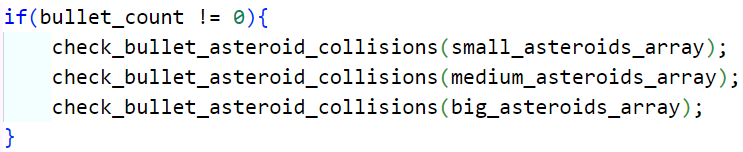


Рис. 20. Проверка количества пуль.

Если данный счётчик положительный, т.е. на экране есть пули которые нужно обработать, тогда запускается функция проверки коллизий, иначе ничего не проиходит.

Таким образом происходит экономия вычислительных ресурсов процессора.

### Оценка сложности оптимизированного алгоритма

После внесения алгоритмической оптимизации, сложность алгоритма так и осталась квадратичной, однако за счёт неполного перебора, количество итераций с функциями проверки коллизий значительно уменьшилось, что и позволило повысить произодительность игры.

### Анализ полученных данных для двух версий алгоритмов

В ходе выполнения работы были получены данные для разного количества создаваемых объектов и сопоставлены им средние значения FPS во время игры для двух версий алгоритмов.

На основе полученных данных была построена соответствующая таблица и график показаний FPS в зависимости от количества созданных астероидов.

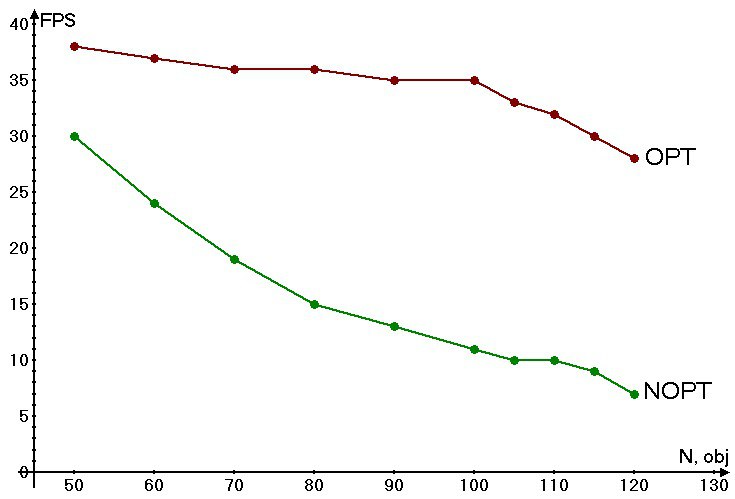


Рис. 21. График зависимости FPS от кол-ва объектов.

Как можно заметить, показатель FPS падает гораздо сильнее и резче для неоптимизированной версии игры. Для оптимизированной версии, изменения количества объектов влияют на FPS более плаво.

# ЗАКЛЮЧЕНИЕ

В ходе выполнения данной работы были выполнены все поставленные задачи, а также достигнута цель работы.

В изначальный алгоритм были внесены алгоритмические и машинно-независимые оптимизации, которые позволили увеличить производительность игры, а также повысить FPS.

Сложность изначального неоптимизированного алгоритма была снижена за счёт алгоритмической оптимизации и, как следствие, была повышена скорость обработки данных.

# ПРИЛОЖЕНИЕ А

Листинг кода - [ссылка на репозиторий в GitHub](https://github.com/clowixdev/space_impact)

* Код файла main.c

#include <GL/glut.h>

#include <stdlib.h>

#include <stdio.h>

#include <stdbool.h>

#include <unistd.h>

#include <windows.h>

#include <mmsystem.h>

#include "constants.h"

#include "structures.h"

#include "lists.h"

#include "keyboard.h"

#include "draw.h"

#include "init.h"

GLuint textures[TEXTURES\_AMT+1] = { 0 };

struct Player player;

struct Boss boss;

struct Heart heart;

struct Menu main\_menu;

struct Bullet\_list \*bullet\_array[LINE\_COUNT];

struct Bullet\_list \*boss\_bullet\_array[LINE\_COUNT];

struct Asteroid\_list \*small\_asteroids\_array[LINE\_COUNT];

struct Asteroid\_list \*medium\_asteroids\_array[LINE\_COUNT];

struct Asteroid\_list \*big\_asteroids\_array[LINE\_COUNT];

struct Bullet\_list \*bullets;

struct Bullet\_list \*boss\_bullets;

struct Asteroid\_list \*small\_asteroids;

struct Asteroid\_list \*medium\_asteroids;

struct Asteroid\_list \*big\_asteroids;

bool changed\_to\_second = false;

bool changed\_to\_third = false;

bool changed\_to\_fourth = false;

bool player\_is\_dead = false;

bool boss\_is\_dead = false;

bool megalovania\_is\_playing = false;

bool spawn\_asteroids = true;

bool game\_just\_started = true;

int boss\_delay = 0;

int update\_count;

int bullet\_count = 0;

int frameCountPerSecond;

int frameCount;

double previousTime;

int \*random\_pool;

int random\_index;

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

glutInit(&argc, argv);

glutInitWindowSize(WINDOW\_WIDTH, WINDOW\_HEIGHT);

glutInitWindowPosition((glutGet(GLUT\_SCREEN\_WIDTH)-WINDOW\_WIDTH)/2,

(glutGet(GLUT\_SCREEN\_HEIGHT)-WINDOW\_HEIGHT)/2);

glutInitDisplayMode(GLUT\_RGBA | GLUT\_DOUBLE);

glutCreateWindow(WINDOW\_CAPTION);

glBlendFunc(GL\_SRC\_ALPHA, GL\_ONE\_MINUS\_SRC\_ALPHA);

glEnable(GL\_BLEND);

for (int i=0; i < LINE\_COUNT; i++){

bullet\_array[i] = init\_bullet\_list\_elem();

boss\_bullet\_array[i] = init\_bullet\_list\_elem();

small\_asteroids\_array[i] = init\_asteroid\_list\_elem('s');

small\_asteroids\_array[i]->asteroid->line = i;

medium\_asteroids\_array[i] = init\_asteroid\_list\_elem('m');

medium\_asteroids\_array[i]->asteroid->line = i;

big\_asteroids\_array[i] = init\_asteroid\_list\_elem('b');

big\_asteroids\_array[i]->asteroid->line = i;

}

main\_menu\_init();

glutDisplayFunc(draw\_main\_menu);

glutKeyboardFunc(handle\_menu\_keyboard);

glutSpecialFunc(handle\_menu\_special\_keyboard);

if (!megalovania\_is\_playing) {

PlaySound("..//..//sounds//menu.wav", NULL, SND\_FILENAME | SND\_ASYNC | SND\_LOOP);

}

glutMainLoop();

return 0;

}

* Код файла rand.c

#include <stdlib.h>

#include <time.h>

#include <GL/glut.h>

#include <stdio.h>

#include "constants.h"

#include "extern\_pointers.h"

void init\_random\_pool() {

srand(time(NULL));

random\_pool = (int\*) malloc(RAND\_POOL\_SIZE \* sizeof(int));

for (int i = 0; i < RAND\_POOL\_SIZE; i++) {

random\_pool[i] = rand();

}

}

int get\_random\_number() {

int num = random\_pool[random\_index];

random\_index = (random\_index + 1) % RAND\_POOL\_SIZE;

return num;

}

* Код файла update.c

#include <GL/glut.h>

#include <windows.h>

#include <mmsystem.h>

#include <stdbool.h>

#include <stdio.h>

#include <time.h>

#include "constants.h"

#include "lists.h"

#include "extern\_pointers.h"

#include "rand.h"

bool is\_bullet\_on\_screen(struct Bullet \*b) {

if (b->bulletX >= 0 && b->bulletX <= WINDOW\_WIDTH) {

return true;

} else {

return false;

}

}

bool is\_asteroid\_on\_screen(struct Asteroid \*a) {

if (a->asteroidX >= 0 && a->asteroidX <= WINDOW\_WIDTH) {

return true;

} else {

return false;

}

}

void add\_bullet(struct Bullet\_list \*\*b\_array) {

bullet\_count++;

int y\_coordinate = player.playerY + player.playerSize / 2 - B\_SIZE / 2;

int line\_number = (y\_coordinate - BORDERS\_SIZE) / BA\_SIZE;

bool flag\_between = false;

struct Bullet\_list \*temp = b\_array[line\_number];

struct Bullet \*bullet;

while (true) {

if (is\_bullet\_on\_screen(temp->bullet) && temp->next == NULL) {

struct Bullet\_list \*bl\_elem = init\_bullet\_list\_elem();

temp->next = bl\_elem;

} else if (is\_bullet\_on\_screen(temp->bullet) && temp->next != NULL) {

temp = temp->next;

} else if (!is\_bullet\_on\_screen(temp->bullet)) {

temp->bullet->bulletY = y\_coordinate;

temp->bullet->bulletX = player.playerX;

bullet = temp->bullet;

if(((y\_coordinate + bullet->bulletSize - BORDERS\_SIZE) / BA\_SIZE) != line\_number) {

flag\_between = true;

}

bullet->between = flag\_between;

break;

}

}

// bullet between lines

if (flag\_between) {

temp = b\_array[line\_number + 1];

while (true) {

if (is\_bullet\_on\_screen(temp->bullet) && temp->next == NULL) {

struct Bullet\_list \*bl\_elem = init\_bullet\_list\_elem();

temp->next = bl\_elem;

} else if (is\_bullet\_on\_screen(temp->bullet) && temp->next != NULL) {

temp = temp->next;

} else if (!is\_bullet\_on\_screen(temp->bullet)) {

temp->bullet = bullet;

break;

}

}

}

}

void add\_boss\_bullet(struct Bullet\_list \*\*b\_array) {

int y\_coordinate = boss.bossY + boss.bossSize / 2 - B\_SIZE / 2;

int line\_number = (y\_coordinate - BORDERS\_SIZE) / BA\_SIZE;

bool flag\_between = false;

if(((y\_coordinate + B\_SIZE - BORDERS\_SIZE) / BA\_SIZE) != line\_number) {

flag\_between = true;

