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

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:

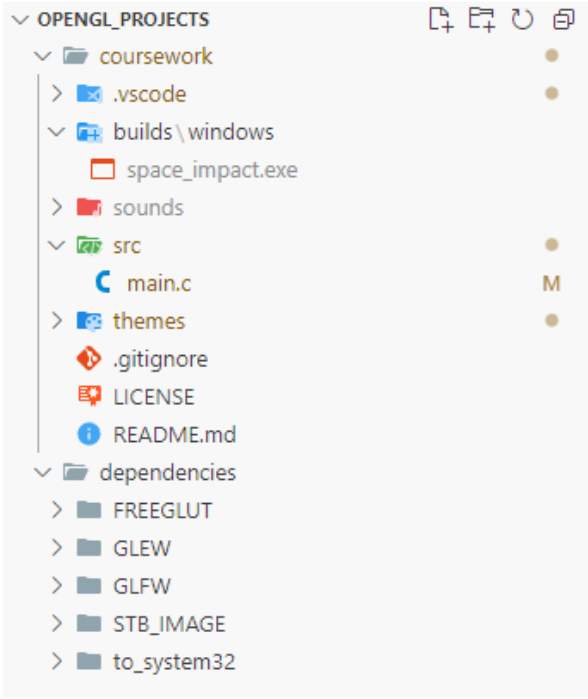


Рис. 5. Структура каталога проекта

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.

2. 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***.

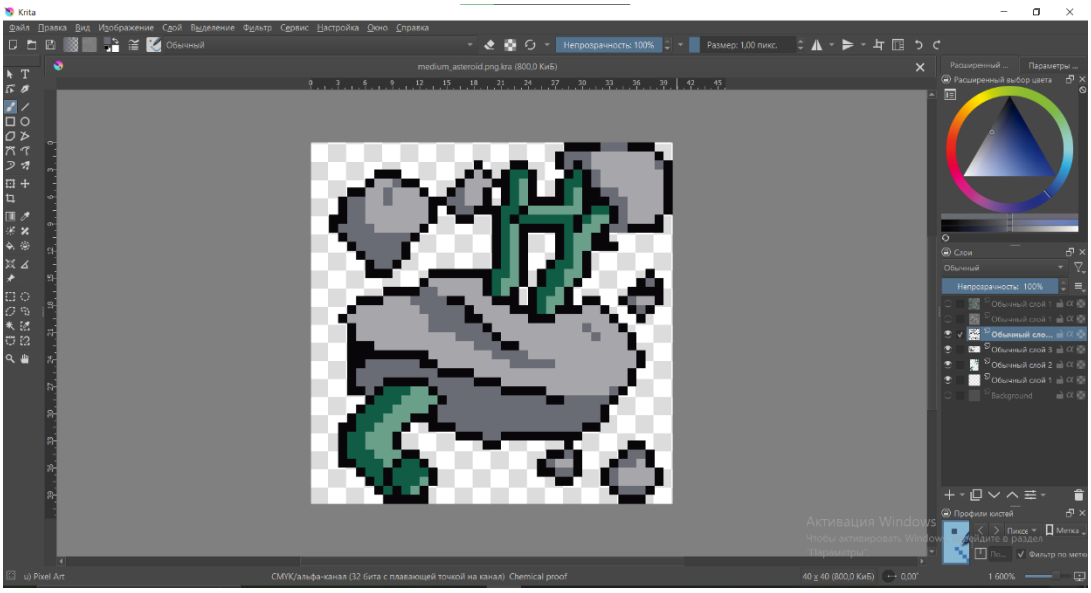


Рис. 6. Растровый графический редактор Krita

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

1. Игрок – **50x50** пикс.
2. Пуля игрока – **10x10** пикс.
3. Малый астероид – **30x30** пикс.
4. Средний астероид – **40x40** пикс.
5. Большой астероид – **50x50** пикс.
6. Босс – **50x50** пикс.
7. Пуля босса – **25x25** пикс.
8. Бонус – **30x30** пикс.
9. Фоны с сердцами (4 шт.) – **1200x700** 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.

