Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное
учреждение высшего образования
«Санкт-Петербургский политехнический университет Петра
Великого»
—
Институт компьютерных наук и кибербезопасности
Высшая школа кибербезопасности

К У Р С О В А Я Р А Б О Т А

Space Impact

по дисциплине «Языки программирования»

Выполнили
студенты гр. 5131001/30002 Мишенев Н. С.
Квашенникова В. М.

Руководитель
программист Малышев Е. В.

«___» _____ 2024 г.

Санкт-Петербург
2024г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель

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

Задачи

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

ОСНОВНАЯ ЧАСТЬ

Теоретическая часть

Реструктуризация проекта

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

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

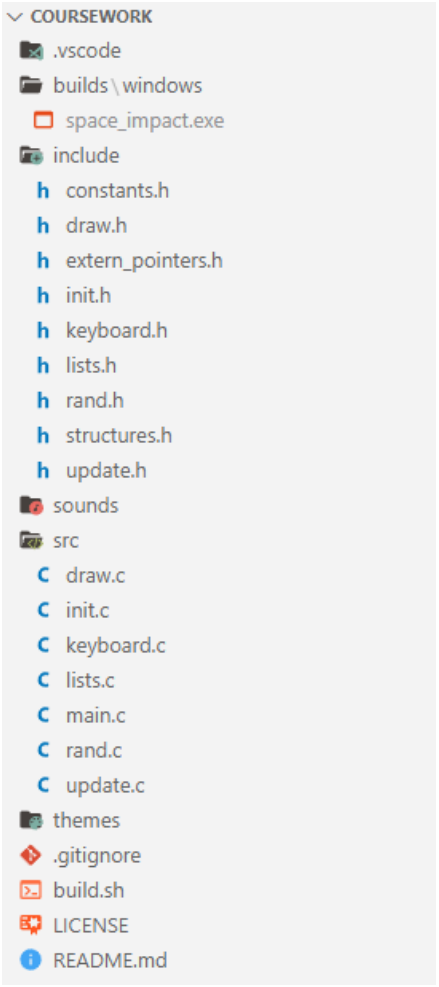

Рис. 1. Структура каталога проекта после внесения изменений.

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

Изменения механик игры

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

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

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

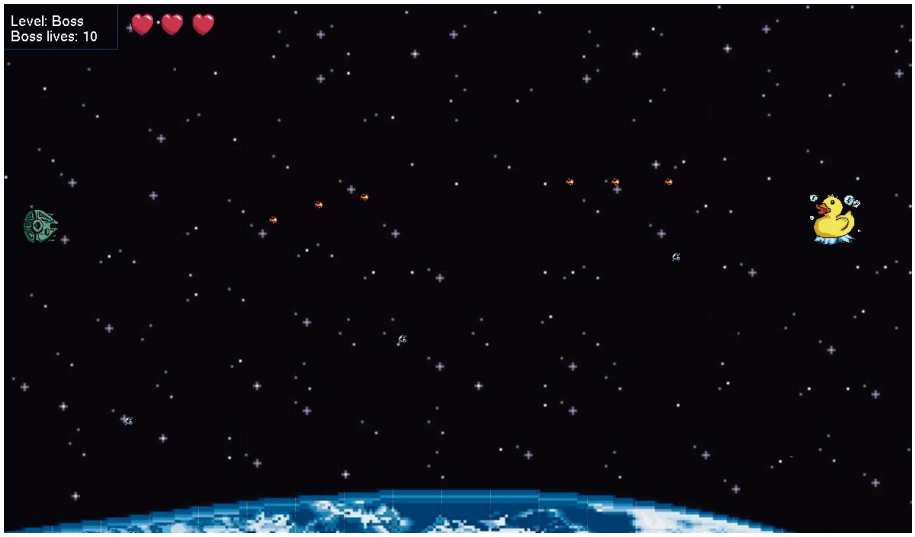

Рис. 2. Битва с боссом.

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

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

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

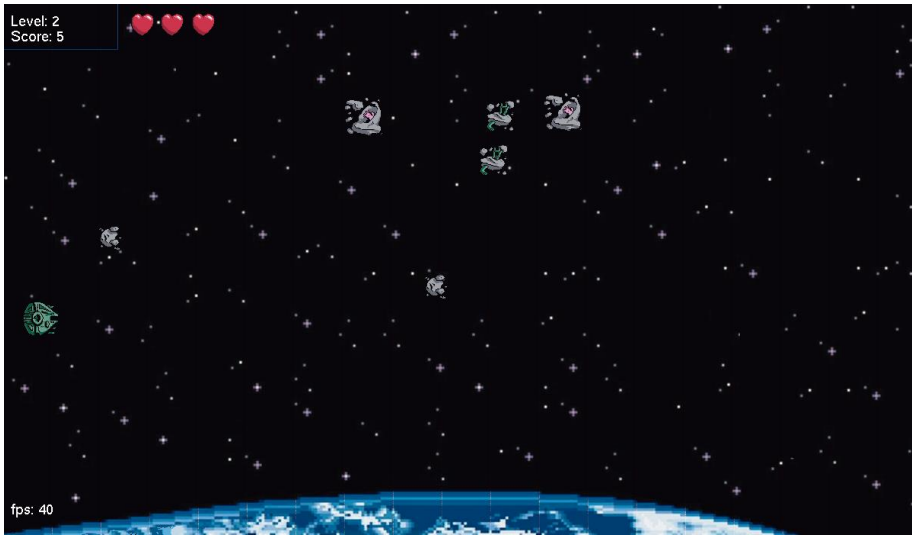

Рис. 3. Демонстрация второго уровня.

