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**

**КУРСОВОЙ ПРОЕКТ**

**Разработка игры Space Impact**

по дисциплине «Структуры данных»

Выполнили

студенты гр. 5131001/30002 Мишенев Н. С.

Квашенникова В. М.

<*подпись*>

Руководитель

программист Вагисаров В. Б.

<*подпись*>

Санкт-Петербург

2024г.

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc1418331109)

[ОСНОВНАЯ ЧАСТЬ 3](#_Toc523577014)

[ТЕОРЕТИЧЕСКАЯ ЧАСТЬ 4](#_Toc1756071619)

[ИСТОРИЯ ИГРЫ SPACE IMPACT 4](#_Toc116343919)

[КЛАССИЧЕСКИЕ ПРАВИЛА ИГРЫ SPACE IMPACT 5](#_Toc969486691)

[OPENGL И FREEGLUT В VSCODE 6](#_Toc425617020)

[ТЕКСТУРЫ И STB\_IMAGE 7](#_Toc26214003)

[ЗВУКИ 10](#_Toc511805342)

[ПРАКТИЧЕСКАЯ ЧАСТЬ 12](#_Toc229396318)

[РЕАЛИЗАЦИЯ МЕХАНИК ДВИЖЕНИЯ И СТРЕЛЬБЫ 14](#_Toc1256778361)

[УСЛОВИЯ СМЕНЫ УРОВНЕЙ 17](#_Toc610827734)

[МЕХАНИКА БОНУСА - СЕРДЦА 20](#_Toc1807005924)

[ЗАКЛЮЧЕНИЕ 24](#_Toc23275291)

[ПРИЛОЖЕНИЕ 26](#_Toc509039194)

ВВЕДЕНИЕ

**Цель**

Разработать прототип игры Space Impact на языке Си с использованием OpenGL.

**Задачи**

1. Изучить литературу необходимую для написания кода.
2. Продумать игровую логику и механики.
3. Реализовать прототип игры.
4. Провести тестирование и исправить выявленные ошибки.
5. Нарисовать текстуры на экране для соответствующих объектов.

ОСНОВНАЯ ЧАСТЬ

## **Теоретическая часть**

### **История игры space impact**

Space Impact — это серия мобильных игр, которая была опубликована Nokia, и ее игры обычно поставлялись в комплекте с несколькими устройствами Nokia.

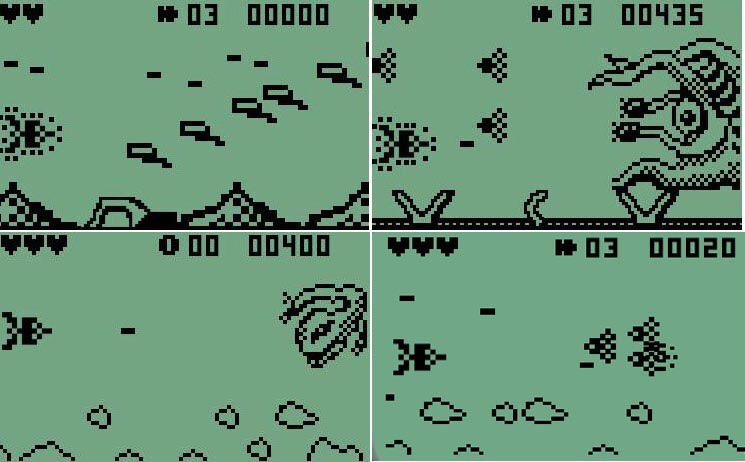
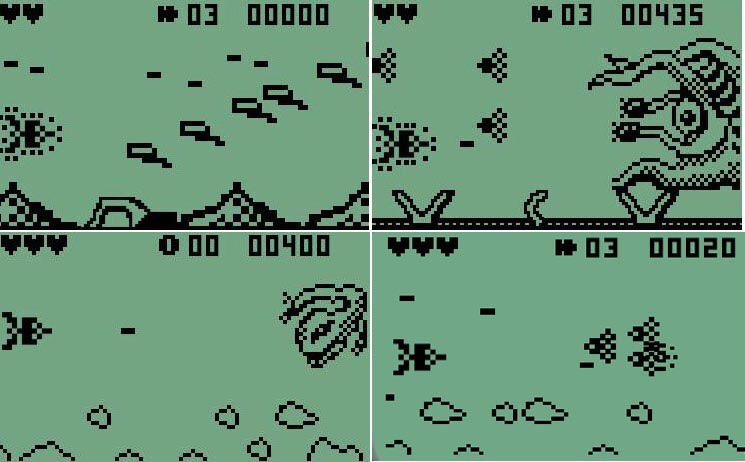
 

Рис. 1, 2. Space Impact на телефонах Nokia

Впервые Space Impact появился на Nokia 3310 в 2000 году, а позже был включен в другие модели, выпущенные в 2001–2004 годах. Расширенные версии Nokia 3310 для ***WAP*** (а именно 3330, 3350 и 3410) давали возможность загружать дополнительные главы Space Impact через WAP соединение телефона.

Эта игра обрела большую популярность среди пользователей и потому развивалась на устройствах Nokia ещё десять лет и восемь версий вплоть до трёхмерной Space Impact: Meteor Shield 2010 года.



Рис. 3. интерфейс Space Impact: Meteor Shield

Эта игра считается одной из культовых игр и именно поэтому, в 2010 году, ***CNET*** (американское СМИ о технологиях) включил ее в топ-10 "величайших мобильных игр всех времен" и сказал, что она раздвигает границы возможного на мобильном устройстве.

### **Классические правила игры space impact**

Говоря о первых версиях игры, правила познаются пользователем довольно интуитивно. У вашего корабля есть 3 жизни. Вы можете стрелять из космического бластера чтобы убивать врагов, которые также будут стараться убивать вас.



Рис. 4. Одна из копий игры Space Impact – Space Attack

В это же время необходимо уворачиваться от летящих в вас метеоритов, чтобы не потерять жизни. После набора необходимого количества очков вы встретите босса, усиленного врага, с которым вам придется сразиться чтобы продолжить игру или же закончить её.

По мере набора очков, темп игры может ускориться, в таком случае, пользователю будет сложнее “оставаться на плаву”

### **Opengl и freeglut в vscode**

Для разработки игры на языке Си был выбран программный интерфейс OpenGL и библиотека, предоставляющая доступ к его возможностям, ***FreeGLUT***. Так как разработка велась в текстовом редакторе ***VSCode***, то предварительно, для успешной компиляции программы, было необходимо настроить все зависимости и правильно подключить рабочие инструменты.

Каталог проекта имеет следующую структуру:

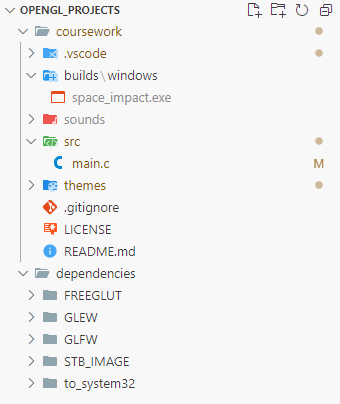


Рис. 5. Структура каталога проекта

Все необходимые инструменты находятся в соседнем каталоге

Всё это производилось посредством изменения аргументов при компиляции проекта. Полная команда для компиляции проекта располагается в файле ***tasks.json*** (Приложение А) в директории ***.vscode*** и вызывается посредством возможностей ***VSCode***.