}

struct Bullet\_list \*temp = b\_array[line\_number];

struct Bullet \*bullet;

while (true) {

if (is\_bullet\_on\_screen(temp->bullet) && temp->next == NULL) {

struct Bullet\_list \*bl\_elem = init\_bullet\_list\_elem();

temp->next = bl\_elem;

} else if (is\_bullet\_on\_screen(temp->bullet) && temp->next != NULL) {

temp = temp->next;

} else if (!is\_bullet\_on\_screen(temp->bullet)) {

temp->bullet->bulletY = y\_coordinate;

temp->bullet->bulletX = boss.bossX;

bullet = temp->bullet;

bullet->between = flag\_between;

break;

}

}

// bullet between lines

if (flag\_between) {

temp = b\_array[line\_number + 1];

while (true) {

if (is\_bullet\_on\_screen(temp->bullet) && temp->next == NULL) {

struct Bullet\_list \*bl\_elem = init\_bullet\_list\_elem();

temp->next = bl\_elem;

} else if (is\_bullet\_on\_screen(temp->bullet) && temp->next != NULL) {

temp = temp->next;

} else if (!is\_bullet\_on\_screen(temp->bullet)) {

temp->bullet = bullet;

break;

}

}

}

}

void add\_asteroid(struct Asteroid\_list \*\*a\_array) {

char asteroid\_type = 'o';

int asteroidsize = -10;

int line\_number;

int y\_coordinate;

if (a\_array == small\_asteroids\_array) {

line\_number = get\_random\_number() % LINE\_COUNT;

asteroid\_type = 's';

asteroidsize = SA\_SIZE;

y\_coordinate = (BORDERS\_SIZE + line\_number \* BA\_SIZE) + get\_random\_number() % (BA\_SIZE - asteroidsize);

} else if (a\_array == medium\_asteroids\_array) {

line\_number = get\_random\_number() % LINE\_COUNT;

asteroid\_type = 'm';

asteroidsize = MA\_SIZE;

y\_coordinate = (BORDERS\_SIZE + line\_number \* BA\_SIZE) + get\_random\_number() % (BA\_SIZE - asteroidsize);

} else if (a\_array == big\_asteroids\_array) {

line\_number = get\_random\_number() % LINE\_COUNT;

y\_coordinate = (BORDERS\_SIZE + line\_number \* BA\_SIZE);

asteroid\_type = 'b';

asteroidsize = BA\_SIZE;

}

struct Asteroid\_list \*temp = a\_array[line\_number];

while (temp != NULL) {

if (temp->next == NULL) {

struct Asteroid\_list \*al\_elem = init\_asteroid\_list\_elem(asteroid\_type);

temp->next = al\_elem;

temp->next->asteroid->asteroidY = y\_coordinate;

temp->next->asteroid->asteroidX = WINDOW\_WIDTH - asteroidsize;

temp->next->asteroid->line = line\_number;

return;

}

temp = temp->next;

}

}

bool is\_colliding\_ap(struct Asteroid \*a, struct Player p) {

if (

(a->asteroidY >= p.playerY && a->asteroidY <= p.playerY + p.playerSize \

&& a->asteroidX >= p.playerX && a->asteroidX <= p.playerX + p.playerSize)

||

(a->asteroidY+a->asteroidSize >= p.playerY && a->asteroidY+a->asteroidSize <= p.playerY + p.playerSize \

&& a->asteroidX+a->asteroidSize >= p.playerX && a->asteroidX+a->asteroidSize <= p.playerX + p.playerSize)

)

{

return true;

} else {

return false;

}

}

bool is\_colliding\_hp(struct Heart h, struct Player p) {

if (

(h.heartY >= p.playerY && h.heartY <= p.playerY + p.playerSize \

&& h.heartX >= p.playerX && h.heartX <= p.playerX + p.playerSize)

||

(h.heartY+h.heartSize >= p.playerY && h.heartY+h.heartSize <= p.playerY + p.playerSize \

&& h.heartX+h.heartSize >= p.playerX && h.heartX+h.heartSize <= p.playerX + p.playerSize)

)

{

return true;

} else {

return false;

}

}

bool is\_colliding\_ba(struct Bullet \*b, struct Asteroid \*a) {

if (

(b->bulletY >= a->asteroidY && b->bulletY <= a->asteroidY + a->asteroidSize \

&& b->bulletX >= a->asteroidX && b->bulletX <= a->asteroidX + a->asteroidSize)

||

(b->bulletY+b->bulletSize >= a->asteroidY && b->bulletY+b->bulletSize <= a->asteroidY + a->asteroidSize \

&& b->bulletX+b->bulletSize >= a->asteroidX && b->bulletX+b->bulletSize <= a->asteroidX + a->asteroidSize)

)

{

return true;

} else {

return false;

}

}

bool is\_colliding\_bp(struct Bullet \*b, struct Player p) {

if (

(b->bulletY >= p.playerY && b->bulletY <= p.playerY + p.playerSize \

&& b->bulletX >= p.playerX && b->bulletX <= p.playerX + p.playerSize)

||

(b->bulletY+b->bulletSize >= p.playerY && b->bulletY+b->bulletSize <= p.playerY + p.playerSize \

&& b->bulletX+b->bulletSize >= p.playerX && b->bulletX+b->bulletSize <= p.playerX + p.playerSize)

)

{

return true;

} else {

return false;

}

}

bool is\_colliding\_bbs(struct Bullet \*b, struct Boss bs) {

if (

(b->bulletY >= bs.bossY && b->bulletY <= bs.bossY + bs.bossSize \

&& b->bulletX >= bs.bossX && b->bulletX <= bs.bossX + bs.bossSize)

||

(b->bulletY+b->bulletSize >= bs.bossY && b->bulletY+b->bulletSize <= bs.bossY + bs.bossSize \

&& b->bulletX+b->bulletSize >= bs.bossX && b->bulletX+b->bulletSize <= bs.bossX + bs.bossSize)

)

{

return true;

} else {

return false;

}

}

void update\_boss\_bullet\_position(struct Bullet \*bullet) {

if (is\_bullet\_on\_screen(bullet)) {

bullet->bulletX -= bullet->bulletSpeed;

if (!is\_bullet\_on\_screen(bullet)) {

int line\_number = (bullet->bulletY - BORDERS\_SIZE) / BA\_SIZE;

remove\_from\_blist(boss\_bullet\_array[line\_number], bullet);

if (bullet->between) {

remove\_from\_blist(boss\_bullet\_array[line\_number + 1], bullet);

}

}

}

}

void update\_bullet\_position(struct Bullet \*bullet) {

if (is\_bullet\_on\_screen(bullet)) {

if (bullet->between) {

bullet->bulletX += bullet->bulletSpeed / 2;

} else {

bullet->bulletX += bullet->bulletSpeed;

}

if (!is\_bullet\_on\_screen(bullet)) {

int line\_number = (bullet->bulletY - BORDERS\_SIZE) / BA\_SIZE;

remove\_from\_blist(bullet\_array[line\_number], bullet);

bullet\_count--;

}

}

}

void update\_asteroid\_position(struct Asteroid\_list \*\*asteroid\_array, struct Asteroid \*asteroid) {

if (is\_asteroid\_on\_screen(asteroid)) {

asteroid->asteroidX -= asteroid->asteroidSpeed + (3 \* (player.currentLevel - 1));

} else if (!is\_asteroid\_on\_screen(asteroid) && player.playerLives > 0) {

if (megalovania\_is\_playing) {

asteroid->asteroidX = WINDOW\_WIDTH - asteroid->asteroidSize;

return;

}

int rand = get\_random\_number();

int new\_line\_number = rand % LINE\_COUNT;

int size\_offset = BA\_SIZE - asteroid->asteroidSize;

int size = asteroid->asteroidSize;

struct Asteroid\_list \* list\_elem;

list\_elem = remove\_from\_alist(asteroid\_array[asteroid->line], asteroid);

add\_to\_alist(asteroid\_array[new\_line\_number], list\_elem);

asteroid->asteroidY = (new\_line\_number \* BA\_SIZE + BORDERS\_SIZE);

if (asteroid->asteroidSize < BA\_SIZE) {

asteroid->asteroidY += rand % size\_offset;

}

asteroid->line = new\_line\_number;

asteroid->asteroidX = WINDOW\_WIDTH - asteroid->asteroidSize;

}

}

void update\_player\_state() {

if (game\_just\_started == true) { //!! move to another place

add\_asteroid(small\_asteroids\_array);

add\_asteroid(medium\_asteroids\_array);

add\_asteroid(big\_asteroids\_array);

game\_just\_started = false;

}

if (player.playerLives == 0 && player\_is\_dead == false) {

if (!megalovania\_is\_playing) {

PlaySound("..//..//sounds//lose.wav", NULL, SND\_FILENAME | SND\_ASYNC);

}

player\_is\_dead = true;

spawn\_asteroids = false;

}

if (boss.bossLives == 0 && boss\_is\_dead == false) {

if (!megalovania\_is\_playing) {

PlaySound("..//..//sounds//win.wav", NULL, SND\_FILENAME | SND\_ASYNC);

}

boss\_is\_dead = true;

spawn\_asteroids = false;

}

if (player.playerScore == FOR\_SECOND\_LEVEL && changed\_to\_second == false) {

if (!megalovania\_is\_playing) {

PlaySound("..//..//sounds//next\_level.wav", NULL, SND\_FILENAME | SND\_ASYNC);

}

add\_asteroid(small\_asteroids\_array);

add\_asteroid(medium\_asteroids\_array);

add\_asteroid(big\_asteroids\_array);

player.currentLevel = 2;

changed\_to\_second = true;

}

if (player.playerScore == FOR\_THIRD\_LEVEL && changed\_to\_third == false) {

if (!megalovania\_is\_playing) {

PlaySound("..//..//sounds//next\_level.wav", NULL, SND\_FILENAME | SND\_ASYNC);