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

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

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

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

Рис. 4. Демонстрация перехода на третий уровень.

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

- **Изменено количество очков здоровья у босса**

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

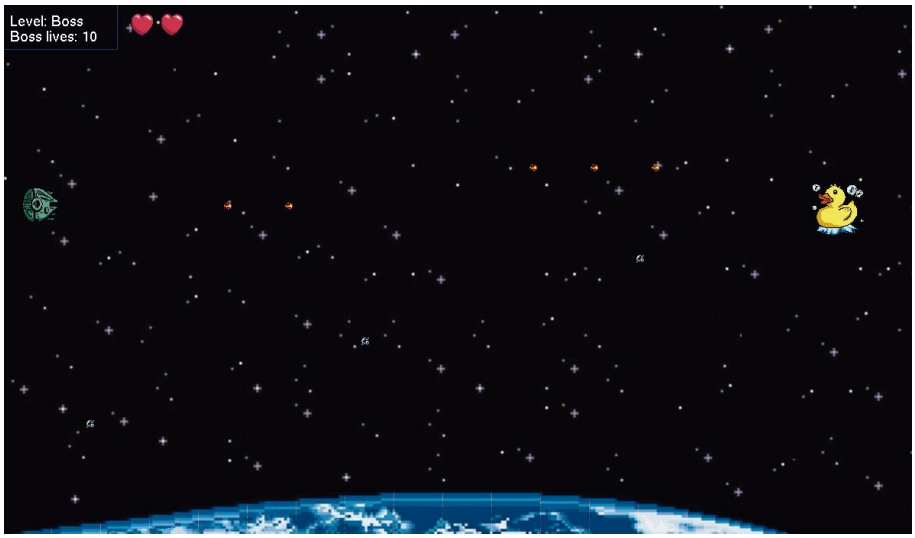

Рис. 5. Демонстрация уровня с боссом.

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

- **Добавлен режим “бенчмарк”**

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

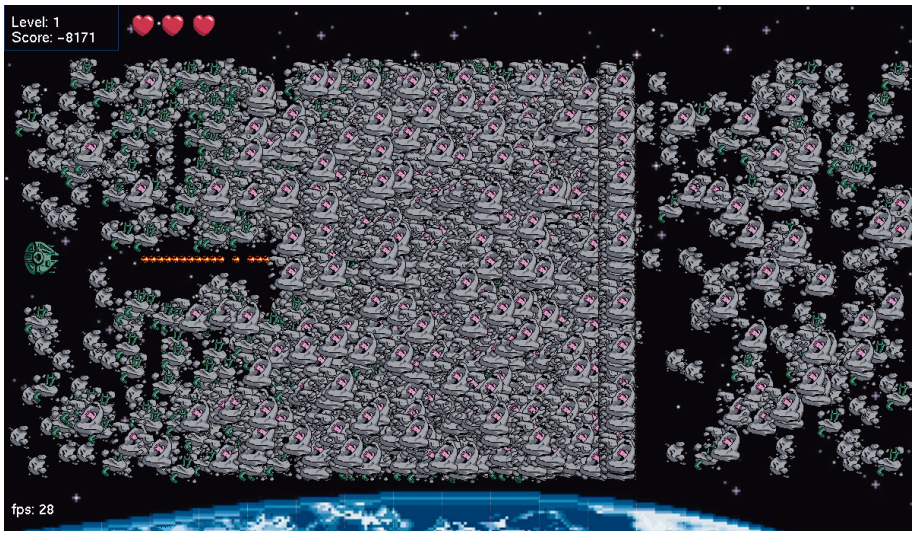

Рис. 6. Демонстрация режима “бенчмарк”.

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

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

Оценка сложности неоптимизированного алгоритма

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

```
336 void check_bullet_asteroid_collisions(struct Asteroid_list *asteroids) {
337     struct Asteroid_list *current_asteroid = asteroids;
338     struct Bullet_list *current_bullet = bullets;
339 
340     while (current_bullet != NULL) {
341         current_asteroid = asteroids;
342         while (current_asteroid != NULL) {
343             if (is_colliding_ba(current_bullet->bullet, current_asteroid->asteroid) &&
344                 is_bullet_on_screen(current_bullet->bullet) &&
345                 is_asteroid_on_screen(current_asteroid->asteroid)) {
346 
347                 player.playerScore++;
348                 if (!megalovania_is_playing) {
349                     PlaySound("..//..//sounds//hit_asteroid.wav", NULL, SND_FILENAME | SND_ASYNC);
350                 }
351 
352                 remove_from_blist(bullets, current_bullet->bullet);
353 
354                 if (asteroids == small_asteroids) {
355                     maybe_spawn_heart(current_asteroid);
356                 }
357 
358                 current_asteroid->asteroid->asteroidY = get_random_number() % ((WINDOW_HEIGHT - BORDERS_SIZE - \
359                                                         current_asteroid->asteroid->asteroidSize) - BORDERS_SIZE + 1) + BORDERS_SIZE;
360                 current_asteroid->asteroid->asteroidX = WINDOW_WIDTH - current_asteroid->asteroid->asteroidSize;
361 
362                 break;
363             }
364             current_asteroid = current_asteroid->next;
365         }
366         current_bullet = current_bullet->next;
367     }
368 }
```

Рис. 7. Функция проверки коллизии пуль с астероидами.