Внутри вызова этой команды путь до локальной директории проекта заменен на ***${workspaceRoot}.***

### **Текстуры и stb\_image**

В процессе создания проекта, было решено сделать особый акцент на визуальном аспекте продукта. Важно было не только создать функциональную игру, но и сделать ее эстетически привлекательной. Особое внимание уделялось внешнему облику объектов и общей стилистике игры, поэтому использовался для объектов стиль ***pixel art*** с целью сохранения духа оригинальной игры Space Impact. Этот выбор был обусловлен двумя важными факторами:

1. Атмосфера

Стиль ***pixel art*** подходит для создания ретро-атмосферы, которая присуща Space Impact. Он позволяет передать ностальгическую эстетику и усилить ощущение игры из прошлого.

1. Компактность

При работе в таком стиле, даже для невероятно малых объектов можно прорисовать мелкие детали и при этом сохранить их четкость и визуальную привлекательность.

Уникальные текстуры были созданы в растровом графическом редакторе ***Krita*** с использованием цветовой модели ***RGBA***.

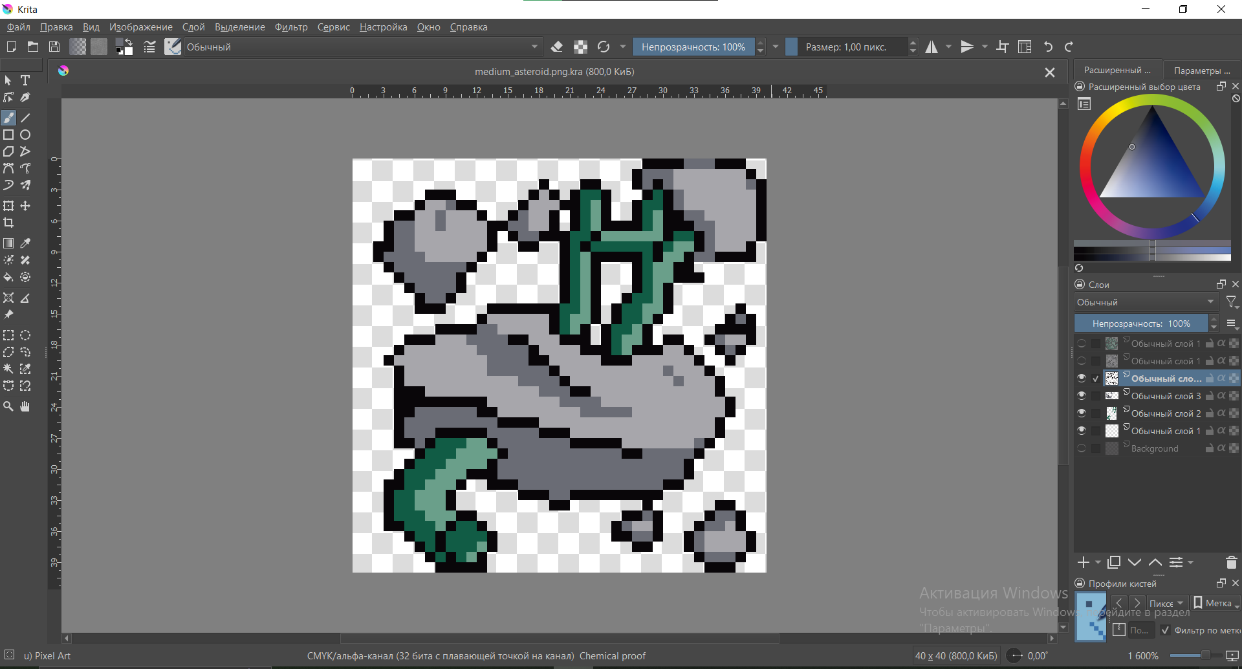


Рис. 6. Растровый графический редактор Krita

В общей сложности было необходимо было отрисовать **14** текстур следующих размеров:

1. Игрок – **50х50** пикс.
2. Пуля игрока – **10х10** пикс.
3. Малый астероид – **30х30** пикс.
4. Средний астероид – **40х40** пикс.
5. Большой астероид – **50х50** пикс.
6. Босс – **50х50** пикс.
7. Пуля босса – **25х25** пикс.
8. Бонус – **30х30** пикс.
9. Фоны с сердцами (4 шт.) – **1200х700** пикс.

10. Меню (Play / Exit) – **1200x700** пикс

Со всеми выполненными текстурами вы можете ознакомиться в приложении Б.

Для дальнейшей работы с текстурами уже в самом проекте следовало сохранить изображения с расширением ***BMP***, для корректного переноса в программу, однако предложенные графическим редактором инструменты не подошли ввиду отсутствия возможности сохранить все 4 канала цветовой модели (не содержался альфа-канал), что было необходимо для достижения полноценной картины.

Поэтому было принято решение сохранить изображения с расширением ***PNG***, а затем воспользоваться сторонним ресурсом – конвертером файлов, с помощью которого удалось добиться необходимого результата.

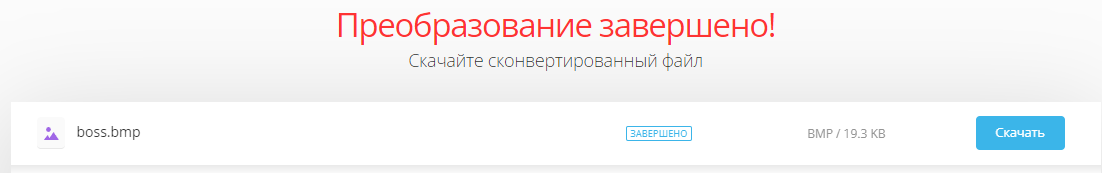


Рис. 7. Конвертирование текстур из PNG в BMP

После успешного сохранения выполненных текстур в формате BMP, в коде проекта производится единоразовая загрузка текстур при инициализации игры. Загрузка текстур выполняется с помощью функции ***stbi\_load()*** из библиотеки ***stb\_image.*** Это упрощает работу с каждой текстурой. Ниже приведен фрагмент кода загрузки одной из текстур главного меню.

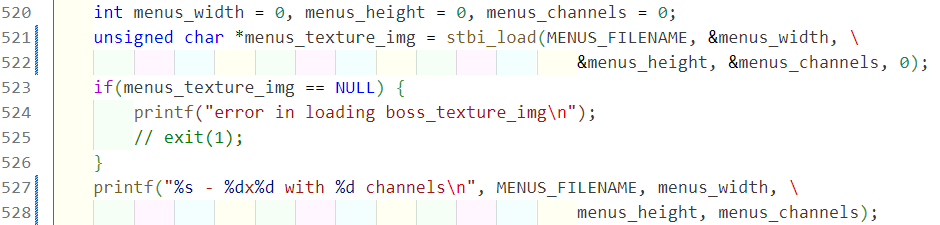


Рис. 8. Фрагмент кода инициализации текстуры старта игры в главном меню

Стоит отметить, что, с целью оптимизации загрузки текстур, при запуске игры, в память загружаются только текстуры меню, на случай если из игры будет сразу произведен выход и текстуры игровых объектов не понадобятся.

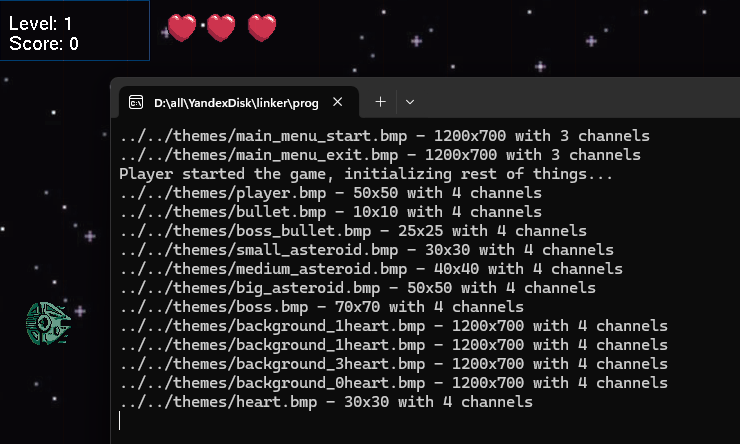


Рис. 9. Старт игры, сообщения в консоли

### **Звуки**

Для того чтобы логически завершить созданную игру, было решено добавить звуковое сопровождение для действий. Так как стилистика игры подобна 8-ми битным видеоиграм, были выбраны звуки из одной популярной игры такого рода - ***Undertale***.