}

add\_asteroid(small\_asteroids\_array);

add\_asteroid(medium\_asteroids\_array);

add\_asteroid(big\_asteroids\_array);

player.currentLevel = 3;

changed\_to\_third = true;

}

if (player.playerScore == FOR\_BOSS\_LEVEL && changed\_to\_fourth == false) {

if (!megalovania\_is\_playing) {

PlaySound("..//..//sounds//next\_level.wav", NULL, SND\_FILENAME | SND\_ASYNC);

}

player.currentLevel = 4;

spawn\_asteroids = false;

changed\_to\_fourth = true;

}

}

void update\_boss\_state() {

if (player.currentLevel == 4 && boss\_is\_dead == false) {

//boss movement

if (boss.bossY >= boss.bossSize && boss.reached\_top == true) {

boss.bossY -= 3;

if (boss.bossY < boss.bossSize) {

boss.bossY = boss.bossSize;

boss.reached\_bot = true;

boss.reached\_top = false;

}

}

if (boss.bossY <= WINDOW\_HEIGHT - boss.bossSize - 40 && boss.reached\_bot == true) {

boss.bossY += 3;

if (boss.bossY > WINDOW\_HEIGHT - boss.bossSize - 40) {

boss.bossY = WINDOW\_HEIGHT - boss.bossSize - 40;

boss.reached\_top = true;

boss.reached\_bot = false;

}

}

//boss shooting

if (boss\_delay < BOSS\_DELAY\_VALUE) {

boss\_delay++;

} else if (boss\_delay == BOSS\_DELAY\_VALUE) {

add\_boss\_bullet(boss\_bullet\_array);

boss\_delay = 0;

}

//update boss\_bullet position

for (int i = 0; i < LINE\_COUNT; i++) {

for\_bullet\_list(boss\_bullets, update\_boss\_bullet\_position);

}

//boss\_bullet - player collision

int line\_number = (player.playerY - BORDERS\_SIZE) / BA\_SIZE;

struct Bullet\_list \*check\_bsp = boss\_bullet\_array[line\_number];

while (check\_bsp != NULL) {

if (is\_colliding\_bp(check\_bsp->bullet, player)) {

if (!player.godMode) {

player.playerLives -= 1;

}

if (!megalovania\_is\_playing) {

PlaySound("..//..//sounds//hit\_player.wav", NULL, SND\_FILENAME | SND\_ASYNC);

}

remove\_from\_blist(boss\_bullet\_array[line\_number], check\_bsp->bullet);

if ((line\_number + 1) <= LINE\_COUNT) {

remove\_from\_blist(boss\_bullet\_array[line\_number + 1], check\_bsp->bullet);

}

break;

}

check\_bsp = check\_bsp->next;

}

// between lines

if ((line\_number \* BA\_SIZE + BORDERS\_SIZE + player.playerSize) / BA\_SIZE != line\_number) {

struct Bullet\_list \*check\_bsp = boss\_bullet\_array[line\_number + 1];

while (check\_bsp != NULL) {

if (is\_colliding\_bp(check\_bsp->bullet, player)) {

if (!player.godMode) {

player.playerLives -= 1;

}

if (!megalovania\_is\_playing) {

PlaySound("..//..//sounds//hit\_player.wav", NULL, SND\_FILENAME | SND\_ASYNC);

}

remove\_from\_blist(boss\_bullet\_array[line\_number], check\_bsp->bullet);

if ((line\_number + 1) <= LINE\_COUNT) {

remove\_from\_blist(boss\_bullet\_array[line\_number + 1], check\_bsp->bullet);

}

break;

}

check\_bsp = check\_bsp->next;

}

}

//bullet - boss collision

line\_number = (boss.bossY - BORDERS\_SIZE) / BA\_SIZE;

struct Bullet\_list \*check\_bbs = bullet\_array[line\_number];

while (check\_bbs != NULL) {

if (is\_colliding\_bbs(check\_bbs->bullet, boss)) {

boss.bossLives -= 1;

if (!megalovania\_is\_playing) {

PlaySound("..//..//sounds//hit\_boss.wav", NULL, SND\_FILENAME | SND\_ASYNC);

}

remove\_from\_blist(bullet\_array[line\_number], check\_bbs->bullet);

if ((line\_number + 1) <= LINE\_COUNT) {

remove\_from\_blist(bullet\_array[line\_number + 1], check\_bsp->bullet);

}

break;

}

check\_bbs = check\_bbs->next;

}

line\_number++;

check\_bbs = bullet\_array[line\_number];

while (check\_bbs != NULL) {

if (is\_colliding\_bbs(check\_bbs->bullet, boss)) {

boss.bossLives -= 1;

if (!megalovania\_is\_playing) {

PlaySound("..//..//sounds//hit\_boss.wav", NULL, SND\_FILENAME | SND\_ASYNC);

}

remove\_from\_blist(bullet\_array[line\_number], check\_bbs->bullet);

if ((line\_number + 1) <= LINE\_COUNT) {

remove\_from\_blist(bullet\_array[line\_number + 1], check\_bsp->bullet);

}

break;

}

check\_bbs = check\_bbs->next;

}

if ((line\_number+1) <= LINE\_COUNT && \

((boss.bossY + boss.bossSize - BORDERS\_SIZE) / BA\_SIZE - \

(boss.bossY - BORDERS\_SIZE) / BA\_SIZE ) == 2) {

line\_number++;

check\_bbs = bullet\_array[line\_number];

while (check\_bbs != NULL) {

if (is\_colliding\_bbs(check\_bbs->bullet, boss)) {

boss.bossLives -= 1;

if (!megalovania\_is\_playing) {

PlaySound("..//..//sounds//hit\_boss.wav", NULL, SND\_FILENAME | SND\_ASYNC);

}

remove\_from\_blist(bullet\_array[line\_number], check\_bbs->bullet);

if ((line\_number + 1) <= LINE\_COUNT) {

remove\_from\_blist(bullet\_array[line\_number + 1], check\_bsp->bullet);

}

break;

}

check\_bbs = check\_bbs->next;

}

}

}

}

void maybe\_spawn\_heart(struct Asteroid\_list \*a) {

int rand\_int = get\_random\_number() % 100;

if (rand\_int > 70 && player.playerLives < 3 && heart.heartX < 0) {

if (!megalovania\_is\_playing) {

PlaySound("..//..//sounds//heart\_spawn.wav", NULL, SND\_FILENAME | SND\_ASYNC);

}

heart.heartX = a->asteroid->asteroidX;

heart.heartY = a->asteroid->asteroidY;

}

}

void check\_bullet\_asteroid\_collisions(struct Asteroid\_list \*\*asteroids) {

for (int i = 0; i < LINE\_COUNT; i++) {

struct Asteroid\_list \*current\_asteroid = asteroids[i]->next;

struct Bullet\_list \*current\_bullet = bullet\_array[i];

while (current\_bullet != NULL) {

while (current\_asteroid != NULL) {

if (is\_colliding\_ba(current\_bullet->bullet, current\_asteroid->asteroid) &&

is\_bullet\_on\_screen(current\_bullet->bullet) &&

is\_asteroid\_on\_screen(current\_asteroid->asteroid)) {

player.playerScore++;

if (!megalovania\_is\_playing) {

PlaySound("..//..//sounds//hit\_asteroid.wav", NULL, SND\_FILENAME | SND\_ASYNC);

if (asteroids == small\_asteroids\_array) {

maybe\_spawn\_heart(current\_asteroid);

}

}

remove\_from\_blist(bullet\_array[i], current\_bullet->bullet);

if ((i + 1) < LINE\_COUNT && current\_bullet->bullet->between) {

remove\_from\_blist(bullet\_array[i + 1], current\_bullet->bullet);

}

bullet\_count--;

int new\_line\_number = get\_random\_number() % LINE\_COUNT;

remove\_from\_alist(asteroids[i], current\_asteroid->asteroid);

add\_to\_alist(asteroids[new\_line\_number], current\_asteroid);

current\_asteroid->asteroid->line = new\_line\_number;

current\_asteroid->asteroid->asteroidY = (new\_line\_number \* BA\_SIZE + BORDERS\_SIZE);//!!!

current\_asteroid->asteroid->asteroidX = WINDOW\_WIDTH - current\_asteroid->asteroid->asteroidSize;

break;

}

current\_asteroid = current\_asteroid->next;

}

current\_bullet = current\_bullet->next;

current\_asteroid = asteroids[i]->next;

}

}

}

void check\_asteroid\_player\_collisions(struct Asteroid\_list \*\*asteroid\_array) {

int line\_number = (player.playerY - BORDERS\_SIZE) / BA\_SIZE;

struct Asteroid\_list \*current\_asteroid = asteroid\_array[line\_number]->next;

struct Asteroid\_list \*next\_asteroid;

while (current\_asteroid != NULL) {

next\_asteroid = current\_asteroid->next;

if (is\_colliding\_ap(current\_asteroid->asteroid, player) \

&& is\_asteroid\_on\_screen(current\_asteroid->asteroid)) {

if (!player.godMode) {

player.playerLives -= 1;

}

if (!megalovania\_is\_playing) {

PlaySound("..//..//sounds//hit\_player.wav", NULL, SND\_FILENAME | SND\_ASYNC);

}

int new\_line\_number = get\_random\_number() % LINE\_COUNT;

remove\_from\_alist(asteroid\_array[line\_number], current\_asteroid->asteroid);

add\_to\_alist(asteroid\_array[new\_line\_number], current\_asteroid);

current\_asteroid->asteroid->line = new\_line\_number;

current\_asteroid->asteroid->asteroidY = (new\_line\_number \* BA\_SIZE + BORDERS\_SIZE);//!!

current\_asteroid->asteroid->asteroidX = WINDOW\_WIDTH - current\_asteroid->asteroid->asteroidSize;