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

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

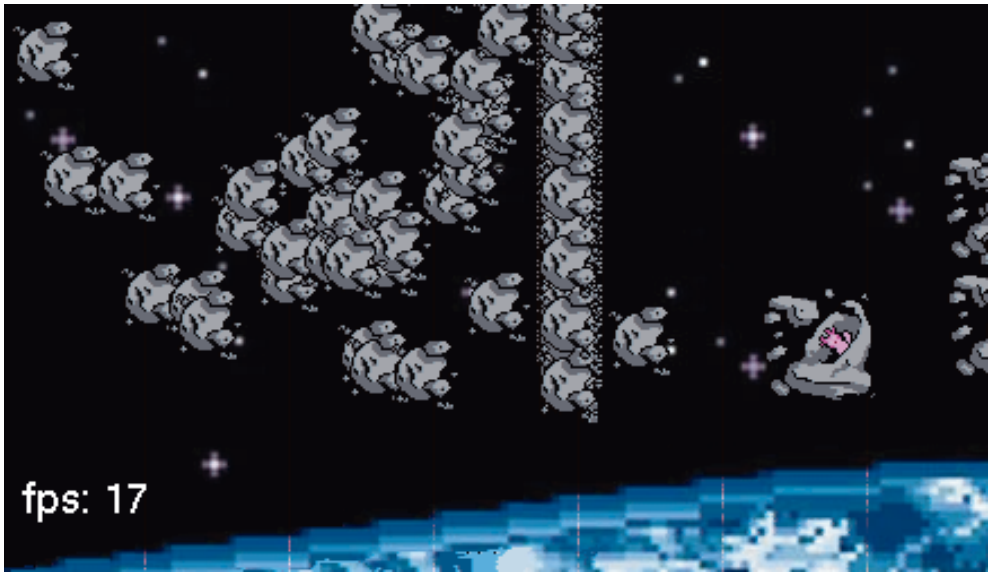

Рис. 9. Оценка FPS для неоптимизированного алгоритма в режиме “бенчмарк” для 100000 астероидов.

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

Практическая часть

Алгоритмическая оптимизация

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

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

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

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

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

```
24  struct Bullet_list *bullet_array[LINE_COUNT];
25  struct Bullet_list *boss_bullet_array[LINE_COUNT];
26  struct Asteroid_list *small_asteroids_array[LINE_COUNT];
27  struct Asteroid_list *medium_asteroids_array[LINE_COUNT];
28  struct Asteroid_list *big_asteroids_array[LINE_COUNT];
```

Рис. 10. Новые структуры данных.

А также были внесены некоторые изменения в старые структуры данных:

```
26 struct Bullet {
27     int bulletSize;
28     int bulletX;
29     int bulletY;
30     int bulletSpeed;
31+    bool between;
32 };
33
34 struct Asteroid {
35     int asteroidSize;
36     int asteroidY;
37     int asteroidX;
38     int asteroidSpeed;
39+    int line;
40 };
```

Рис. 11. Изменения в старых структурах данных.

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

Особое внимание уделено добавлению пуль:

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

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

```
29  void add_bullet(struct Bullet_list **b_array) {
30      bullet_count++;
31      int y_coordinate = player.playerY + player.playerSize / 2 - B_SIZE / 2;
32      int line_number = (y_coordinate - BORDERS_SIZE) / BA_SIZE;
33      bool flag_between = false;
34      struct Bullet_list *temp = b_array[line_number];
35      struct Bullet *bullet;
36
37      while (true) {
38          if (is_bullet_on_screen(temp->bullet) && temp->next == NULL) {…
41          } else if (is_bullet_on_screen(temp->bullet) && temp->next != NULL) {…
43          } else if (!is_bullet_on_screen(temp->bullet)) {
44              temp->bullet->bulletY = y_coordinate;
45              temp->bullet->bulletX = player.playerX;
46              bullet = temp->bullet;
47              if(((y_coordinate + bullet->bulletSize - BORDERS_SIZE) / BA_SIZE) != line_number){
48                  flag_between = true;
49              }
50              bullet->between = flag_between;
51
52              break;
53          }
54      }
55
56      if (flag_between){ //bullet between lines
57          temp = b_array[line_number+1];
58          while (true) {…
70      }
71  }
```

Рис. 12. Разработанная функция добавления пули.

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

Главным преобразованием в работе стало изменение функций проверки коллизий.

- **Функция проверки коллизии пуль с астероидами**

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

Циклы проверки:

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

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

```
488  void check_bullet_asteroid_collisions(struct Asteroid_list **asteroids) {
489      for (int i = 0; i < LINE_COUNT; i++)
490      {
491          struct Asteroid_list *current_asteroid = asteroids[i]->next;
492          struct Bullet_list *current_bullet = bullet_array[i];
493          while (current_bullet != NULL) {
494              while (current_asteroid != NULL) {
495                  if (is_colliding_ba(current_bullet->bullet, current_asteroid->asteroid) &&
496                      is_bullet_on_screen(current_bullet->bullet) &&
497                      is_asteroid_on_screen(current_asteroid->asteroid)) {
498
499                      player.playerScore++;
500                      if (!megalovania_is_playing) {
501                          PlaySound("..//..//sounds//hit_asteroid.wav", NULL, SND_FILENAME | SND_ASYNC);
502                          if (asteroids == small_asteroids_array){
503                              maybe_spawn_heart(current_asteroid);
504                          }
505                      }
506
507                      remove_from_blist(bullet_array[i], current_bullet->bullet);
508
509                      if ((i+1) < LINE_COUNT && current_bullet->bullet->between){
510                          remove_from_blist(bullet_array[i+1], current_bullet->bullet);
511                      }
512                      bullet_count--;
513
514                      int new_line_number = get_random_number() % LINE_COUNT;
515
516                      remove_from_alist(asteroids[i],current_asteroid->asteroid);
517                      add_to_alist(asteroids[new_line_number], current_asteroid);
518
519                      current_asteroid->asteroid->line = new_line_number;
520                      current_asteroid->asteroid->asteroidY = (new_line_number * BA_SIZE + BORDERS_SIZE);//!!!
521                      current_asteroid->asteroid->asteroidX = WINDOW_WIDTH - current_asteroid->asteroid->asteroidSize;
522
523                      break;
524                  }
525                  current_asteroid = current_asteroid->next;
526              }
527          current_bullet = current_bullet->next;
528          current_asteroid = asteroids[i]->next;
529      }
530      }
531
532  }
```

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

- **Функция проверки коллизии игрока с астероидами**

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

Циклы проверки:

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

```
534  void check_asteroid_player_collisions(struct Asteroid_list **asteroid_array)
535  {
536      int line_number = (player.playerY - BORDERS_SIZE) / BA_SIZE;
537      struct Asteroid_list *current_asteroid = asteroid_array[line_number]->next;
538      struct Asteroid_list *next_asteroid;
539      while(current_asteroid != NULL) {
540          next_asteroid = current_asteroid->next;
541          if (is_colliding_ap(current_asteroid->asteroid, player) \
542          && is_asteroid_on_screen(current_asteroid->asteroid)) {
543              if (!player.godMode) {
544                  player.playerLives -= 1;
545              }
546              if (!megalovania_is_playing) {
547                  PlaySound("..//..//sounds//hit_player.wav", NULL, SND_FILENAME | SND_ASYNC);
548              }
549              int new_line_number = get_random_number() % LINE_COUNT;
550
551              remove_from_alist(asteroid_array[line_number],current_asteroid->asteroid);
552              add_to_alist(asteroid_array[new_line_number], current_asteroid);
553
554              current_asteroid->asteroid->line = new_line_number;
555              current_asteroid->asteroid->asteroidY = (new_line_number * BA_SIZE + BORDERS_SIZE);//!!
556              current_asteroid->asteroid->asteroidX = WINDOW_WIDTH - current_asteroid->asteroid->asteroidSize;
557          }
558          current_asteroid = next_asteroid;
559      }
560      if (line_number == LINE_COUNT) return; //last line
561
562      current_asteroid = asteroid_array[line_number+1];
563      while(current_asteroid != NULL) {
564          next_asteroid = current_asteroid->next;
565          if (is_colliding_ap(current_asteroid->asteroid, player) \
566          && is_asteroid_on_screen(current_asteroid->asteroid)) {···
584          current_asteroid = next_asteroid;
585      }
586  }
```

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

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

Машинно-независимые оптимизации

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

- **Оптимизация через предварительную генерацию случайных чисел**

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

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

```
 9  void init_random_pool() {
10      srand(time(NULL));
11      random_pool = (int*) malloc(RAND_POOL_SIZE * sizeof(int));
12      for (int i = 0; i < RAND_POOL_SIZE; i++) {
13          random_pool[i] = rand();
14      }
15  }
16
17  int get_random_number() {
18      int num = random_pool[random_index];
19      random_index = (random_index+1) % RAND_POOL_SIZE;
20      return num;
21  }
```

Рис. 15. Генерация случайных чисел.

- **Частота проверки коллизий**

В целях экономии процессорных ресурсов, частота проверки коллизий и обновления состояний у разных объектов была уменьшена.

Для этого, в функцию update(...) было добавлено множество условий, из-за которых частота вызовов функций проверок уменьшается.

```
if (update_count % 2 == 0 ){
    check_bullet_asteroid_collisions(small_asteroids_array);
    check_bullet_asteroid_collisions(medium_asteroids_array);
    check_bullet_asteroid_collisions(big_asteroids_array);

    check_asteroid_player_collisions(small_asteroids_array);
    check_asteroid_player_collisions(medium_asteroids_array);
    check_asteroid_player_collisions(big_asteroids_array);
}
```

Рис. 16. Уменьшение частоты проверки коллизий.

```
if (!player.godMode && update_count % 16 == 0 ){
    update_player_state();
}
```

Рис. 17. Уменьшение частоты обновления состояния игрока.

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

- **Удаление избыточных вызовов**

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

```
45  void draw_main_menu() {
46      glClear(GL_COLOR_BUFFER_BIT);
47
48      glEnable(GL_TEXTURE_2D);
49      //draw main menu
50      glColor3f(1.0, 1.0, 1.0);
51      if (main_menu.option == 1) {
52          draw_rectangle(0, 0, WINDOW_WIDTH, WINDOW_HEIGHT, 11);
53      } else if (main_menu.option == 0) {
54          draw_rectangle(0, 0, WINDOW_WIDTH, WINDOW_HEIGHT, 12);
55      }
56      glDisable(GL_TEXTURE_2D);
57      glutSwapBuffers();
58  }
```

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

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

- **Кеширование текстур**

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

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

```
21  void draw_rectangle(int x, int y, int width, int height, int texture_id) {
22      if (texture_id != last_texture_id) {
23          glBindTexture(GL_TEXTURE_2D, textures[texture_id]);
24          last_texture_id = texture_id;
25      }
```

Рис. 19. Кеширование номера текстуры.