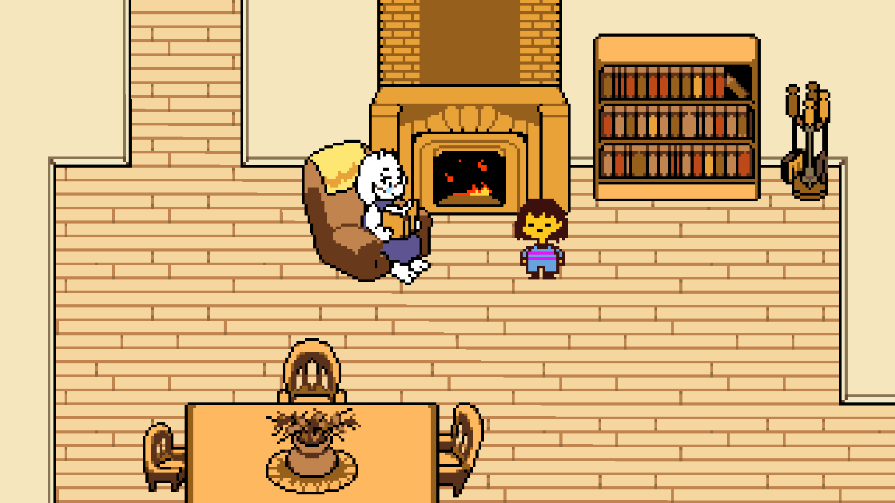


Рис. 10. Сцена из игры Undertale

Для того чтобы успешно проигрывать звуки, использовалась функция ***PlaySound().*** Она принимает в качестве аргументов путь от исполняемого файла до звука, в формате ***.wav,*** и некоторый набор аргументов для проигрывания звука. Ниже приведен пример использования функции для проигрывания музыки в главном меню.

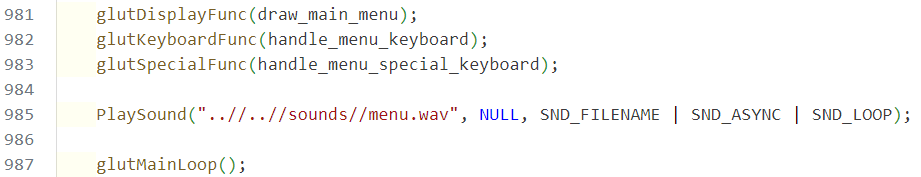


Рис. 11. Использование функции PlaySound для проигрывания звуков

Функция предоставляет для использования различные флаги, помогающие определить её поведение. В данном случае были использованы 3 флага.

1. ***SND\_FILENAME*** - сообщает функции, что звук передается по имени файла.
2. ***SND\_ASYNC*** - сообщает функции, что после начала проигрывания звука, стоит освободить процесс для других операций.
3. ***SND\_LOOP*** - сообщает функции, что звук, который она воспроизводит, необходимо запустить снова после его окончания.

Полный список флагов и их назначение находятся в официальной документации.

Единственным недостатком такого способа реализации проигрывания звуков, является невозможность проигрывать два звука одновременно. При попытке проиграть следующий звук, при условии того, что текущий еще не прекратился, текущий просто заменится следующим, не дойдя до своего конца.

## **ПРАКТИЧЕСКАЯ ЧАСТЬ**

**Структуры используемых объектов**

При создании игры, по мере надобности создавались структуры объектов, которые используются при игре. С кодом всех структур вы можете ознакомиться в приложении В. Приведем для примера реализацию некоторых из них.

*Структура “Игрок”.*

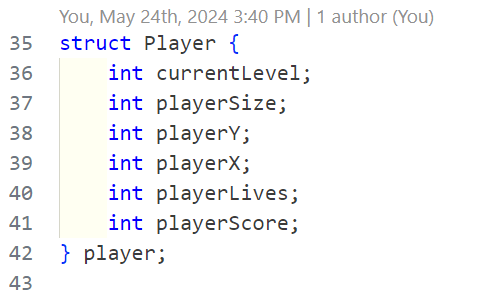


Рис. 12. Структура “Игрок”

В данной структуре используются поля, содержащие информацию о размерах объекта игрока, его положении в двухмерном пространстве, количестве набранных очков, достигнутом уровне и оставшихся жизнях в целочисленном виде.

Данным полям присваиваются стартовые значения в отдельной функции при старте игры.

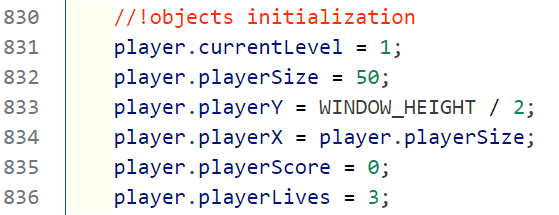


Рис. 13. Инициализация объекта игрока

В данном случае изначальная позиция игрока, определяемая левой нижней вершиной воображаемого квадрата, устанавливается в середине левой границы экрана.

*Структура “Пуля”.*

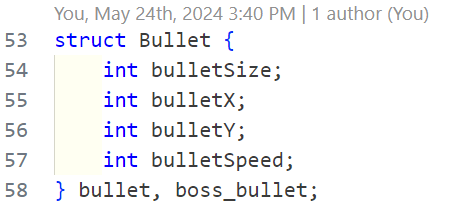


Рис. 14. Структура “Пуля”

В данной структуре также используются поля, содержащие информацию о размерах объекта игрока, его положении в двухмерном пространстве и скорости его полета.

Данным полям присваиваются стартовые значения в той же отдельной функции при старте игры. В ней инициализируются все используемые объекты.

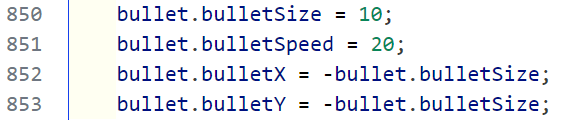


Рис. 15. Инициализация объекта пули

В данном случае, объекту пули, при инициализации, присваиваются “невозможные” значения положения в двухмерном пространстве, так как левый нижний угол игрового поля имеет координаты ***(0, 0)***. Таким образом, отрицательные координаты объекта позволяют с точностью утверждать, что на игровом поле этот объект не существует и, как следствие, не отображается.

### **Реализация механик движения и стрельбы**

Движение игрока осуществляется с помощью клавиш стрелок. Нажатия кнопок отслеживаются специально назначенной функцией при создании окна. Стоит отметить, что нажатия специальных кнопок (стрелки, F-клавиши и т.д.) в ***FreeGLUT*** предусмотрена отдельная привязка функции-слушателя.

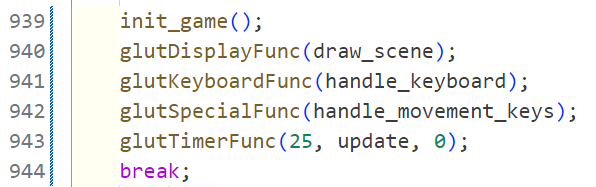


Рис. 16. Привязка функций-слушателей для отслеживания нажатий

При движении в каждом направлении, установлены границы. Таким образом, при нажатии какой-либо из кнопок движения, изменяются координаты объекта игрока и, если эти координаты выходят за границы заранее определенной игровой области, то игрок возвращается в максимально допустимую координату на этом направлении. Тем самым, игрок не может выйти за отведенные для него пределы поля.