}

current\_asteroid = next\_asteroid;

}

//last line

if (line\_number == LINE\_COUNT) {

return;

}

current\_asteroid = asteroid\_array[line\_number + 1];

while (current\_asteroid != NULL) {

next\_asteroid = current\_asteroid->next;

if (is\_colliding\_ap(current\_asteroid->asteroid, player) \

&& is\_asteroid\_on\_screen(current\_asteroid->asteroid)) {

if (!player.godMode) {

player.playerLives -= 1;

}

if (!megalovania\_is\_playing) {

PlaySound("..//..//sounds//hit\_player.wav", NULL, SND\_FILENAME | SND\_ASYNC);

}

int new\_line\_number = get\_random\_number() % LINE\_COUNT;

remove\_from\_alist(asteroid\_array[line\_number + 1],current\_asteroid->asteroid);

add\_to\_alist(asteroid\_array[new\_line\_number], current\_asteroid);

current\_asteroid->asteroid->line = new\_line\_number;

current\_asteroid->asteroid->asteroidY = (new\_line\_number \* BA\_SIZE + BORDERS\_SIZE);//!!

current\_asteroid->asteroid->asteroidX = WINDOW\_WIDTH - current\_asteroid->asteroid->asteroidSize;

}

current\_asteroid = next\_asteroid;

}

}

void update(int aux) {

if (megalovania\_is\_playing) {

add\_bullet(bullet\_array);

}

if (!player.godMode && update\_count % 16 == 0) {

update\_player\_state();

}

if (!player.godMode && player.currentLevel == 4) {

//TODO

//update\_boss\_state();

}

//update player bullet position

for(int i = 0; i < LINE\_COUNT; i++) {

for\_bullet\_list(bullet\_array[i], update\_bullet\_position);

}

if (!player.godMode) {

//update heart position

if (heart.heartX >= 0) {

heart.heartX -= heart.heartSpeed;

if (heart.heartX <= 0) {

heart.heartY = -heart.heartSize;

heart.heartX = -heart.heartSize;

}

}

//check heart-player collision

if (update\_count % 4 == 0 && is\_colliding\_hp(heart, player)) {

player.playerLives += 1;

if (!megalovania\_is\_playing) {

PlaySound("..//..//sounds//heal.wav", NULL, SND\_FILENAME | SND\_ASYNC);

}

heart.heartY = -heart.heartSize;

heart.heartX = -heart.heartSize;

}

}

// collisions check

if (spawn\_asteroids == true && player.currentLevel != 4) {

for(int i = 0; i < LINE\_COUNT; i++) {

for\_asteroid\_list(small\_asteroids\_array, small\_asteroids\_array[i], update\_asteroid\_position);

for\_asteroid\_list(medium\_asteroids\_array, medium\_asteroids\_array[i], update\_asteroid\_position);

for\_asteroid\_list(big\_asteroids\_array, big\_asteroids\_array[i], update\_asteroid\_position);

}

if (update\_count % 2 == 0 ){

if(bullet\_count != 0){

check\_bullet\_asteroid\_collisions(small\_asteroids\_array);

check\_bullet\_asteroid\_collisions(medium\_asteroids\_array);

check\_bullet\_asteroid\_collisions(big\_asteroids\_array);

}

check\_asteroid\_player\_collisions(small\_asteroids\_array);

check\_asteroid\_player\_collisions(medium\_asteroids\_array);

check\_asteroid\_player\_collisions(big\_asteroids\_array);

}

}

update\_count++;

glutPostRedisplay();

glutTimerFunc(25, update, 0);

}

* Код файла lists.c

#include <stdlib.h>

#include <stdio.h>

#include "structures.h"

#include "constants.h"

struct Asteroid\_list\* init\_asteroid\_list\_elem(char type) {

struct Asteroid\_list \*asteroid\_list\_elem = (struct Asteroid\_list \*)calloc(1 ,sizeof(struct Asteroid\_list));

struct Asteroid \*asteroid = (struct Asteroid \*)calloc(1 ,sizeof(struct Asteroid));

switch (type) {

case 's':

asteroid->asteroidSize = SA\_SIZE;

asteroid->asteroidSpeed = SA\_SPEED;

asteroid->asteroidY = -asteroid->asteroidSize;

asteroid->asteroidX = WINDOW\_WIDTH - asteroid->asteroidSize;

asteroid->line = -1;

break;

case 'm':

asteroid->asteroidSize = MA\_SIZE;

asteroid->asteroidSpeed = MA\_SPEED;

asteroid->asteroidY = -asteroid->asteroidSize;

asteroid->asteroidX = WINDOW\_WIDTH - asteroid->asteroidSize;

asteroid->line = -1;

break;

case 'b':

asteroid->asteroidSize = BA\_SIZE;

asteroid->asteroidSpeed = BA\_SPEED;

asteroid->asteroidY = -asteroid->asteroidSize;

asteroid->asteroidX = WINDOW\_WIDTH - asteroid->asteroidSize;

asteroid->line = -1;

break;

default:

break;

}

asteroid\_list\_elem->asteroid = asteroid;

asteroid\_list\_elem->next = NULL;

return asteroid\_list\_elem;

}

struct Bullet\_list \* init\_bullet\_list\_elem() {

struct Bullet\_list \*bullet\_list\_elem = (struct Bullet\_list \*)calloc(1 ,sizeof(struct Bullet\_list));

struct Bullet \*bullet = (struct Bullet \*)calloc(1 ,sizeof(struct Bullet));

bullet->bulletSize = B\_SIZE;

bullet->bulletSpeed = B\_SPEED;

bullet->bulletY = -bullet->bulletSize;

bullet->bulletX = -bullet->bulletSize;

bullet\_list\_elem->bullet = bullet;

bullet\_list\_elem->next = NULL;

return bullet\_list\_elem;

}

void add\_to\_alist(struct Asteroid\_list \*list\_head, struct Asteroid\_list \*a){

a->next = list\_head->next;

list\_head->next = a;

}

void add\_to\_blist(struct Bullet\_list \*list\_head, struct Bullet\_list \*b){

b->next = list\_head->next;

list\_head->next = b;

}

struct Asteroid\_list \* remove\_from\_alist(struct Asteroid\_list \*list\_head, struct Asteroid \*a) {

if (list\_head->asteroid == a) {

list\_head->asteroid->asteroidX = -a->asteroidSize;

list\_head->asteroid->asteroidY = -a->asteroidSize;

return list\_head;

}

struct Asteroid\_list \*temp = list\_head->next;

struct Asteroid\_list \*prev = list\_head;

while(temp != NULL)

{

if (temp->asteroid == a){

prev->next = temp->next;

return temp;

}

prev = temp;

temp = temp->next;

}

return NULL;

}

struct Bullet\_list \* remove\_from\_blist(struct Bullet\_list \*list\_head, struct Bullet \*b) {

if (list\_head->bullet == b) {

list\_head->bullet->bulletX = -b->bulletSize;

list\_head->bullet->bulletY = -b->bulletSize;

return list\_head;

}

struct Bullet\_list \*temp = list\_head->next;

struct Bullet\_list \*prev = list\_head;

while(temp != NULL)

{

if (temp->bullet == b){

prev->next = temp->next;

return temp;

}

prev = temp;

temp = temp->next;

}

}

void for\_asteroid\_list(struct Asteroid\_list \*\*array, struct Asteroid\_list \*list\_head, void (\*func)(struct Asteroid\_list \*\*list\_head, struct Asteroid \*asteroid)) {

struct Asteroid\_list \*temp = list\_head->next;

while(temp != NULL)

{

func(array, temp->asteroid);

temp = temp->next;

}

}

void for\_bullet\_list(struct Bullet\_list \*list\_head, void (\*func)(struct Bullet\*)) {

struct Bullet\_list \*temp = list\_head;

while(temp != NULL)

{

func(temp->bullet);

temp = temp->next;

}

}

* Код файла init.c

#include <GL/glut.h>

#include <stdio.h>

#define STB\_IMAGE\_IMPLEMENTATION

#include <stb\_image.h>

#include "constants.h"

#include "lists.h"

#include "extern\_pointers.h"

#include "rand.h"

void main\_menu\_init() {

glClearColor(1.0, 1.0, 1.0, 1.0);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0, WINDOW\_WIDTH, 0, WINDOW\_HEIGHT);

//?player -> bullet -> boss\_bullet -> s\_asteroid -> m\_asteroid -> b\_asteroid -> boss -> background -> main\_menu\_start -> main\_menu\_exit

glGenTextures(TEXTURES\_AMT, textures);

//!main\_menu\_start - start

glBindTexture(GL\_TEXTURE\_2D, textures[11]);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_S, GL\_REPEAT);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_T, GL\_REPEAT);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);

int menus\_width = 0, menus\_height = 0, menus\_channels = 0;

unsigned char \*menus\_texture\_img = stbi\_load(MENUS\_FILENAME, &menus\_width, &menus\_height, &menus\_channels, 0);

if(menus\_texture\_img == NULL) {

printf("error in loading boss\_texture\_img\n");

// exit(1);

}

printf("%s - %dx%d with %d channels\n", MENUS\_FILENAME, menus\_width, menus\_height, menus\_channels);

if (menus\_channels == 3) {

glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGB, menus\_width, menus\_height, 0, GL\_RGB, GL\_UNSIGNED\_BYTE, menus\_texture\_img);

} else if (menus\_channels == 4) {

glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGBA, menus\_width, menus\_height, 0, GL\_RGBA, GL\_UNSIGNED\_BYTE, menus\_texture\_img);

}

stbi\_image\_free(menus\_texture\_img);

glBindTexture(GL\_TEXTURE\_2D, 0);

//!main\_menu\_start - end

//!main\_menu\_exit - start

glBindTexture(GL\_TEXTURE\_2D, textures[12]);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_S, GL\_REPEAT);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_T, GL\_REPEAT);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);

int menue\_width = 0, menue\_height = 0, menue\_channels = 0;

unsigned char \*menue\_texture\_img = stbi\_load(MENUE\_FILENAME, &menue\_width, &menue\_height, &menue\_channels, 0);

if(menue\_texture\_img == NULL) {

printf("error in loading boss\_texture\_img\n");