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

- **Условие проверки коллизий**

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

```
if(bullet_count != 0){
    check_bullet_asteroid_collisions(small_asteroids_array);
    check_bullet_asteroid_collisions(medium_asteroids_array);
    check_bullet_asteroid_collisions(big_asteroids_array);
}
```

Рис. 20. Проверка количества пуль.

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

Таким образом происходит экономия вычислительных ресурсов процессора.

Оценка сложности оптимизированного алгоритма

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

Анализ полученных данных для двух версий алгоритмов

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

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

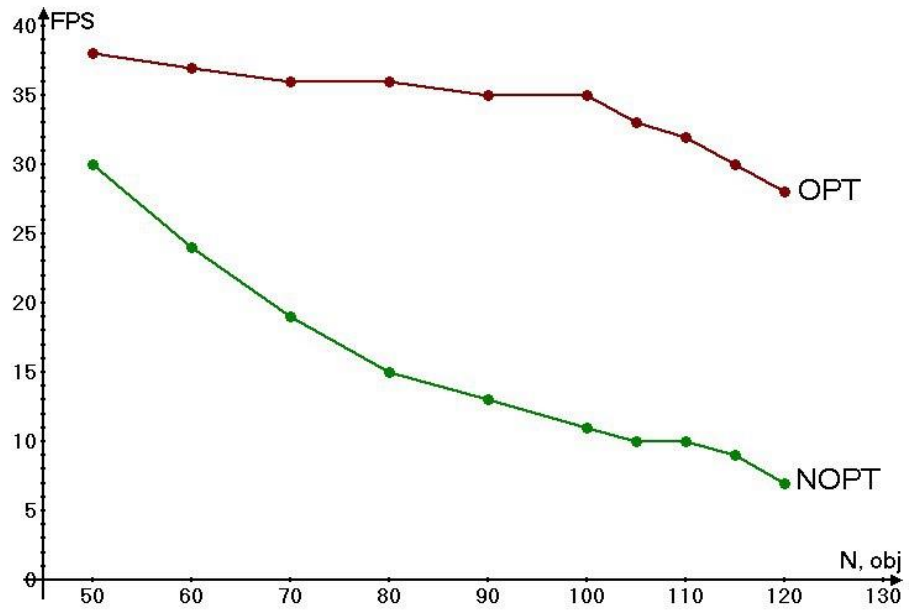

Рис. 21. График зависимости FPS от кол-ва объектов.

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

ЗАКЛЮЧЕНИЕ

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

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

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

ПРИЛОЖЕНИЕ А

Листинг кода - ссылка на репозиторий в GitHub

- Код файла main.c

```
#include <GL/glut.h>

#include <stdlib.h>
#include <stdio.h>
#include <stdbool.h>
#include <unistd.h>
#include <windows.h>
#include <mmsystem.h>

#include "constants.h"
#include "structures.h"
#include "lists.h"
#include "keyboard.h"
#include "draw.h"
#include "init.h"

GLuint textures[TEXTURES_AMT+1] = { 0 };

struct Player player;
struct Boss boss;
struct Heart heart;
struct Menu main_menu;

struct Bullet_list *bullet_array[LINE_COUNT];
struct Bullet_list *boss_bullet_array[LINE_COUNT];
struct Asteroid_list *small_asteroids_array[LINE_COUNT];
struct Asteroid_list *medium_asteroids_array[LINE_COUNT];
struct Asteroid_list *big_asteroids_array[LINE_COUNT];

struct Bullet_list *bullets;
struct Bullet_list *boss_bullets;
struct Asteroid_list *small_asteroids;
struct Asteroid_list *medium_asteroids;
struct Asteroid_list *big_asteroids;

bool changed_to_second = false;
bool changed_to_third = false;
bool changed_to_fourth = false;
bool player_is_dead = false;
bool boss_is_dead = false;
bool megalovania_is_playing = false;
bool spawn_asteroids = true;
bool game_just_started = true;

int boss_delay = 0;

int update_count;
int bullet_count = 0;

int frameCountPerSecond;
int frameCount;
double previousTime;

int *random_pool;
int random_index;

int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitWindowSize(WINDOW_WIDTH, WINDOW_HEIGHT);
    glutInitWindowPosition((glutGet(GLUT_SCREEN_WIDTH)-WINDOW_WIDTH)/2,
                           (glutGet(GLUT_SCREEN_HEIGHT)-WINDOW_HEIGHT)/2);
```

```
            glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE);
            glutCreateWindow(WINDOW_CAPTION);
            glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
            glEnable(GL_BLEND);

            for (int i=0; i < LINE_COUNT; i++){
                bullet_array[i] = init_bullet_list_elem();
                boss_bullet_array[i] = init_bullet_list_elem();

                small_asteroids_array[i] = init_asteroid_list_elem('s');
                small_asteroids_array[i]->asteroid->line = i;

                medium_asteroids_array[i] = init_asteroid_list_elem('m');
                medium_asteroids_array[i]->asteroid->line = i;

                big_asteroids_array[i] = init_asteroid_list_elem('b');
                big_asteroids_array[i]->asteroid->line = i;
            }

            main_menu_init();
            glutDisplayFunc(draw_main_menu);
            glutKeyboardFunc(handle_menu_keyboard);
            glutSpecialFunc(handle_menu_special_keyboard);

            if (!megalovania_is_playing) {
                PlaySound("..//..//sounds//menu.wav",   NULL,   SND_FILENAME   |   SND_ASYNC   |
SND_LOOP);
            }

            glutMainLoop();

            return 0;
        }
```