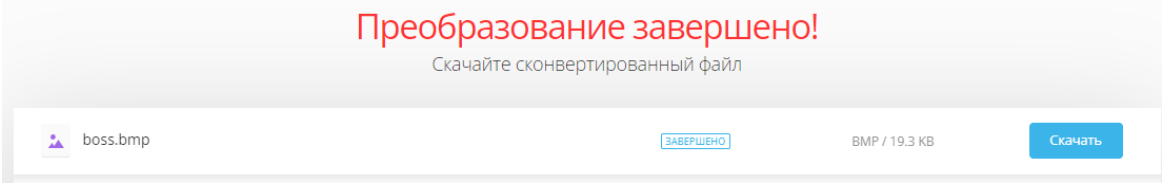


Рис. 7. Конвертирование текстур из PNG в BMP

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.

```
520     int menus_width = 0, menus_height = 0, menus_channels = 0;
521     unsigned char *menus_texture_img = stbi_load(MENUS_FILENAME, &menus_width, \
522                                                  &menus_height, &menus_channels, 0);
523     if(menus_texture_img == NULL) {
524         printf("error in loading boss_texture_img\n");
525         // exit(1);
526     }
527     printf("%s - %dx%d with %d channels\n", MENUS_FILENAME, menus_width, \
528                                             menus_height, menus_channels);
```

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.

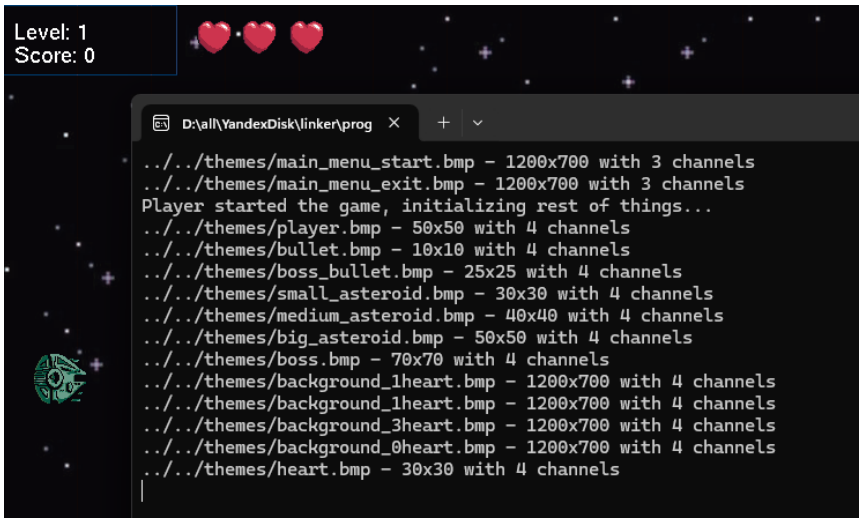


Рис. 9. Старт игры, сообщения в консоли

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.

```
981     glutDisplayFunc(draw_main_menu);
982     glutKeyboardFunc(handle_menu_keyboard);
983     glutSpecialFunc(handle_menu_special_keyboard);
984
985     PlaySound("..//..//sounds//menu.wav", NULL, SND_FILENAME | SND_ASYNC | SND_LOOP);
986
987     glutMainLoop();
```

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”.

```
You, May 24th, 2024 3:40 PM | 1 author (You)
35  struct Player {
36      int currentLevel;
37      int playerSize;
38      int playerY;
39      int playerX;
40      int playerLives;
41      int playerScore;
42  } player;
```

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.

```
830     //!objects initialization
831     player.currentLevel = 1;
832     player.playerSize = 50;
833     player.playerY = WINDOW_HEIGHT / 2;
834     player.playerX = player.playerSize;
835     player.playerScore = 0;
836     player.playerLives = 3;
```

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”.

```
You, May 24th, 2024 3:40 PM | 1 author (You)
53  struct Bullet {
54      int bulletSize;
55      int bulletX;
56      int bulletY;
57      int bulletSpeed;
58  } bullet, boss_bullet;
```

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.

```
850     bullet.bulletSize = 10;
851     bullet.bulletSpeed = 20;
852     bullet.bulletX = -bullet.bulletSize;
853     bullet.bulletY = -bullet.bulletSize;
```

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.

```
939     init_game();
940     glutDisplayFunc(draw_scene);
941     glutKeyboardFunc(handle_keyboard);
942     glutSpecialFunc(handle_movement_keys);
943     glutTimerFunc(25, update, 0);
944     break;
```

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.

```
898 void handle_movement_keys(int key, int x, int y) {
899     switch (key) {
900         case GLUT_KEY_DOWN:
901             player.playerY -= 10;
902             if (player.playerY < BORDERS_SIZE) {
903                 player.playerY = BORDERS_SIZE;
904             }
905             break;
906     //...
```