// exit(1);

}

printf("%s - %dx%d with %d channels\n", MENUE\_FILENAME, menue\_width, menue\_height, menue\_channels);

if (menue\_channels == 3) {

glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGB, menue\_width, menue\_height, 0, GL\_RGB, GL\_UNSIGNED\_BYTE, menue\_texture\_img);

} else if (menue\_channels == 4) {

glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGBA, menue\_width, menue\_height, 0, GL\_RGBA, GL\_UNSIGNED\_BYTE, menue\_texture\_img);

}

stbi\_image\_free(menue\_texture\_img);

glBindTexture(GL\_TEXTURE\_2D, 0);

//!main\_menu\_exit - end

//!objects initialization

main\_menu.option = 1;

}

void init\_game() {

//!player texture - start

glBindTexture(GL\_TEXTURE\_2D, textures[0]);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_S, GL\_REPEAT);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_T, GL\_REPEAT);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);

int pt\_width = 0, pt\_height = 0, pt\_channels = 0;

unsigned char \*player\_texture\_img = stbi\_load(PT\_FILENAME, &pt\_width, &pt\_height, &pt\_channels, 0);

if(player\_texture\_img == NULL) {

printf("error in loading player\_texture\_img\n");

// exit(1);

}

printf("%s - %dx%d with %d channels\n", PT\_FILENAME, pt\_width, pt\_height, pt\_channels);

if (pt\_channels == 3) {

glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGB, pt\_width, pt\_height, 0, GL\_RGB, GL\_UNSIGNED\_BYTE, player\_texture\_img);

} else if (pt\_channels == 4) {

glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGBA, pt\_width, pt\_height, 0, GL\_RGBA, GL\_UNSIGNED\_BYTE, player\_texture\_img);

}

stbi\_image\_free(player\_texture\_img);

glBindTexture(GL\_TEXTURE\_2D, 0);

//!player texture - end

//!bullet texture - start

glBindTexture(GL\_TEXTURE\_2D, textures[1]);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_S, GL\_REPEAT);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_T, GL\_REPEAT);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);

int bt\_width = 0, bt\_height = 0, bt\_channels = 0;

unsigned char \*bullet\_texture\_img = stbi\_load(BT\_FILENAME, &bt\_width, &bt\_height, &bt\_channels, 0);

if(bullet\_texture\_img == NULL) {

printf("error in loading bullet\_texture\_img\n");

// exit(1);

}

printf("%s - %dx%d with %d channels\n", BT\_FILENAME, bt\_width, bt\_height, bt\_channels);

if (bt\_channels == 3) {

glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGB, bt\_width, bt\_height, 0, GL\_RGB, GL\_UNSIGNED\_BYTE, bullet\_texture\_img);

} else if (bt\_channels == 4) {

glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGBA, bt\_width, bt\_height, 0, GL\_RGBA, GL\_UNSIGNED\_BYTE, bullet\_texture\_img);

}

stbi\_image\_free(bullet\_texture\_img);

glBindTexture(GL\_TEXTURE\_2D, 0);

//!bullet texture - end

//!boss\_bullet texture - start

glBindTexture(GL\_TEXTURE\_2D, textures[2]);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_S, GL\_REPEAT);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_T, GL\_REPEAT);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);

int bbt\_width = 0, bbt\_height = 0, bbt\_channels = 0;

unsigned char \*boss\_bullet\_texture\_img = stbi\_load(BBT\_FILENAME, &bbt\_width, &bbt\_height, &bbt\_channels, 0);

if(boss\_bullet\_texture\_img == NULL) {

printf("error in loading boss\_bullet\_texture\_img\n");

// exit(1);

}

printf("%s - %dx%d with %d channels\n", BBT\_FILENAME, bbt\_width, bbt\_height, bbt\_channels);

if (bbt\_channels == 3) {

glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGB, bbt\_width, bbt\_height, 0, GL\_RGB, GL\_UNSIGNED\_BYTE, boss\_bullet\_texture\_img);

} else if (bbt\_channels == 4) {

glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGBA, bbt\_width, bbt\_height, 0, GL\_RGBA, GL\_UNSIGNED\_BYTE, boss\_bullet\_texture\_img);

}

stbi\_image\_free(boss\_bullet\_texture\_img);

glBindTexture(GL\_TEXTURE\_2D, 0);

//!boss\_bullet texture - end

//!small\_asteroid\_texture - start

glBindTexture(GL\_TEXTURE\_2D, textures[3]);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_S, GL\_REPEAT);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_T, GL\_REPEAT);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);

int sat\_width = 0, sat\_height = 0, sat\_channels = 0;

unsigned char \*small\_asteroid\_texture\_img = stbi\_load(SAT\_FILENAME, &sat\_width, &sat\_height, &sat\_channels, 0);

if(small\_asteroid\_texture\_img == NULL) {

printf("error in loading small\_asteroid\_texture\_img\n");

// exit(1);

}

printf("%s - %dx%d with %d channels\n", SAT\_FILENAME, sat\_width, sat\_height, sat\_channels);

if (sat\_channels == 3) {

glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGB, sat\_width, sat\_height, 0, GL\_RGB, GL\_UNSIGNED\_BYTE, small\_asteroid\_texture\_img);

} else if (sat\_channels == 4) {

glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGBA, sat\_width, sat\_height, 0, GL\_RGBA, GL\_UNSIGNED\_BYTE, small\_asteroid\_texture\_img);

}

stbi\_image\_free(small\_asteroid\_texture\_img);

glBindTexture(GL\_TEXTURE\_2D, 0);

//!small\_asteroid\_texture - end

//!medium\_asteroid\_texture - start

glBindTexture(GL\_TEXTURE\_2D, textures[4]);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_S, GL\_REPEAT);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_T, GL\_REPEAT);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);

int mat\_width = 0, mat\_height = 0, mat\_channels = 0;

unsigned char \*medium\_asteroid\_texture\_img = stbi\_load(MAT\_FILENAME, &mat\_width, &mat\_height, &mat\_channels, 0);

if(medium\_asteroid\_texture\_img == NULL) {

printf("error in loading medium\_asteroid\_texture\_img\n");

// exit(1);

}

printf("%s - %dx%d with %d channels\n", MAT\_FILENAME, mat\_width, mat\_height, mat\_channels);

if (mat\_channels == 3) {

glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGB, mat\_width, mat\_height, 0, GL\_RGB, GL\_UNSIGNED\_BYTE, medium\_asteroid\_texture\_img);

} else if (mat\_channels == 4) {

glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGBA, mat\_width, mat\_height, 0, GL\_RGBA, GL\_UNSIGNED\_BYTE, medium\_asteroid\_texture\_img);

}

stbi\_image\_free(medium\_asteroid\_texture\_img);

glBindTexture(GL\_TEXTURE\_2D, 0);

//!medium\_asteroid\_texture - start

//!big\_asteroid\_texture - start

glBindTexture(GL\_TEXTURE\_2D, textures[5]);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_S, GL\_REPEAT);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_T, GL\_REPEAT);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);

int bat\_width = 0, bat\_height = 0, bat\_channels = 0;

unsigned char \*big\_asteroid\_texture\_img = stbi\_load(BAT\_FILENAME, &bat\_width, &bat\_height, &bat\_channels, 0);

if(big\_asteroid\_texture\_img == NULL) {

printf("error in loading big\_asteroid\_texture\_img\n");

// exit(1);

}

printf("%s - %dx%d with %d channels\n", BAT\_FILENAME, bat\_width, bat\_height, bat\_channels);

if (bat\_channels == 3) {

glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGB, bat\_width, bat\_height, 0, GL\_RGB, GL\_UNSIGNED\_BYTE, big\_asteroid\_texture\_img);

} else if (bat\_channels == 4) {

glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGBA, bat\_width, bat\_height, 0, GL\_RGBA, GL\_UNSIGNED\_BYTE, big\_asteroid\_texture\_img);

}

stbi\_image\_free(big\_asteroid\_texture\_img);

glBindTexture(GL\_TEXTURE\_2D, 0);

//!big\_asteroid\_texture - end

//!boss\_texture - start

glBindTexture(GL\_TEXTURE\_2D, textures[6]);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_S, GL\_REPEAT);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_T, GL\_REPEAT);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);

int bosst\_width = 0, bosst\_height = 0, bosst\_channels = 0;

unsigned char \*boss\_texture\_img = stbi\_load(BOSST\_FILENAME, &bosst\_width, &bosst\_height, &bosst\_channels, 0);

if(boss\_texture\_img == NULL) {

printf("error in loading boss\_texture\_img\n");

// exit(1);

}

printf("%s - %dx%d with %d channels\n", BOSST\_FILENAME, bosst\_width, bosst\_height, bosst\_channels);

if (bosst\_channels == 3) {

glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGB, bosst\_width, bosst\_height, 0, GL\_RGB, GL\_UNSIGNED\_BYTE, boss\_texture\_img);

} else if (bosst\_channels == 4) {

glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGBA, bosst\_width, bosst\_height, 0, GL\_RGBA, GL\_UNSIGNED\_BYTE, boss\_texture\_img);

}

stbi\_image\_free(boss\_texture\_img);

glBindTexture(GL\_TEXTURE\_2D, 0);

//!boss\_texture - end

//!backgorund\_1heart - start

glBindTexture(GL\_TEXTURE\_2D, textures[7]);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_S, GL\_REPEAT);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_T, GL\_REPEAT);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);

int bgr1\_width = 0, bgr1\_height = 0, bgr1\_channels = 0;

unsigned char \*bgr1\_texture\_img = stbi\_load(BGR1\_FILENAME, &bgr1\_width, &bgr1\_height, &bgr1\_channels, 0);

if(bgr1\_texture\_img == NULL) {

printf("error in loading bgr1\_texture\_img\n");