- Код файла rand.c

```
        #include <stdlib.h>
        #include <time.h>
        #include <GL/glut.h>
        #include <stdio.h>

        #include "constants.h"
        #include "extern_pointers.h"

        void init_random_pool() {
            srand(time(NULL));
            random_pool = (int*) malloc(RAND_POOL_SIZE * sizeof(int));
            for (int i = 0; i < RAND_POOL_SIZE; i++) {
                random_pool[i] = rand();
            }
        }

        int get_random_number() {
            int num = random_pool[random_index];
            random_index = (random_index + 1) % RAND_POOL_SIZE;
            return num;
        }
```

- Код файла update.c

```
        #include <GL/glut.h>
        #include <windows.h>
        #include <mmsystem.h>
        #include <stdbool.h>
        #include <stdio.h>
        #include <time.h>

        #include "constants.h"
        #include "lists.h"
```

```
#include "extern_pointers.h"
#include "rand.h"

bool is_bullet_on_screen(struct Bullet *b) {
    if (b->bulletX >= 0 && b->bulletX <= WINDOW_WIDTH) {
        return true;
    } else {
        return false;
    }
}

bool is_asteroid_on_screen(struct Asteroid *a) {
    if (a->asteroidX >= 0 && a->asteroidX <= WINDOW_WIDTH) {
        return true;
    } else {
        return false;
    }
}

void add_bullet(struct Bullet_list **b_array) {
    bullet_count++;
    int y_coordinate = player.playerY + player.playerSize / 2 - B_SIZE / 2;
    int line_number = (y_coordinate - BORDERS_SIZE) / BA_SIZE;
    bool flag_between = false;
    struct Bullet_list *temp = b_array[line_number];
    struct Bullet *bullet;

    while (true) {
        if (is_bullet_on_screen(temp->bullet) && temp->next == NULL) {
            struct Bullet_list *bl_elem = init_bullet_list_elem();
            temp->next = bl_elem;
        } else if (is_bullet_on_screen(temp->bullet) && temp->next != NULL) {
            temp = temp->next;
        } else if (!is_bullet_on_screen(temp->bullet)) {
            temp->bullet->bulletY = y_coordinate;
            temp->bullet->bulletX = player.playerX;
            bullet = temp->bullet;

            if(((y_coordinate + bullet->bulletSize - BORDERS_SIZE) / BA_SIZE) !=
line_number) {
                flag_between = true;
            }
            bullet->between = flag_between;

            break;
        }
    }

    // bullet between lines
    if (flag_between) {
        temp = b_array[line_number + 1];
        while (true) {
            if (is_bullet_on_screen(temp->bullet) && temp->next == NULL) {
                struct Bullet_list *bl_elem = init_bullet_list_elem();
                temp->next = bl_elem;
            } else if (is_bullet_on_screen(temp->bullet) && temp->next != NULL) {
                temp = temp->next;
            } else if (!is_bullet_on_screen(temp->bullet)) {
                temp->bullet = bullet;

                break;
            }
        }
    }
}

void add_boss_bullet(struct Bullet_list **b_array) {
    int y_coordinate = boss.bossY + boss.bossSize / 2 - B_SIZE / 2;
    int line_number = (y_coordinate - BORDERS_SIZE) / BA_SIZE;
    bool flag_between = false;
```

```
            if(((y_coordinate + B_SIZE - BORDERS_SIZE) / BA_SIZE) != line_number) {
                flag_between = true;
            }

            struct Bullet_list *temp = b_array[line_number];
            struct Bullet *bullet;

            while (true) {
                if (is_bullet_on_screen(temp->bullet) && temp->next == NULL) {
                    struct Bullet_list *bl_elem = init_bullet_list_elem();
                    temp->next = bl_elem;
                } else if (is_bullet_on_screen(temp->bullet) && temp->next != NULL) {
                    temp = temp->next;
                } else if (!is_bullet_on_screen(temp->bullet)) {
                    temp->bullet->bulletY = y_coordinate;
                    temp->bullet->bulletX = boss.bossX;
                    bullet = temp->bullet;
                    bullet->between = flag_between;

                    break;
                }
            }

            // bullet between lines
            if (flag_between) {
                temp = b_array[line_number + 1];
                while (true) {
                    if (is_bullet_on_screen(temp->bullet) && temp->next == NULL) {
                        struct Bullet_list *bl_elem = init_bullet_list_elem();
                        temp->next = bl_elem;
                    } else if (is_bullet_on_screen(temp->bullet) && temp->next != NULL) {
                        temp = temp->next;
                    } else if (!is_bullet_on_screen(temp->bullet)) {
                        temp->bullet = bullet;

                        break;
                    }
                }
            }
        }

        void add_asteroid(struct Asteroid_list **a_array) {
            char asteroid_type = 'o';
            int asteroidsize = -10;
            int line_number;
            int y_coordinate;

            if (a_array == small_asteroids_array) {
                line_number = get_random_number() % LINE_COUNT;
                asteroid_type = 's';
                asteroidsize = SA_SIZE;
                y_coordinate = (BORDERS_SIZE + line_number * BA_SIZE) + get_random_number() %
(BA_SIZE - asteroidsize);
            } else if (a_array == medium_asteroids_array) {