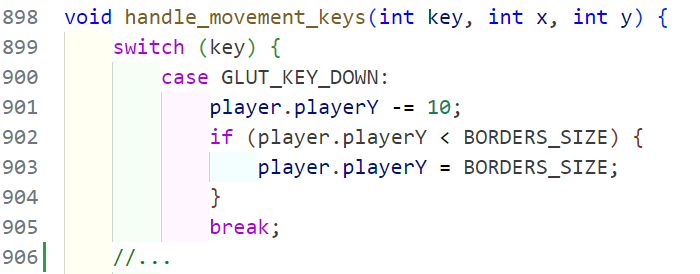


Рис. 17. Реализация движения вверх

Стрельба производится при нажатии кнопки ***“SPACE”***. Так как определен только 1 объект пули, то на поле одновременно не может находиться более 1 летящей пули.

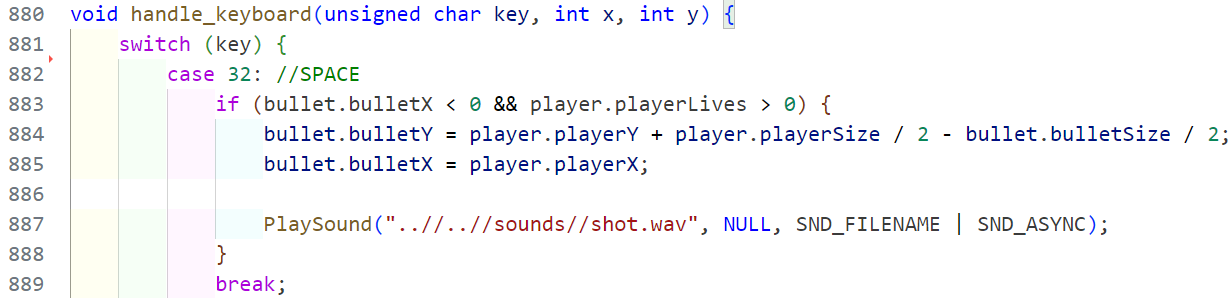


Рис. 18. Реализация стрельбы

При срабатывании оператора ***switch***, проверяется, может ли игрок стрелять (т.е. жив ли он) и не находится ли пуля уже на игровом поле. Затем проигрывается звук выстрела. Если все условия выполнены, то координаты пули меняются на координаты центра игрока, откуда она и начинает свое движение.

Обновление позиции пули происходит внутри функции, которая отвечает за обновление позиций и статусов всех объектов игры.

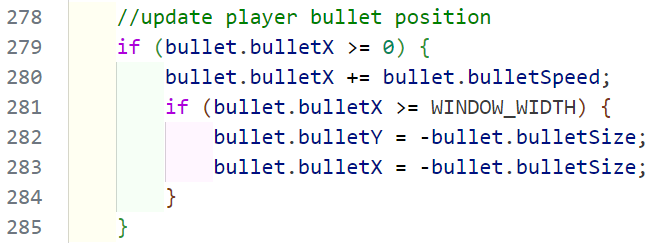


Рис. 19. Обновление позиции пули

Логика достаточно проста. Если пуля существует на игровом поле (её координаты положительны), то мы сдвигаем её посредством добавления к её координате по оси X значения её скорости. Если в какой-то момент, значение абсциссы превышает максимально допустимое, значит пуля улетела “за окно” и можно сбросить ее координату до изначальной.

Коллизии пули с любыми другими объектами обнаруживаются, если координаты пули находятся внутри координат объекта, с которым она столкнулась. В таком случае изменяются определенны данные (увеличивается счетчик очков, уменьшаются жизни босса и т.д.) и координата пули сбрасывается до изначальной. Для примера приведем код проверки коллизии пули с большим астероидом.

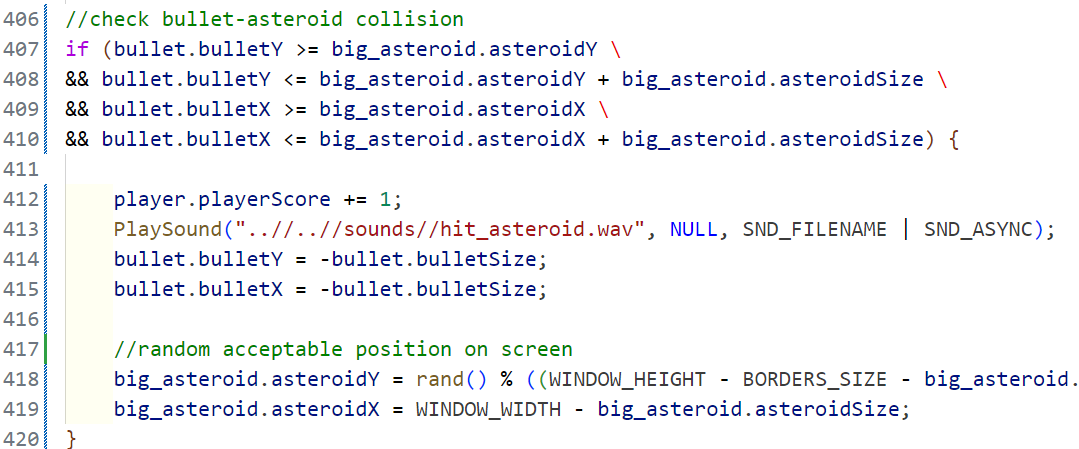


Рис. 20. Пример реализации отслеживания коллизий пули

### **Условия смены уровней**

Механика смены уровней очень проста и интуитивно понятна для игрока. При достижении определенных отметок набранных очков проигрывается звук повышения уровня и, вместе со значением уровня, увеличивается скорость летящих в игрока астероидов.

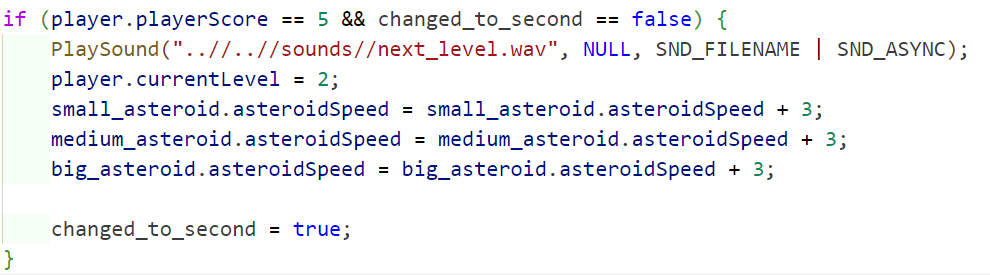


Рис. 21. Механика смены уровня сложности

При достижении 4-го уровня (15 очков) игра переходит на уровень с боссом. При этом, флаги, которые регулируют появления астероидов, опускаются, для того чтобы случайно не попасть в фантомно-летящий астероид при сражении с боссом.

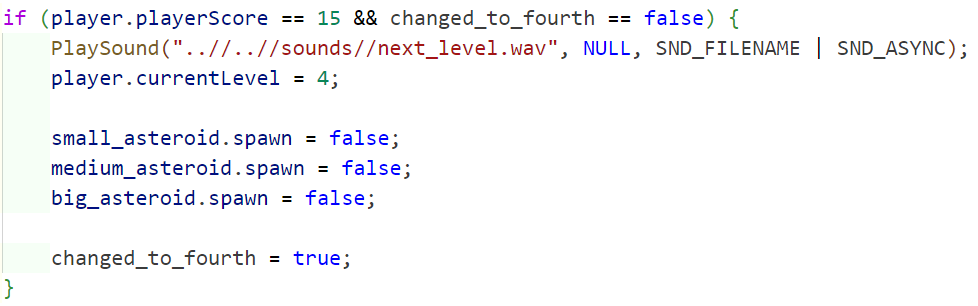


Рис. 22. Механика перехода на уровень с боссом

Также, при переходе на уровень с боссом, функция отрисовки теперь постоянно переходит в ветку в которой не нужно отображать астероиды и “сердца-бонусы” вовсе. Ей приходится рисовать только текст уровня, жизни босса, игрока и босса с их снарядами соответственно.