// exit(1);

}

printf("%s - %dx%d with %d channels\n", BGR1\_FILENAME, bgr1\_width, bgr1\_height, bgr1\_channels);

if (bgr1\_channels == 3) {

glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGB, bgr1\_width, bgr1\_height, 0, GL\_RGB, GL\_UNSIGNED\_BYTE, bgr1\_texture\_img);

} else if (bgr1\_channels == 4) {

glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGBA, bgr1\_width, bgr1\_height, 0, GL\_RGBA, GL\_UNSIGNED\_BYTE, bgr1\_texture\_img);

}

stbi\_image\_free(bgr1\_texture\_img);

glBindTexture(GL\_TEXTURE\_2D, 0);

//!background\_1heart - end

//!backgorund\_2heart - start

glBindTexture(GL\_TEXTURE\_2D, textures[8]);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_S, GL\_REPEAT);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_T, GL\_REPEAT);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);

int bgr2\_width = 0, bgr2\_height = 0, bgr2\_channels = 0;

unsigned char \*bgr2\_texture\_img = stbi\_load(BGR2\_FILENAME, &bgr2\_width, &bgr2\_height, &bgr2\_channels, 0);

if(bgr2\_texture\_img == NULL) {

printf("error in loading bgr2\_texture\_img\n");

// exit(1);

}

printf("%s - %dx%d with %d channels\n", BGR1\_FILENAME, bgr2\_width, bgr2\_height, bgr2\_channels);

if (bgr2\_channels == 3) {

glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGB, bgr2\_width, bgr2\_height, 0, GL\_RGB, GL\_UNSIGNED\_BYTE, bgr2\_texture\_img);

} else if (bgr2\_channels == 4) {

glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGBA, bgr2\_width, bgr2\_height, 0, GL\_RGBA, GL\_UNSIGNED\_BYTE, bgr2\_texture\_img);

}

stbi\_image\_free(bgr2\_texture\_img);

glBindTexture(GL\_TEXTURE\_2D, 0);

//!background\_2heart - end

//!backgorund\_3heart - start

glBindTexture(GL\_TEXTURE\_2D, textures[9]);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_S, GL\_REPEAT);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_T, GL\_REPEAT);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);

int bgr3\_width = 0, bgr3\_height = 0, bgr3\_channels = 0;

unsigned char \*bgr3\_texture\_img = stbi\_load(BGR3\_FILENAME, &bgr3\_width, &bgr3\_height, &bgr3\_channels, 0);

if(bgr3\_texture\_img == NULL) {

printf("error in loading bgr3\_texture\_img\n");

// exit(1);

}

printf("%s - %dx%d with %d channels\n", BGR3\_FILENAME, bgr3\_width, bgr3\_height, bgr3\_channels);

if (bgr3\_channels == 3) {

glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGB, bgr3\_width, bgr3\_height, 0, GL\_RGB, GL\_UNSIGNED\_BYTE, bgr3\_texture\_img);

} else if (bgr3\_channels == 4) {

glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGBA, bgr3\_width, bgr3\_height, 0, GL\_RGBA, GL\_UNSIGNED\_BYTE, bgr3\_texture\_img);

}

stbi\_image\_free(bgr3\_texture\_img);

glBindTexture(GL\_TEXTURE\_2D, 0);

//!background\_3heart - end

//!backgorund\_0heart - start

glBindTexture(GL\_TEXTURE\_2D, textures[10]);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_S, GL\_REPEAT);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_T, GL\_REPEAT);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);

int bgr0\_width = 0, bgr0\_height = 0, bgr0\_channels = 0;

unsigned char \*bgr0\_texture\_img = stbi\_load(BGR0\_FILENAME, &bgr0\_width, &bgr0\_height, &bgr0\_channels, 0);

if(bgr0\_texture\_img == NULL) {

printf("error in loading bgr0\_texture\_img\n");

// exit(1);

}

printf("%s - %dx%d with %d channels\n", BGR0\_FILENAME, bgr0\_width, bgr0\_height, bgr0\_channels);

if (bgr0\_channels == 3) {

glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGB, bgr0\_width, bgr0\_height, 0, GL\_RGB, GL\_UNSIGNED\_BYTE, bgr0\_texture\_img);

} else if (bgr0\_channels == 4) {

glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGBA, bgr0\_width, bgr0\_height, 0, GL\_RGBA, GL\_UNSIGNED\_BYTE, bgr0\_texture\_img);

}

stbi\_image\_free(bgr0\_texture\_img);

glBindTexture(GL\_TEXTURE\_2D, 0);

//!background\_0heart - end

//!heart\_texture - start

glBindTexture(GL\_TEXTURE\_2D, textures[13]);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_S, GL\_REPEAT);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_T, GL\_REPEAT);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);

int heart\_width = 0, heart\_height = 0, heart\_channels = 0;

unsigned char \*heart\_texture\_img = stbi\_load(HEART\_FILENAME, &heart\_width, &heart\_height, &heart\_channels, 0);

if(heart\_texture\_img == NULL) {

printf("error in loading heart\_texture\_img\n");

// exit(1);

}

printf("%s - %dx%d with %d channels\n", HEART\_FILENAME, heart\_width, heart\_height, heart\_channels);

if (heart\_channels == 3) {

glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGB, heart\_width, heart\_height, 0, GL\_RGB, GL\_UNSIGNED\_BYTE, heart\_texture\_img);

} else if (heart\_channels == 4) {

glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGBA, heart\_width, heart\_height, 0, GL\_RGBA, GL\_UNSIGNED\_BYTE, heart\_texture\_img);

}

stbi\_image\_free(heart\_texture\_img);

glBindTexture(GL\_TEXTURE\_2D, 0);

//!heart\_texture - end

//!objects initialization

player.currentLevel = PLAYER\_START\_LEVEL;

player.playerSize = PLAYER\_SIZE;

player.playerY = WINDOW\_HEIGHT / 2;

player.playerX = player.playerSize;

player.playerScore = PLAYER\_SCORE;

player.playerLives = PLAYER\_LIVES;

player.godMode = false;

boss.bossLives = BOSS\_LIVES;

boss.bossSize = BOSS\_SIZE;

boss.bossY = WINDOW\_HEIGHT / 2;

boss.bossX = WINDOW\_WIDTH - boss.bossSize \* 2;

heart.heartSize = HEART\_SIZE;

heart.spawn = true;

heart.heartX = WINDOW\_WIDTH - heart.heartSize;

heart.heartY = -heart.heartSize;

heart.heartSpeed = HEART\_SPEED;

boss.reached\_top = true;

boss.reached\_bot = false;

init\_random\_pool();

}

* Код файла keyboard.c

#include <GL/glut.h>

#include <unistd.h>

#include <stdio.h>

#include <windows.h>

#include <mmsystem.h>

#include "constants.h"

#include "extern\_pointers.h"

#include "update.h"

#include "draw.h"

#include "init.h"

void handle\_keyboard(unsigned char key, int x, int y) {

switch (key) {

case 32: //SPACE

if (player.playerLives > 0) {

add\_bullet(bullet\_array);

if (!megalovania\_is\_playing) {

PlaySound("..//..//sounds//shot.wav", NULL, SND\_FILENAME | SND\_ASYNC);

}

}

break;

case 27: //ESC

PlaySound("..//..//sounds//exit.wav", NULL, SND\_FILENAME | SND\_ASYNC);

sleep(1);

exit(0);

break;

case 97: //a

megalovania\_is\_playing = true;

player.godMode = true;

player.playerScore = -9999;

PlaySound("..//..//sounds//megalovania.wav", NULL, SND\_FILENAME | SND\_ASYNC);

for (int amt = 0; amt < MAX\_ASTEROIDS\_IN\_BENCH\_MODE/3; amt++) {

add\_asteroid(small\_asteroids\_array);

add\_asteroid(medium\_asteroids\_array);

add\_asteroid(big\_asteroids\_array);

}

break;

case 105: //i

//TODO infinity mode

break;

}

}

void handle\_movement\_keys(int key, int x, int y) {

switch (key) {

case GLUT\_KEY\_DOWN:

player.playerY -= PLAYER\_MOVE\_STEP;

if (player.playerY < BORDERS\_SIZE) {

player.playerY = BORDERS\_SIZE;

}

break;

case GLUT\_KEY\_UP:

player.playerY += PLAYER\_MOVE\_STEP;

if (player.playerY + player.playerSize + BORDERS\_SIZE > WINDOW\_HEIGHT) {

player.playerY = WINDOW\_HEIGHT - player.playerSize - BORDERS\_SIZE;

}

break;

case GLUT\_KEY\_RIGHT:

player.playerX += PLAYER\_MOVE\_STEP;

if (player.playerX > player.playerSize + PLAYER\_MOVEMENTS\_WIDTH) {

player.playerX = player.playerSize + PLAYER\_MOVEMENTS\_WIDTH;

}

break;

case GLUT\_KEY\_LEFT:

player.playerX -= PLAYER\_MOVE\_STEP;

if (player.playerX < player.playerSize) {

player.playerX = player.playerSize;

}

break;

}

}

void handle\_menu\_keyboard(unsigned char key, int x, int y) {

switch (key) {

case 27: //ESC

PlaySound("..//..//sounds//exit.wav", NULL, SND\_FILENAME | SND\_ASYNC);

sleep(1);

exit(0);

break;

case 13: //ENTER

if (main\_menu.option == 1) {

printf("Player started the game, initializing rest of things...\n");

if (!megalovania\_is\_playing) {

PlaySound("..//..//sounds//start.wav", NULL, SND\_FILENAME | SND\_ASYNC);

}

init\_game();

glutDisplayFunc(draw\_scene);

glutKeyboardFunc(handle\_keyboard);

glutSpecialFunc(handle\_movement\_keys);

glutTimerFunc(25, update, 0);

break;