                line_number = get_random_number() % LINE_COUNT;
                asteroid_type = 'm';
                asteroidsize = MA_SIZE;
                y_coordinate = (BORDERS_SIZE + line_number * BA_SIZE) + get_random_number() %
(BA_SIZE - asteroidsize);
            } else if (a_array == big_asteroids_array) {
                line_number = get_random_number() % LINE_COUNT;
                y_coordinate = (BORDERS_SIZE + line_number * BA_SIZE);
                asteroid_type = 'b';
                asteroidsize = BA_SIZE;
            }

            struct Asteroid_list *temp = a_array[line_number];
            while (temp != NULL) {
                if (temp->next == NULL) {
```

```
                struct Asteroid_list *al_elem = init_asteroid_list_elem(asteroid_type);
                temp->next = al_elem;

                temp->next->asteroid->asteroidY = y_coordinate;
                temp->next->asteroid->asteroidX = WINDOW_WIDTH - asteroidsize;
                temp->next->asteroid->line = line_number;

                return;
            }
            temp = temp->next;
        }
    }

    bool is_colliding_ap(struct Asteroid *a, struct Player p) {
        if (
            (a->asteroidY >= p.playerY && a->asteroidY <= p.playerY + p.playerSize \
            && a->asteroidX >= p.playerX && a->asteroidX <= p.playerX + p.playerSize)
            ||
            (a->asteroidY+a->asteroidSize >= p.playerY && a->asteroidY+a->asteroidSize <=
p.playerY + p.playerSize \
            && a->asteroidX+a->asteroidSize >= p.playerX && a->asteroidX+a->asteroidSize <=
p.playerX + p.playerSize)
            )
        {
            return true;
        } else {
            return false;
        }
    }

    bool is_colliding_hp(struct Heart h, struct Player p) {
        if (
            (h.heartY >= p.playerY && h.heartY <= p.playerY + p.playerSize \
            && h.heartX >= p.playerX && h.heartX <= p.playerX + p.playerSize)
            ||
            (h.heartY+h.heartSize >= p.playerY && h.heartY+h.heartSize <= p.playerY +
p.playerSize \
            && h.heartX+h.heartSize >= p.playerX && h.heartX+h.heartSize <= p.playerX +
p.playerSize)
            )
        {
            return true;
        } else {
            return false;
        }
    }

    bool is_colliding_ba(struct Bullet *b, struct Asteroid *a) {
        if (
            (b->bulletY >= a->asteroidY && b->bulletY <= a->asteroidY + a->asteroidSize \
            && b->bulletX >= a->asteroidX && b->bulletX <= a->asteroidX + a->asteroidSize)
            ||
            (b->bulletY+b->bulletSize >= a->asteroidY && b->bulletY+b->bulletSize <= a-
>asteroidY + a->asteroidSize \
            && b->bulletX+b->bulletSize >= a->asteroidX && b->bulletX+b->bulletSize <= a-
>asteroidX + a->asteroidSize)
            )
        {
            return true;
        } else {
            return false;
        }
    }

    bool is_colliding_bp(struct Bullet *b, struct Player p) {
        if (
            (b->bulletY >= p.playerY && b->bulletY <= p.playerY + p.playerSize \
            && b->bulletX >= p.playerX && b->bulletX <= p.playerX + p.playerSize)
            ||
            (b->bulletY+b->bulletSize >= p.playerY && b->bulletY+b->bulletSize <= p.playerY
```

```
+ p.playerSize \
               &&  b->bulletX+b->bulletSize  >=  p.playerX  &&  b->bulletX+b->bulletSize  <=
p.playerX + p.playerSize)
               )
           {
               return true;
           } else {
               return false;
           }
       }

       bool is_colliding_bbs(struct Bullet *b, struct Boss bs) {
           if (
               (b->bulletY >= bs.bossY && b->bulletY <= bs.bossY + bs.bossSize \
               && b->bulletX >= bs.bossX && b->bulletX <= bs.bossX + bs.bossSize)
               ||
               (b->bulletY+b->bulletSize >= bs.bossY && b->bulletY+b->bulletSize <= bs.bossY +
bs.bossSize \
               && b->bulletX+b->bulletSize >= bs.bossX && b->bulletX+b->bulletSize <= bs.bossX
+ bs.bossSize)
               )
           {
               return true;
           } else {
               return false;
           }
       }

       void update_boss_bullet_position(struct Bullet *bullet) {
           if (is_bullet_on_screen(bullet)) {
               bullet->bulletX -= bullet->bulletSpeed;
               if (!is_bullet_on_screen(bullet)) {
                   int line_number = (bullet->bulletY - BORDERS_SIZE) / BA_SIZE;
                   remove_from_blist(boss_bullet_array[line_number], bullet);
                   if (bullet->between) {
                       remove_from_blist(boss_bullet_array[line_number + 1], bullet);
                   }
               }
           }
       }

       void update_bullet_position(struct Bullet *bullet) {
           if (is_bullet_on_screen(bullet)) {
               if (bullet->between) {
                   bullet->bulletX += bullet->bulletSpeed / 2;
               } else {
                   bullet->bulletX += bullet->bulletSpeed;
               }

               if (!is_bullet_on_screen(bullet)) {
                   int line_number = (bullet->bulletY - BORDERS_SIZE) / BA_SIZE;
                   remove_from_blist(bullet_array[line_number], bullet);
                   bullet_count--;
               }
           }
       }

       void update_asteroid_position(struct Asteroid_list **asteroid_array, struct Asteroid