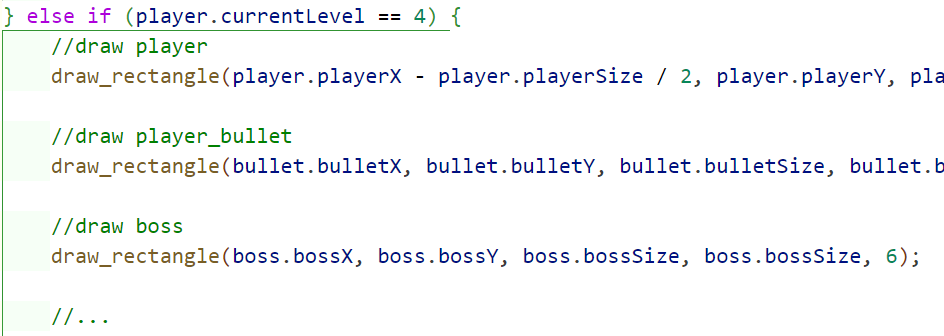


Рис. 23. Функция отрисовки на уровне с боссом

**Механики появления метеоритов**

Механики появления метеоритов идентичны друг другу, поэтому для примера приведем реализацию механики появления среднего астероида.

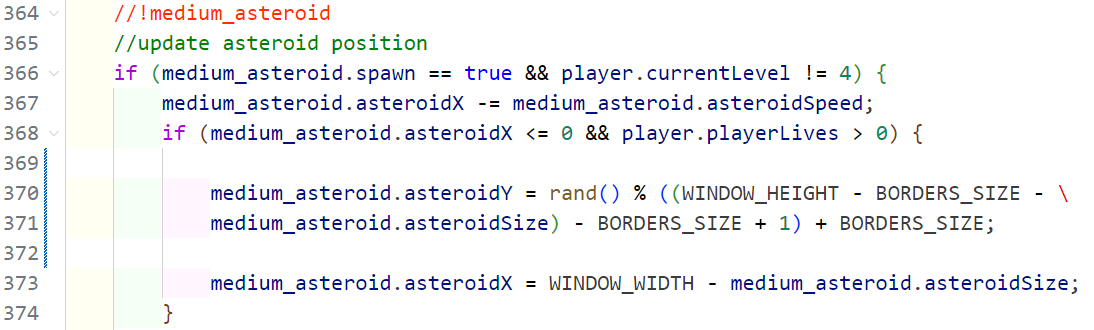


Рис. 24. реализация механики появления среднего астероида

Обновление положения астероида производится только в том случае, если флаг, сообщающий о том, надо ли это делать, поднят. В таком случае положение астероида по оси Х уменьшается на величину равную скорости его полета. Когда астероид достигает конца игровой области или же происходит коллизия с игроком или пулей, координаты астероида обновляются.

Положение астероида по оси Х задается максимально возможно дальним от игрока. Положение астероида по оси Y же задается случайно в пределах активной игровой области игрока таким образом, чтобы игрок смог поразить эти астероиды.

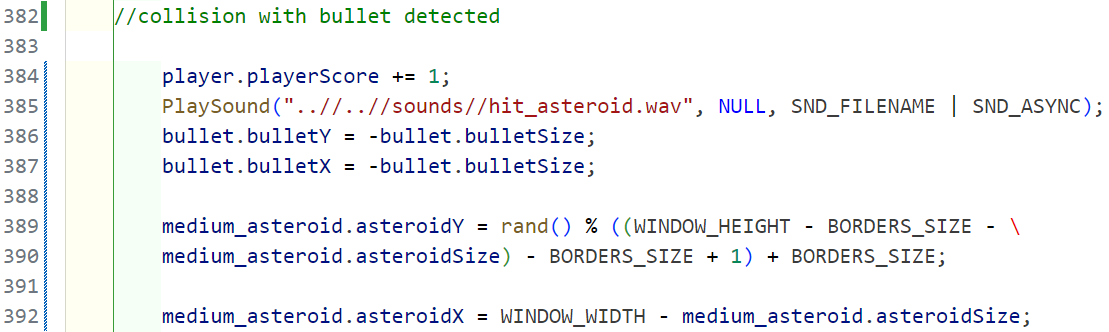


Рис. 25. Обновление позиции при коллизии с пулей

### **Механика бонуса - сердца**

Было принято решение добавить в игру бонус, поощряющий игрока за риск. Если у игрока не полное здоровье, т.е. менее ***3*** сердец, то с шансом ***30%*** при уничтожении самого маленького и как следствие самого быстрого астероида, выпадет бонус - сердце. Также, при выпадении сердца, проигрывается соответствующий звук. Если игрок соберет его, то к количеству его здоровья будет прибавлена 1.

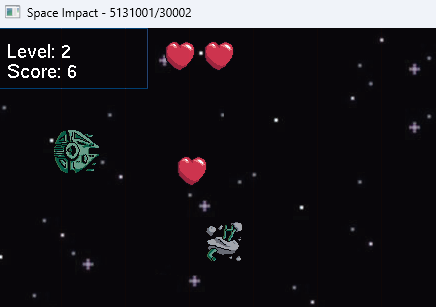


Рис. 26. Бонус-сердце в игре

Расчет шанса выпадения бонуса-сердца происходит в момент коллизии пули и малого астероида. Генерируется случайное число от ***0*** ***до 100***, и если оно больше ***70***, то сердце выпадает. Следовательно, вероятность того, что случайное число будет больше ***70*** примерно равна ***0.3***.

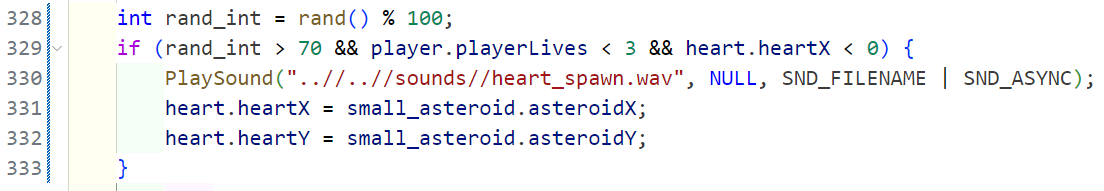


Рис. 27. Реализация выпадения бонуса-сердца

Обновление позиции бонуса-сердца происходит схожим с астероидом образом. Пока сердце не достигло левой видимой границы поля, оно двигается с заранее инициализированной скоростью. В противном случае его координаты обнуляются.

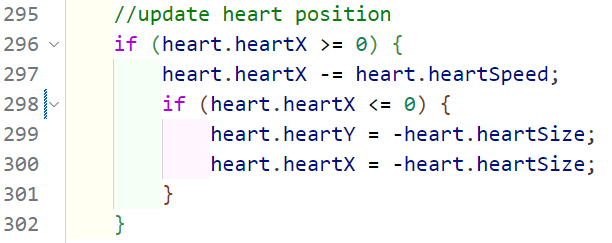


Рис. 28. Обновление позиции бонуса-сердца

**Условие победы/поражения**

У игры есть два возможных исхода. Вы можете завершить игру победив босса или же потеряв все очки здоровья. В обоих случаях, отрисовка всех объектов на экране прекращается, вы увидите свой финальный счет и предложение покинуть игру нажатием на ***ESC.***



Рис. 29. Экран поражения



Рис. 30. Экран победы

Если в какой-то момент, у босса или у игрока закончились жизни, проигрывается звук победы или поражения соответственно и отключается генерация астероидов.