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.

```
880 void handle_keyboard(unsigned char key, int x, int y) {
881     switch (key) {
882         case 32: //SPACE
883             if (bullet.bulletX < 0 && player.playerLives > 0) {
884                 bullet.bulletY = player.playerY + player.playerSize / 2 - bullet.bulletSize / 2;
885                 bullet.bulletX = player.playerX;
886 
887                 PlaySound("..//..//sounds//shot.wav", NULL, SND_FILENAME | SND_ASYNC);
888             }
889             break;
```

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.

```
278     //update player bullet position
279     if (bullet.bulletX >= 0) {
280         bullet.bulletX += bullet.bulletSpeed;
281         if (bullet.bulletX >= WINDOW_WIDTH) {
282             bullet.bulletY = -bullet.bulletSize;
283             bullet.bulletX = -bullet.bulletSize;
284         }
285     }
```

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.

```
406 //check bullet-asteroid collision
407 if (bullet.bulletY >= big_asteroid.asteroidY \
408 && bullet.bulletY <= big_asteroid.asteroidY + big_asteroid.asteroidSize \
409 && bullet.bulletX >= big_asteroid.asteroidX \
410 && bullet.bulletX <= big_asteroid.asteroidX + big_asteroid.asteroidSize) {
411
412     player.playerScore += 1;
413     PlaySound("..//..//sounds//hit_asteroid.wav", NULL, SND_FILENAME | SND_ASYNC);
414     bullet.bulletY = -bullet.bulletSize;
415     bullet.bulletX = -bullet.bulletSize;
416
417     //random acceptable position on screen
418     big_asteroid.asteroidY = rand() % ((WINDOW_HEIGHT - BORDERS_SIZE - big_asteroid.
419     big_asteroid.asteroidX = WINDOW_WIDTH - big_asteroid.asteroidSize;
420 }
```

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.

```
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;
}
```

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.

```
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;
}
```

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.

```
} else if (player.currentLevel == 4) {
    //draw player
    draw_rectangle(player.playerX - player.playerSize / 2, player.playerY, pla

    //draw player_bullet
    draw_rectangle(bullet.bulletX, bullet.bulletY, bullet.bulletSize, bullet.b

    //draw boss
    draw_rectangle(boss.bossX, boss.bossY, boss.bossSize, boss.bossSize, 6);

    //...
```

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.

```
364     //!medium_asteroid
365     //update asteroid position
366     if (medium_asteroid.spawn == true && player.currentLevel != 4) {
367         medium_asteroid.asteroidX -= medium_asteroid.asteroidSpeed;
368         if (medium_asteroid.asteroidX <= 0 && player.playerLives > 0) {
369
370             medium_asteroid.asteroidY = rand() % ((WINDOW_HEIGHT - BORDERS_SIZE - \
371             medium_asteroid.asteroidSize) - BORDERS_SIZE + 1) + BORDERS_SIZE;
372
373             medium_asteroid.asteroidX = WINDOW_WIDTH - medium_asteroid.asteroidSize;
374         }
```

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 X 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 X 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.

```
382     //collision with bullet detected
383
384         player.playerScore += 1;
385         PlaySound("..//..//sounds//hit_asteroid.wav", NULL, SND_FILENAME | SND_ASYNC);
386         bullet.bulletY = -bullet.bulletSize;
387         bullet.bulletX = -bullet.bulletSize;
388
389         medium_asteroid.asteroidY = rand() % ((WINDOW_HEIGHT - BORDERS_SIZE - \
390         medium_asteroid.asteroidSize) - BORDERS_SIZE + 1) + BORDERS_SIZE;
391
392         medium_asteroid.asteroidX = WINDOW_WIDTH - medium_asteroid.asteroidSize;
```

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.