} else if (main\_menu.option == 0) {

PlaySound("..//..//sounds//exit.wav", NULL, SND\_FILENAME | SND\_ASYNC);

sleep(1);

exit(0);

break;

}

}

}

void handle\_menu\_special\_keyboard(int key, int x, int y) {

switch (key) {

case GLUT\_KEY\_DOWN:

// PlaySound("..//..//sounds//select.wav", NULL, SND\_FILENAME | SND\_ASYNC);

main\_menu.option = 0;

break;

case GLUT\_KEY\_UP:

// PlaySound("..//..//sounds//select.wav", NULL, SND\_FILENAME | SND\_ASYNC);

main\_menu.option = 1;

break;

}

glutPostRedisplay();

}

* Код файла draw.c

#include <GL/glut.h>

#include <stdio.h>

#include "constants.h"

#include "extern\_pointers.h"

#include "update.h"

static int last\_texture\_id = -1;

void frame\_counter () {

frameCount++;

double currentTime = (double)glutGet(GLUT\_ELAPSED\_TIME) / 1000.0;

if (currentTime - previousTime >= 1.0) {

frameCountPerSecond = frameCount;

frameCount = 0;

previousTime = currentTime;

}

}

void draw\_rectangle(int x, int y, int width, int height, int texture\_id) {

if (texture\_id != last\_texture\_id) {

glBindTexture(GL\_TEXTURE\_2D, textures[texture\_id]);

last\_texture\_id = texture\_id;

}

glBegin(GL\_QUADS);

glTexCoord2f(0.0f, 1.0f); glVertex2f(x, y);

glTexCoord2f(1.0f, 1.0f); glVertex2f(x + width, y);

glTexCoord2f(1.0f, 0.0f); glVertex2f(x + width, y + height);

glTexCoord2f(0.0f, 0.0f); glVertex2f(x, y + height);

glEnd();

}

void draw\_text(int x, int y, char\* string) {

glColor3f(1.0, 1.0, 1.0);

glRasterPos2f(x, y);

for (int i = 0; string[i] != '\0'; ++i) {

glutBitmapCharacter(GLUT\_BITMAP\_HELVETICA\_18, string[i]);

}

}

void draw\_main\_menu() {

glClear(GL\_COLOR\_BUFFER\_BIT);

glEnable(GL\_TEXTURE\_2D);

//draw main menu

glColor3f(1.0, 1.0, 1.0);

if (main\_menu.option == 1) {

draw\_rectangle(0, 0, WINDOW\_WIDTH, WINDOW\_HEIGHT, 11);

} else if (main\_menu.option == 0) {

draw\_rectangle(0, 0, WINDOW\_WIDTH, WINDOW\_HEIGHT, 12);

}

glDisable(GL\_TEXTURE\_2D);

glutSwapBuffers();

}

void draw\_scene() {

glClear(GL\_COLOR\_BUFFER\_BIT);

glEnable(GL\_TEXTURE\_2D);

//draw background

if (player.playerLives == 3) {

draw\_rectangle(0, 0, WINDOW\_WIDTH, WINDOW\_HEIGHT, 9);

} else if (player.playerLives == 2) {

draw\_rectangle(0, 0, WINDOW\_WIDTH, WINDOW\_HEIGHT, 8);

} else if (player.playerLives == 1) {

draw\_rectangle(0, 0, WINDOW\_WIDTH, WINDOW\_HEIGHT, 7);

}

glDisable(GL\_TEXTURE\_2D);

if (player.playerLives == 0) {

//draw background

glEnable(GL\_TEXTURE\_2D);

draw\_rectangle(0, 0, WINDOW\_WIDTH, WINDOW\_HEIGHT, 10);

glDisable(GL\_TEXTURE\_2D);

char game\_over[DRAW\_TEXT\_LENGTH];

sprintf(game\_over, "Game over!");

draw\_text((WINDOW\_WIDTH / 2) - 50, WINDOW\_HEIGHT / 2, game\_over);

char score[DRAW\_TEXT\_LENGTH];

sprintf(score, "Score: %d", player.playerScore);

draw\_text((WINDOW\_WIDTH / 2) - 45, WINDOW\_HEIGHT / 2 - 40, score);

char level[DRAW\_TEXT\_LENGTH];

sprintf(level, "Level: %d", player.currentLevel);

draw\_text((WINDOW\_WIDTH / 2) - 40, WINDOW\_HEIGHT / 2 - 60, level);

char closing[DRAW\_TEXT\_LENGTH];

sprintf(closing, "To close game press ESC");

draw\_text((WINDOW\_WIDTH / 2) - 120, WINDOW\_HEIGHT / 2 - 80, closing);

} else if (boss.bossLives == 0) {

//draw background

glEnable(GL\_TEXTURE\_2D);

draw\_rectangle(0, 0, WINDOW\_WIDTH, WINDOW\_HEIGHT, 10);

glDisable(GL\_TEXTURE\_2D);

char game\_end[DRAW\_TEXT\_LENGTH];

sprintf(game\_end, "You won!");

draw\_text((WINDOW\_WIDTH / 2) - 50, WINDOW\_HEIGHT / 2, game\_end);

char score[DRAW\_TEXT\_LENGTH];

sprintf(score, "Score: %d", player.playerScore);

draw\_text((WINDOW\_WIDTH / 2) - 45, WINDOW\_HEIGHT / 2 - 40, score);

char level[DRAW\_TEXT\_LENGTH];

sprintf(level, "Level: %d", player.currentLevel);

draw\_text((WINDOW\_WIDTH / 2) - 40, WINDOW\_HEIGHT / 2 - 60, level);

char closing[DRAW\_TEXT\_LENGTH];

sprintf(closing, "To close game press ESC");

draw\_text((WINDOW\_WIDTH / 2) - 120, WINDOW\_HEIGHT / 2 - 80, closing);

} else if (player.currentLevel == 4) {

//draw player

glEnable(GL\_TEXTURE\_2D);

draw\_rectangle(player.playerX - player.playerSize / 2, player.playerY, player.playerSize, player.playerSize, 0);

//draw player\_bullet

for (int i = 0; i < LINE\_COUNT; i++) {

struct Bullet\_list \*bullet\_elem = bullet\_array[i];

while (bullet\_elem != NULL) {

if (is\_bullet\_on\_screen(bullet\_elem->bullet)) {

if (!bullet\_elem->bullet->between || (bullet\_elem->bullet->between && i % 2 == 0)) {

draw\_rectangle(bullet\_elem->bullet->bulletX, \

bullet\_elem->bullet->bulletY, \

bullet\_elem->bullet->bulletSize, \

bullet\_elem->bullet->bulletSize, 1);

}

}

bullet\_elem = bullet\_elem->next;

}

}

//draw boss

draw\_rectangle(boss.bossX, boss.bossY, boss.bossSize, boss.bossSize, 6);

//TODO

//draw boss\_bullet

// for (int i = 0; i < LINE\_COUNT; i++) {

// struct Bullet\_list \*boss\_bullet\_elem = boss\_bullet\_array[i];

// while (boss\_bullet\_elem != NULL) {

// if (is\_bullet\_on\_screen(boss\_bullet\_elem->bullet)) {

// draw\_rectangle(boss\_bullet\_elem->bullet->bulletX, \

// boss\_bullet\_elem->bullet->bulletY, \

// boss\_bullet\_elem->bullet->bulletSize, \

// boss\_bullet\_elem->bullet->bulletSize, 2);

// }

// boss\_bullet\_elem = boss\_bullet\_elem->next;

// }

// }

glDisable(GL\_TEXTURE\_2D);

//draw level

char level[DRAW\_TEXT\_LENGTH];

sprintf(level, "Level: Boss");

draw\_text(10, WINDOW\_HEIGHT - 30, level);

//draw boss\_lives

char boss\_lives[DRAW\_TEXT\_LENGTH];

sprintf(boss\_lives, "Boss lives: %d", boss.bossLives);

draw\_text(10, WINDOW\_HEIGHT - 50, boss\_lives);

} else {

//draw player

glEnable(GL\_TEXTURE\_2D);

draw\_rectangle(player.playerX - player.playerSize / 2, player.playerY, player.playerSize, player.playerSize, 0);

//draw heart

draw\_rectangle(heart.heartX, heart.heartY, heart.heartSize, heart.heartSize, 13);

//draw bullet

for (int i = 0; i < LINE\_COUNT; i++) {

struct Bullet\_list \*bullet\_elem = bullet\_array[i];

while (bullet\_elem != NULL) {

if (is\_bullet\_on\_screen(bullet\_elem->bullet)) {

if (!bullet\_elem->bullet->between) {

draw\_rectangle(bullet\_elem->bullet->bulletX, \

bullet\_elem->bullet->bulletY, \

bullet\_elem->bullet->bulletSize, \

bullet\_elem->bullet->bulletSize, 1);

} else if (i % 2 == 0) {

draw\_rectangle(bullet\_elem->bullet->bulletX, \

bullet\_elem->bullet->bulletY, \

bullet\_elem->bullet->bulletSize, \

bullet\_elem->bullet->bulletSize, 1);

}

}

bullet\_elem = bullet\_elem->next;

}

}

//draw small\_asteroid

for (int i = 0; i < LINE\_COUNT; i++) {

struct Asteroid\_list \*sa\_elem = small\_asteroids\_array[i]->next;

while (sa\_elem != NULL) {

if (is\_asteroid\_on\_screen(sa\_elem->asteroid)) {

draw\_rectangle(sa\_elem->asteroid->asteroidX, \

sa\_elem->asteroid->asteroidY, \

sa\_elem->asteroid->asteroidSize, \

sa\_elem->asteroid->asteroidSize, 3);

}

sa\_elem = sa\_elem->next;