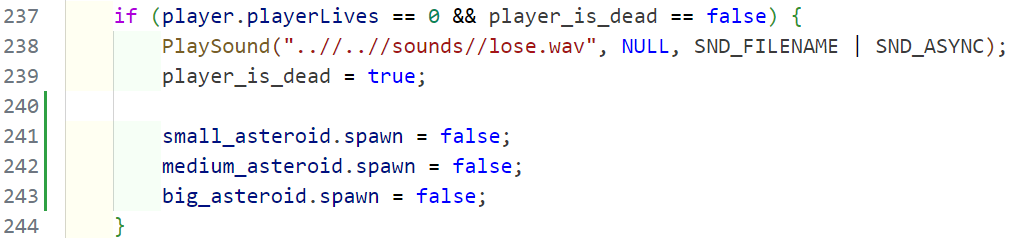


Рис. 31. Проигрывание звука поражения

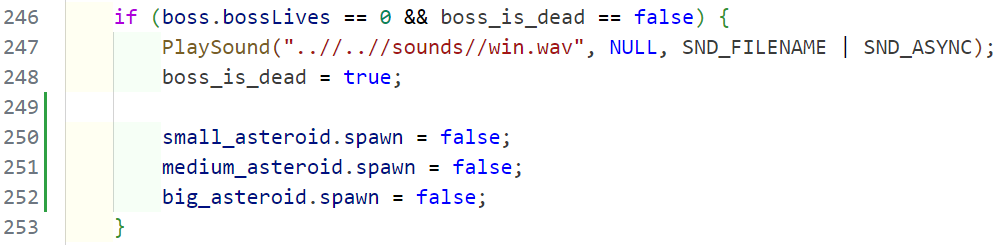


Рис. 32. Проигрывание звука победы

Флаги, информирующие о состоянии босса и игрока (***boss\_is\_dead*** и ***player\_is\_dead***) используются для того, чтобы единожды проиграть звук, так как функция обновления состояний вызывается гораздо чаще.

ЗАКЛЮЧЕНИЕ

В ходе выполнения данной работы были выполнены все поставленные задачи, а также достигнута цель работы. В процессе изучения соответствующей литературы при разработке продукта, были исследованы такие инструменты как библиотека утилит для приложений с OpenGL - ***FreeGLUT***, библиотека для загрузки и чтения картинок ***stb\_image,*** а также функции позволяющие проигрывать звуки.

Визуальный аспект продукта был учтён при разработке, поэтому были собственноручно нарисованы текстуры для всех видимых объектов игры, а также для элементов главного меню и заднего фона.

Был разработан прототип игры Space Impact (**приложении Б**) на языке Cи с использованием самодельных текстур (**приложение А**) и звуков из компьютерной игры Undertale.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

I Toby Fox. Компьютерная игра Undertale. Набор звуков из игры. 2013 – URL: <https://undertale.fandom.com/ru/wiki/Undertale_Soundtrack> - (дата обращения: 01.06.2024г.)

II Joey de Vries. Онлайн книга для изучения OpenGL. Перевод от Александра Ушанова. 2016 – URL: https://habr.com/ru/articles/310790/ - (дата обращения: 23.05.2024г.)

III Sean T. Barrett. Библиотека для загрузки картинок stm\_image. 2021 – URL: <https://github.com/planetack/stb_image/tree/master?tab=readme-ov-file> - (дата обращения: 28.06.2024г.)

IV Ковалев Роман Евгеньевич. Заметки автора о работе с OpenGL в VSCode. 2022г. - URL: <https://rekovalev.site/opengl-1-begin-vs-code/> - (дата обращения: 23.05.2024г.)

ПРИЛОЖЕНИЕ

**Приложение А**

**-** команда компиляции

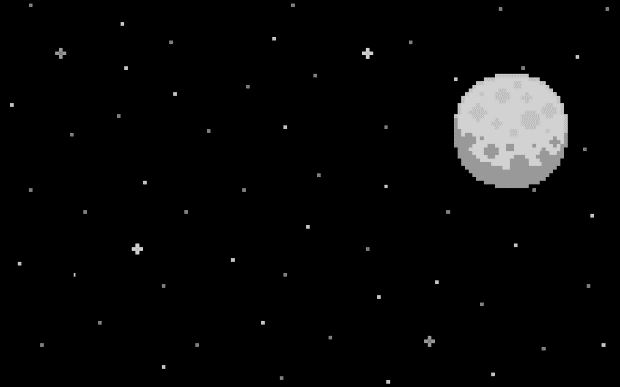
gcc ${workspaceRoot}/src/main.c -g --std=c99   
–I${workspaceRoot}/../dependencies/STB\_IMAGE/include  
-I${workspaceRoot}/../dependencies/GLFW/include  
-I${workspaceRoot}/../dependencies/FREEGLUT/include  
-I${workspaceRoot}/../dependencies/GLEW/include   
-lwinmm -lopengl32 –lglu32  
-L${workspaceRoot}/../dependencies/GLEW/bin/Release/x64  
-lglew32  
-L${workspaceRoot}/../dependencies/FREEGLUT/bin/x64  
-lfreeglut  
-L${workspaceRoot}/../dependencies/GLFW/lib-mingw-w64  
-lglfw3dll   
-o ${workspaceRoot}/builds/windows/space\_impact

**Приложение Б**

**-** нарисованные текстуры

**Приложение В**

**-** исходный код программы на языке С

#include <GL/glut.h>

#include <stdlib.h>

#include <stdio.h>

#include <stdbool.h>

#include <unistd.h>

#include <windows.h>

#include <mmsystem.h>

#define STB\_IMAGE\_IMPLEMENTATION

#include <stb\_image.h>

#define WINDOW\_WIDTH 1200

#define WINDOW\_HEIGHT 700

#define DRAW\_TEXT\_LENGHT 15

#define BORDERS\_SIZE 70

#define TEXTURES\_AMT 13

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

#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"

struct Player {

int currentLevel;

int playerSize;

int playerY;

int playerX;

int playerLives;

int playerScore;

} player;

struct Boss {

int bossSize;

int bossY;

int bossX;

int bossLives;

bool reached\_top;

bool reached\_bot;

} boss;

struct Bullet {

int bulletSize;

int bulletX;

int bulletY;

int bulletSpeed;

} bullet, boss\_bullet;

struct Asteroid {

bool spawn;

int asteroidSize;

int asteroidY;

int asteroidX;

int asteroidSpeed;

} small\_asteroid, medium\_asteroid, big\_asteroid;

struct Heart {

bool spawn;

int heartSize;

int heartY;

int heartX;

int heartSpeed;

} heart;

struct Menu {

int option;

} main\_menu;

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

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

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

glEnable(GL\_TEXTURE\_2D);

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();

glDisable(GL\_TEXTURE\_2D);

}

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);

//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);

}

glutSwapBuffers();

}

void draw\_scene() {

glClear(GL\_COLOR\_BUFFER\_BIT);

//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);

}

if (player.playerLives == 0) {

//draw background

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

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

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

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

char score[DRAW\_TEXT\_LENGHT];

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

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

char level[DRAW\_TEXT\_LENGHT];

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

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

char closing[DRAW\_TEXT\_LENGHT];

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

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

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

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

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

char score[DRAW\_TEXT\_LENGHT];