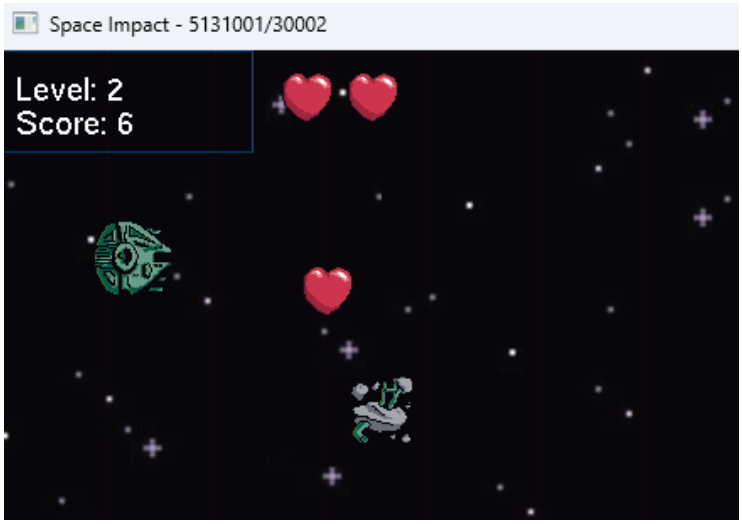


Рис. 26. Бонус-сердце в игре

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***.

```
328     int rand_int = rand() % 100;
329     if (rand_int > 70 && player.playerLives < 3 && heart.heartX < 0) {
330         PlaySound("..//..//sounds//heart_spawn.wav", NULL, SND_FILENAME | SND_ASYNC);
331         heart.heartX = small_asteroid.asteroidX;
332         heart.heartY = small_asteroid.asteroidY;
333     }
```

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.

```
295     //update heart position
296     if (heart.heartX >= 0) {
297         heart.heartX -= heart.heartSpeed;
298         if (heart.heartX <= 0) {
299             heart.heartY = -heart.heartSize;
300             heart.heartX = -heart.heartSize;
301         }
302     }
```

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.

```
237     if (player.playerLives == 0 && player_is_dead == false) {
238         PlaySound("..//..//sounds//lose.wav", NULL, SND_FILENAME | SND_ASYNC);
239         player_is_dead = true;
240 
241         small_asteroid.spawn = false;
242         medium_asteroid.spawn = false;
243         big_asteroid.spawn = false;
244     }
```

Рис. 31. Проигрывание звука поражения

```
246     if (boss.bossLives == 0 && boss_is_dead == false) {
247         PlaySound("..//..//sounds//win.wav", NULL, SND_FILENAME | SND_ASYNC);
248         boss_is_dead = true;
249 
250         small_asteroid.spawn = false;
251         medium_asteroid.spawn = false;
252         big_asteroid.spawn = false;
253     }
```

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 Си 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

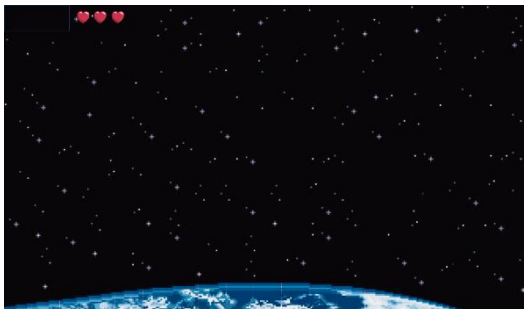
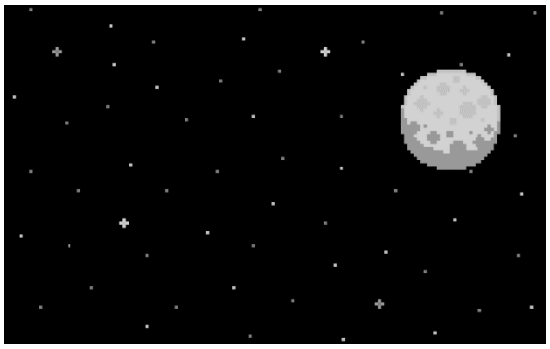
- команда компиляции

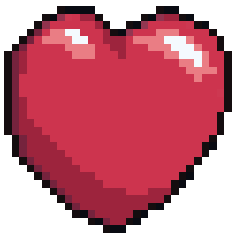
```
gcc ${workspaceRoot}/src/main.c -g --std=c99
–I${workspaceRoot}/../dependencies/STB_IMAGE/include
-I${workspaceRoot}/../dependencies/GLFW/include
-I${workspaceRoot}/../dependencies/FREEGLUT/include
-I${workspaceRoot}/../dependencies/GLEW/include
-lwinmm -lopengl32 –lglu32
-L${workspaceRoot}/../dependencies/GLEW/bin/Release/x64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

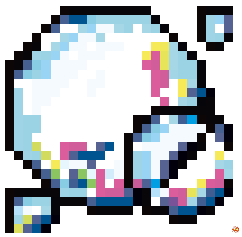
Приложение Б

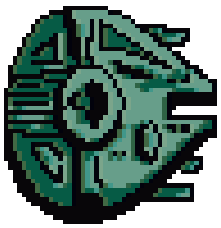
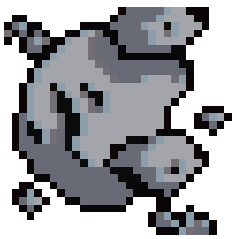
- нарисованные текстуры







Приложение В

- исходный код программы на языке C

```
#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;
}
```