}

}

//draw medium\_asteroid

for (int i = 0; i < LINE\_COUNT; i++) {

struct Asteroid\_list \*ma\_elem = medium\_asteroids\_array[i]->next;

while (ma\_elem != NULL) {

if (is\_asteroid\_on\_screen(ma\_elem->asteroid)) {

draw\_rectangle(ma\_elem->asteroid->asteroidX, \

ma\_elem->asteroid->asteroidY, \

ma\_elem->asteroid->asteroidSize, \

ma\_elem->asteroid->asteroidSize, 4);

}

ma\_elem = ma\_elem->next;

}

}

//draw big\_asteroid

for (int i = 0; i < LINE\_COUNT; i++) {

struct Asteroid\_list \*ba\_elem = big\_asteroids\_array[i]->next;

while (ba\_elem != NULL) {

if (is\_asteroid\_on\_screen(ba\_elem->asteroid)) {

draw\_rectangle(ba\_elem->asteroid->asteroidX, \

ba\_elem->asteroid->asteroidY, \

ba\_elem->asteroid->asteroidSize, \

ba\_elem->asteroid->asteroidSize, 5);

}

ba\_elem = ba\_elem->next;

}

}

glDisable(GL\_TEXTURE\_2D);

//draw level

char level[DRAW\_TEXT\_LENGTH];

sprintf(level, "Level: %d", player.currentLevel);

draw\_text(10, WINDOW\_HEIGHT - 30, level);

//draw score

char score[DRAW\_TEXT\_LENGTH];

sprintf(score, "Score: %d", player.playerScore);

draw\_text(10, WINDOW\_HEIGHT - 50, score);

//draw fps

char fps[DRAW\_TEXT\_LENGTH];

sprintf(fps, "fps: %d", frameCountPerSecond);

draw\_text(10, 30, fps);

}

frame\_counter();

glutSwapBuffers();

}

* Код файла constants.h

#ifndef CONSTANTS\_H

#define CONSTANTS\_H

// Window constants

#define WINDOW\_CAPTION "Space Impact - 5131001/30002"

#define WINDOW\_WIDTH 1200

#define WINDOW\_HEIGHT 700

#define DRAW\_TEXT\_LENGTH 15

// rand.c constants

#define RAND\_POOL\_SIZE 200

// Game canvas constants

#define LINE\_COUNT ((WINDOW\_HEIGHT - 2 \* BORDERS\_SIZE) / BA\_SIZE)

#define BORDERS\_SIZE 75

// Game points constants

#define FOR\_SECOND\_LEVEL 5

#define FOR\_THIRD\_LEVEL 15

#define FOR\_BOSS\_LEVEL 35

#define BOSS\_DELAY\_VALUE 35

// Game objects constants

#define MAX\_ASTEROIDS\_IN\_BENCH\_MODE 100000

#define SA\_SPEED 4

#define SA\_SIZE 30

#define MA\_SPEED 3

#define MA\_SIZE 40

#define BA\_SPEED 2

#define BA\_SIZE 50

#define B\_SIZE 10

#define B\_SPEED 10

#define PLAYER\_SIZE 50

#define PLAYER\_LIVES 3

#define PLAYER\_SCORE 0

#define PLAYER\_START\_LEVEL 1

#define PLAYER\_MOVE\_STEP 10

#define PLAYER\_MOVEMENTS\_WIDTH 200

#define BOSS\_LIVES 10

#define BOSS\_SIZE 70

#define HEART\_SIZE 30

#define HEART\_SPEED 4

// Textures constants

#define TEXTURES\_AMT 13

#define PT\_FILENAME "../../themes/player.bmp"

#define BT\_FILENAME "../../themes/bullet.bmp"

#define BBT\_FILENAME "../../themes/boss\_bullet.bmp"

#define SAT\_FILENAME "../../themes/small\_asteroid.bmp"

#define MAT\_FILENAME "../../themes/medium\_asteroid.bmp"

#define BAT\_FILENAME "../../themes/big\_asteroid.bmp"

#define BOSST\_FILENAME "../../themes/boss.bmp"

#define BGR0\_FILENAME "../../themes/background\_0heart.bmp"

#define BGR1\_FILENAME "../../themes/background\_1heart.bmp"

#define BGR2\_FILENAME "../../themes/background\_2heart.bmp"

#define BGR3\_FILENAME "../../themes/background\_3heart.bmp"

#define MENUS\_FILENAME "../../themes/main\_menu\_start.bmp"

#define MENUE\_FILENAME "../../themes/main\_menu\_exit.bmp"

#define HEART\_FILENAME "../../themes/heart.bmp"

#endif

* Код файла draw.h

#ifndef DRAW\_H

#define DRAW\_H

void frame\_counter(void);

void draw\_rectangle(int, int, int, int, int);

void draw\_text(int, int, char\*);

void draw\_main\_menu(void);

void draw\_scene(void);

#endif

* Код файла extern\_pointers.h

#ifndef EXTERN\_P\_H

#define EXTERN\_P\_H

#include "constants.h"

#include "structures.h"

#include "lists.h"

extern GLuint textures[TEXTURES\_AMT+1];

extern bool changed\_to\_second;

extern bool changed\_to\_third;

extern bool changed\_to\_fourth;

extern bool player\_is\_dead;

extern bool boss\_is\_dead;

extern bool spawn\_asteroids;

extern bool game\_just\_started;

extern bool megalovania\_is\_playing;

extern struct Player player;

extern struct Boss boss;

extern struct Heart heart;

extern struct Menu main\_menu;

extern int boss\_delay;

extern struct Bullet\_list \*bullet\_array[LINE\_COUNT];

extern struct Bullet\_list \*boss\_bullet\_array[LINE\_COUNT];

extern struct Asteroid\_list \*small\_asteroids\_array[LINE\_COUNT];

extern struct Asteroid\_list \*medium\_asteroids\_array[LINE\_COUNT];

extern struct Asteroid\_list \*big\_asteroids\_array[LINE\_COUNT];

extern struct Bullet\_list \*bullets;

extern struct Bullet\_list \*boss\_bullets;

extern struct Asteroid\_list \*small\_asteroids;

extern struct Asteroid\_list \*medium\_asteroids;

extern struct Asteroid\_list \*big\_asteroids;

extern int update\_count;

extern int bullet\_count;

extern int frameCountPerSecond;

extern int frameCount;

extern double previousTime;

extern int \*random\_pool;

extern int random\_index;

#endif

* Код файла init.h

#ifndef INIT\_H

#define INIT\_H

void main\_menu\_init(void);

void init\_game(void);

#endif

* Код файла keyboard.h

#ifndef KEYBOARD\_H

#define KEYBOARD\_H

void handle\_keyboard(unsigned char, int, int);

void handle\_movement\_keys(int, int, int);

void handle\_menu\_keyboard(unsigned char, int, int);

void handle\_menu\_special\_keyboard(int, int, int);

#endif

* Код файла lists.h

#ifndef LISTS\_H

#define LISTS\_H

#include "structures.h"

struct Asteroid\_list \* init\_asteroid\_list\_elem(char type);

struct Bullet\_list \* init\_bullet\_list\_elem();

void remove\_from\_alist(struct Asteroid\_list \*list\_head, struct Asteroid \*a);

void remove\_from\_blist(struct Bullet\_list \*list\_head, struct Bullet \*b);

void for\_asteroid\_list(struct Asteroid\_list \*list\_head, void (\*func)(struct Asteroid\*));

void for\_bullet\_list(struct Bullet\_list \*list\_head, void (\*func)(struct Bullet\*));

#endif

* Код файла rand.h

#ifndef RAND\_H

#define RAND\_H

void init\_random\_pool();

int get\_random\_number();

#endif

* Код файла structures.h

#ifndef STRUCTURES\_H

#define STRUCTURES\_H

#include <stdbool.h>

struct Player {

int currentLevel;

int playerSize;

int playerY;

int playerX;

int playerLives;

int playerScore;

bool godMode;

bool between;

};

struct Boss {

int bossSize;

int bossY;

int bossX;

int bossLives;

bool reached\_top;

bool reached\_bot;

};

struct Bullet {

int bulletSize;

int bulletX;

int bulletY;

int bulletSpeed;

bool between;

};

struct Asteroid {

int asteroidSize;

int asteroidY;

int asteroidX;

int asteroidSpeed;

int line;

};

struct Heart {

bool spawn;

int heartSize;

int heartY;

int heartX;

int heartSpeed;

};

struct Menu {

int option;

};

struct Bullet\_list {

struct Bullet \*bullet;

struct Bullet\_list \*next;

};

struct Asteroid\_list {

struct Asteroid \*asteroid;

struct Asteroid\_list \*next;

};

#endif

* Код файла update.h

#ifndef UPDATE\_H

#define UPDATE\_H

#include <stdbool.h>

#include "structures.h"

bool is\_asteroid\_on\_screen(struct Asteroid \*a);

bool is\_bullet\_on\_screen(struct Bullet \*b);

void add\_asteroid(struct Asteroid\_list \*\*a\_array);

void add\_boss\_bullet(struct Bullet\_list \*\*b\_array);

void add\_bullet(struct Bullet\_list \*\*b\_array);

void maybe\_spawn\_heart(struct Asteroid\_list \*a);

bool is\_colliding\_ap(struct Asteroid \*a, struct Player p);

bool is\_colliding\_hp(struct Heart h, struct Player p);

bool is\_colliding\_ba(struct Bullet \*b, struct Asteroid \*a);

bool is\_colliding\_bp(struct Bullet \*b, struct Player p);

bool is\_colliding\_bbs(struct Bullet \*b, struct Boss bs);

void check\_bullet\_asteroid\_collisions(struct Asteroid\_list \*\*asteroids);

void check\_asteroid\_player\_collisions(struct Asteroid\_list \*\*asteroid\_array);

void update\_asteroid\_position(struct Asteroid\_list \*\*asteroid\_array, struct Asteroid \*asteroid);

void update\_bullet\_position(struct Bullet \*bullet);

void update\_boss\_state(void);

void player\_state\_update(void);

void update(int aux);

#endif