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

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

char level[DRAW\_TEXT\_LENGHT];

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

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

char closing[DRAW\_TEXT\_LENGHT];

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

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

//draw player\_bullet

draw\_rectangle(bullet.bulletX, bullet.bulletY, bullet.bulletSize, bullet.bulletSize, 1);

//draw boss

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

//draw boss\_bullet

draw\_rectangle(boss\_bullet.bulletX, boss\_bullet.bulletY, boss\_bullet.bulletSize, boss\_bullet.bulletSize, 2);

//draw level

char level[DRAW\_TEXT\_LENGHT];

sprintf(level, "Level: Boss");

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

//draw boss\_lives

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

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

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

} else {

//draw player

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

draw\_rectangle(bullet.bulletX, bullet.bulletY, bullet.bulletSize, bullet.bulletSize, 1);

//draw small\_asteroid

draw\_rectangle(small\_asteroid.asteroidX, small\_asteroid.asteroidY, small\_asteroid.asteroidSize, small\_asteroid.asteroidSize, 3);

//draw medium\_asteroid

draw\_rectangle(medium\_asteroid.asteroidX, medium\_asteroid.asteroidY, medium\_asteroid.asteroidSize, medium\_asteroid.asteroidSize, 4);

//draw big\_asteroid

draw\_rectangle(big\_asteroid.asteroidX, big\_asteroid.asteroidY, big\_asteroid.asteroidSize, big\_asteroid.asteroidSize, 5);

//draw level

char level[DRAW\_TEXT\_LENGHT];

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

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

//draw score

char score[DRAW\_TEXT\_LENGHT];

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

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

}

glutSwapBuffers();

}

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;

void update() {

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

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

player\_is\_dead = true;

}

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

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

boss\_is\_dead = true;

}

if (player.playerScore == 5 && changed\_to\_second == false) {

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

player.currentLevel = 2;

small\_asteroid.asteroidSpeed = small\_asteroid.asteroidSpeed + 3;

medium\_asteroid.asteroidSpeed = medium\_asteroid.asteroidSpeed + 3;

big\_asteroid.asteroidSpeed = big\_asteroid.asteroidSpeed + 3;

changed\_to\_second = true;

}

if (player.playerScore == 10 && changed\_to\_third == false) {

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

player.currentLevel = 3;

small\_asteroid.asteroidSpeed = small\_asteroid.asteroidSpeed + 3;

medium\_asteroid.asteroidSpeed = medium\_asteroid.asteroidSpeed + 3;

big\_asteroid.asteroidSpeed = big\_asteroid.asteroidSpeed + 3;

changed\_to\_third = true;

}

if (player.playerScore == 15 && changed\_to\_fourth == false) {

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

player.currentLevel = 4;

small\_asteroid.spawn = false;

medium\_asteroid.spawn = false;

big\_asteroid.spawn = false;

changed\_to\_fourth = true;

}

//update player bullet position

if (bullet.bulletX >= 0) {

bullet.bulletX += bullet.bulletSpeed;

if (bullet.bulletX >= WINDOW\_WIDTH) {

bullet.bulletY = -bullet.bulletSize;

bullet.bulletX = -bullet.bulletSize;

}

}

//update heart position

if (heart.heartX >= 0) {

heart.heartX -= heart.heartSpeed;

if (heart.heartX >= WINDOW\_WIDTH) {

heart.heartY = -heart.heartSize;

heart.heartX = -heart.heartSize;

}

}

//check heart-player collision

if (heart.heartY >= player.playerY \

&& heart.heartY <= player.playerY + player.playerSize \

&& heart.heartX >= player.playerX \

&& heart.heartX <= player.playerX + player.playerSize) {

player.playerLives += 1;

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

heart.heartY = -heart.heartSize;

heart.heartX = -heart.heartSize;

}

//!small\_asteroid

//update asteroid position

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

small\_asteroid.asteroidX -= small\_asteroid.asteroidSpeed;

if (small\_asteroid.asteroidX <= 0 && player.playerLives > 0) {

small\_asteroid.asteroidY = rand() % ((WINDOW\_HEIGHT - BORDERS\_SIZE - small\_asteroid.asteroidSize) - BORDERS\_SIZE + 1) + BORDERS\_SIZE;

small\_asteroid.asteroidX = WINDOW\_WIDTH - small\_asteroid.asteroidSize;

}

//check bullet-asteroid collision

if (bullet.bulletY >= small\_asteroid.asteroidY \

&& bullet.bulletY <= small\_asteroid.asteroidY + small\_asteroid.asteroidSize \

&& bullet.bulletX >= small\_asteroid.asteroidX \

&& bullet.bulletX <= small\_asteroid.asteroidX + small\_asteroid.asteroidSize) {

player.playerScore += 1;

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

bullet.bulletY = -bullet.bulletSize;

bullet.bulletX = -bullet.bulletSize;

int rand\_int = rand() % 100;

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

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

heart.heartX = small\_asteroid.asteroidX;

heart.heartY = small\_asteroid.asteroidY;

}

small\_asteroid.asteroidY = rand() % ((WINDOW\_HEIGHT - BORDERS\_SIZE - small\_asteroid.asteroidSize) - BORDERS\_SIZE + 1) + BORDERS\_SIZE;

small\_asteroid.asteroidX = WINDOW\_WIDTH - small\_asteroid.asteroidSize;

}

//check asteroid-player collision

if (small\_asteroid.asteroidY >= player.playerY \

&& small\_asteroid.asteroidY <= player.playerY + player.playerSize \

&& small\_asteroid.asteroidX >= player.playerX \

&& small\_asteroid.asteroidX <= player.playerX + player.playerSize) {

player.playerLives -= 1;

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

small\_asteroid.asteroidY = -small\_asteroid.asteroidSize;

small\_asteroid.asteroidX = -small\_asteroid.asteroidSize;

small\_asteroid.asteroidY = rand() % ((WINDOW\_HEIGHT - BORDERS\_SIZE - small\_asteroid.asteroidSize) - BORDERS\_SIZE + 1) + BORDERS\_SIZE;

small\_asteroid.asteroidX = WINDOW\_WIDTH - small\_asteroid.asteroidSize;

}

}

//!medium\_asteroid

//update asteroid position

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

medium\_asteroid.asteroidX -= medium\_asteroid.asteroidSpeed;

if (medium\_asteroid.asteroidX <= 0 && player.playerLives > 0) {

medium\_asteroid.asteroidY = rand() % ((WINDOW\_HEIGHT - BORDERS\_SIZE - medium\_asteroid.asteroidSize) - BORDERS\_SIZE + 1) + BORDERS\_SIZE;

medium\_asteroid.asteroidX = WINDOW\_WIDTH - medium\_asteroid.asteroidSize;

}

//check bullet-asteroid collision

if (bullet.bulletY >= medium\_asteroid.asteroidY \

&& bullet.bulletY <= medium\_asteroid.asteroidY + medium\_asteroid.asteroidSize \

&& bullet.bulletX >= medium\_asteroid.asteroidX \

&& bullet.bulletX <= medium\_asteroid.asteroidX + medium\_asteroid.asteroidSize) {

player.playerScore += 1;

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

bullet.bulletY = -bullet.bulletSize;

bullet.bulletX = -bullet.bulletSize;

medium\_asteroid.asteroidY = rand() % ((WINDOW\_HEIGHT - BORDERS\_SIZE - medium\_asteroid.asteroidSize) - BORDERS\_SIZE + 1) + BORDERS\_SIZE;

medium\_asteroid.asteroidX = WINDOW\_WIDTH - medium\_asteroid.asteroidSize;

}

//check asteroid-player collision

if (medium\_asteroid.asteroidY >= player.playerY \

&& medium\_asteroid.asteroidY <= player.playerY + player.playerSize \

&& medium\_asteroid.asteroidX >= player.playerX \

&& medium\_asteroid.asteroidX <= player.playerX + player.playerSize) {

player.playerLives -= 1;

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

medium\_asteroid.asteroidY = -medium\_asteroid.asteroidSize;

medium\_asteroid.asteroidX = -medium\_asteroid.asteroidSize;

medium\_asteroid.asteroidY = rand() % ((WINDOW\_HEIGHT - BORDERS\_SIZE - medium\_asteroid.asteroidSize) - BORDERS\_SIZE + 1) + BORDERS\_SIZE;

medium\_asteroid.asteroidX = WINDOW\_WIDTH - medium\_asteroid.asteroidSize;

}

}

//!big\_asteroid

//update asteroid position

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

big\_asteroid.asteroidX -= big\_asteroid.asteroidSpeed;

if (big\_asteroid.asteroidX <= 0 && player.playerLives > 0) {

big\_asteroid.asteroidY = rand() % ((WINDOW\_HEIGHT - BORDERS\_SIZE - big\_asteroid.asteroidSize) - BORDERS\_SIZE + 1) + BORDERS\_SIZE;

big\_asteroid.asteroidX = WINDOW\_WIDTH - big\_asteroid.asteroidSize;

}

//check bullet-asteroid collision

if (bullet.bulletY >= big\_asteroid.asteroidY \

&& bullet.bulletY <= big\_asteroid.asteroidY + big\_asteroid.asteroidSize \

&& bullet.bulletX >= big\_asteroid.asteroidX \

&& bullet.bulletX <= big\_asteroid.asteroidX + big\_asteroid.asteroidSize) {

player.playerScore += 1;

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

bullet.bulletY = -bullet.bulletSize;

bullet.bulletX = -bullet.bulletSize;

big\_asteroid.asteroidY = rand() % ((WINDOW\_HEIGHT - BORDERS\_SIZE - big\_asteroid.asteroidSize) - BORDERS\_SIZE + 1) + BORDERS\_SIZE;

big\_asteroid.asteroidX = WINDOW\_WIDTH - big\_asteroid.asteroidSize;

}

//check asteroid-player collision

if (big\_asteroid.asteroidY >= player.playerY \

&& big\_asteroid.asteroidY <= player.playerY + player.playerSize \

&& big\_asteroid.asteroidX >= player.playerX \

&& big\_asteroid.asteroidX <= player.playerX + player.playerSize) {

player.playerLives -= 1;

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

big\_asteroid.asteroidY = -big\_asteroid.asteroidSize;

big\_asteroid.asteroidX = -big\_asteroid.asteroidSize;

big\_asteroid.asteroidY = rand() % ((WINDOW\_HEIGHT - BORDERS\_SIZE - big\_asteroid.asteroidSize) - BORDERS\_SIZE + 1) + BORDERS\_SIZE;

big\_asteroid.asteroidX = WINDOW\_WIDTH - big\_asteroid.asteroidSize;

}

}

//! update boss

if (player.currentLevel == 4) {

//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\_bullet.bulletX < 0) {

boss\_bullet.bulletY = boss.bossY + boss.bossSize / 2 - boss\_bullet.bulletSize / 2;

boss\_bullet.bulletX = boss.bossX;

}

//update boss\_bullet position

if (boss\_bullet.bulletX >= 0) {

boss\_bullet.bulletX -= boss\_bullet.bulletSpeed;

if (boss\_bullet.bulletX <= 0) {

boss\_bullet.bulletY = -boss\_bullet.bulletSize;

boss\_bullet.bulletX = -boss\_bullet.bulletSize;

}

}

//boss\_bullet - player collision

if (boss\_bullet.bulletY >= player.playerY \

&& boss\_bullet.bulletY <= player.playerY + player.playerSize \

&& boss\_bullet.bulletX >= player.playerX \

&& boss\_bullet.bulletX <= player.playerX + player.playerSize) {

player.playerLives -= 1;

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

boss\_bullet.bulletY = -boss\_bullet.bulletSize;

boss\_bullet.bulletX = -boss\_bullet.bulletSize;

}

//bullet - boss collision

if (bullet.bulletY >= boss.bossY \

&& bullet.bulletY <= boss.bossY + boss.bossSize \

&& bullet.bulletX >= boss.bossX \

&& bullet.bulletX <= boss.bossX + boss.bossSize) {

boss.bossLives -= 1;

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

bullet.bulletY = -bullet.bulletSize;

bullet.bulletX = -bullet.bulletSize;

}

}

glutPostRedisplay();

glutTimerFunc(25, update, 0);

}

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 = 1;

player.playerSize = 50;

player.playerY = WINDOW\_HEIGHT / 2;

player.playerX = player.playerSize;

player.playerScore = 0;

player.playerLives = 3;

boss.bossLives = 3;

boss.bossSize = 70;

boss.bossY = WINDOW\_HEIGHT / 2;

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

boss\_bullet.bulletSize = 25;

boss\_bullet.bulletSpeed = 10;

boss\_bullet.bulletX = -boss\_bullet.bulletSize;

boss\_bullet.bulletY = -boss\_bullet.bulletSize;

boss.reached\_top = true;

boss.reached\_bot = false;

bullet.bulletSize = 10;

bullet.bulletSpeed = 20;

bullet.bulletX = -bullet.bulletSize;

bullet.bulletY = -bullet.bulletSize;

small\_asteroid.asteroidSize = 30;

small\_asteroid.spawn = true;

small\_asteroid.asteroidSpeed = 4;

small\_asteroid.asteroidY = -small\_asteroid.asteroidSize;

small\_asteroid.asteroidX = WINDOW\_WIDTH - small\_asteroid.asteroidSize;

heart.heartSize = 30;

heart.spawn = true;

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

heart.heartY = -heart.heartSize;

heart.heartSpeed = 4;

medium\_asteroid.asteroidSize = 40;

medium\_asteroid.spawn = true;

medium\_asteroid.asteroidSpeed = 3;

medium\_asteroid.asteroidY = -medium\_asteroid.asteroidSize;

medium\_asteroid.asteroidX = WINDOW\_WIDTH - medium\_asteroid.asteroidSize;

big\_asteroid.asteroidSize = 50;

big\_asteroid.spawn = true;

big\_asteroid.asteroidSpeed = 2;

big\_asteroid.asteroidY = -big\_asteroid.asteroidSize;

big\_asteroid.asteroidX = WINDOW\_WIDTH - big\_asteroid.asteroidSize;

}

void handle\_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 32: //SPACE

if (bullet.bulletX < 0 && player.playerLives > 0) {

bullet.bulletY = player.playerY + player.playerSize / 2 - bullet.bulletSize / 2;

bullet.bulletX = player.playerX;

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

}

break;

}

}

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

switch (key) {

case GLUT\_KEY\_DOWN:

player.playerY -= 10;

if (player.playerY < BORDERS\_SIZE) {

player.playerY = BORDERS\_SIZE;

}

break;

case GLUT\_KEY\_UP:

player.playerY += 10;

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

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

}

break;

case GLUT\_KEY\_RIGHT:

player.playerX += 10;

if (player.playerX > player.playerSize + 200) {

player.playerX = player.playerSize + 200;

}

break;

case GLUT\_KEY\_LEFT:

player.playerX -= 10;

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) {

PlaySound(NULL, NULL, SND\_ASYNC);

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();

}

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);

main\_menu\_init();

glutDisplayFunc(draw\_main\_menu);

glutKeyboardFunc(handle\_menu\_keyboard);

glutSpecialFunc(handle\_menu\_special\_keyboard);

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

glutMainLoop();

return 0;

}