*asteroid) {
           if (is_asteroid_on_screen(asteroid)) {
               asteroid->asteroidX -= asteroid->asteroidSpeed + (3 * (player.currentLevel -
1));
           } else if (!is_asteroid_on_screen(asteroid) && player.playerLives > 0) {
               if (megalovania_is_playing) {
                   asteroid->asteroidX = WINDOW_WIDTH - asteroid->asteroidSize;
                   return;
               }

               int rand = get_random_number();
               int new_line_number = rand % LINE_COUNT;
```

```
            int size_offset = BA_SIZE - asteroid->asteroidSize;
            int size = asteroid->asteroidSize;
            struct Asteroid_list * list_elem;
            list_elem = remove_from_alist(asteroid_array[asteroid->line], asteroid);
            add_to_alist(asteroid_array[new_line_number], list_elem);
            asteroid->asteroidY = (new_line_number * BA_SIZE + BORDERS_SIZE);

            if (asteroid->asteroidSize < BA_SIZE) {
                asteroid->asteroidY += rand % size_offset;
            }

            asteroid->line = new_line_number;
            asteroid->asteroidX = WINDOW_WIDTH - asteroid->asteroidSize;
        }
    }

    void update_player_state() {
        if (game_just_started == true) { //!! move to another place
            add_asteroid(small_asteroids_array);
            add_asteroid(medium_asteroids_array);
            add_asteroid(big_asteroids_array);

            game_just_started = false;
        }

        if (player.playerLives == 0 && player_is_dead == false) {
            if (!megalovania_is_playing) {
                PlaySound("..//..//sounds//lose.wav", NULL, SND_FILENAME | SND_ASYNC);
            }

            player_is_dead = true;
            spawn_asteroids = false;
        }

        if (boss.bossLives == 0 && boss_is_dead == false) {
            if (!megalovania_is_playing) {
                PlaySound("..//..//sounds//win.wav", NULL, SND_FILENAME | SND_ASYNC);
            }

            boss_is_dead = true;
            spawn_asteroids = false;
        }

        if (player.playerScore == FOR_SECOND_LEVEL && changed_to_second == false) {
            if (!megalovania_is_playing) {
                PlaySound("..//..//sounds//next_level.wav",    NULL,    SND_FILENAME    |
SND_ASYNC);
            }

            add_asteroid(small_asteroids_array);
            add_asteroid(medium_asteroids_array);
            add_asteroid(big_asteroids_array);

            player.currentLevel = 2;
            changed_to_second = true;
        }

        if (player.playerScore == FOR_THIRD_LEVEL && changed_to_third == false) {
            if (!megalovania_is_playing) {
                PlaySound("..//..//sounds//next_level.wav",    NULL,    SND_FILENAME    |
SND_ASYNC);
            }

            add_asteroid(small_asteroids_array);
            add_asteroid(medium_asteroids_array);
            add_asteroid(big_asteroids_array);

            player.currentLevel = 3;
            changed_to_third = true;
        }
```

```
            if (player.playerScore == FOR_BOSS_LEVEL && changed_to_fourth == false) {
                if (!megalovania_is_playing) {
                    PlaySound("..//..//sounds//next_level.wav",     NULL,     SND_FILENAME     |
SND_ASYNC);
                }

                player.currentLevel = 4;

                spawn_asteroids = false;
                changed_to_fourth = true;
            }
        }

        void update_boss_state() {
            if (player.currentLevel == 4 && boss_is_dead == false) {
                //boss movement
                if (boss.bossY >= boss.bossSize && boss.reached_top == true) {
                    boss.bossY -= 3;

                    if (boss.bossY < boss.bossSize) {
                        boss.bossY = boss.bossSize;
                        boss.reached_bot = true;
                        boss.reached_top = false;
                    }
                }

                if (boss.bossY <= WINDOW_HEIGHT - boss.bossSize - 40 && boss.reached_bot ==
true) {
                    boss.bossY += 3;

                    if (boss.bossY > WINDOW_HEIGHT - boss.bossSize - 40) {
                        boss.bossY = WINDOW_HEIGHT - boss.bossSize - 40;
                        boss.reached_top = true;
                        boss.reached_bot = false;
                    }
                }

                //boss shooting
                if (boss_delay < BOSS_DELAY_VALUE) {
                    boss_delay++;
                } else if (boss_delay == BOSS_DELAY_VALUE) {
                    add_boss_bullet(boss_bullet_array);
                    boss_delay = 0;
                }

                //update boss_bullet position
                for (int i = 0; i < LINE_COUNT; i++) {
                    for_bullet_list(boss_bullets, update_boss_bullet_position);
                }

                //boss_bullet - player collision
                int line_number = (player.playerY - BORDERS_SIZE) / BA_SIZE;
                struct Bullet_list *check_bsp = boss_bullet_array[line_number];
                while (check_bsp != NULL) {
                    if (is_colliding_bp(check_bsp->bullet, player)) {
                        if (!player.godMode) {
                            player.playerLives -= 1;
                        }

                        if (!megalovania_is_playing) {
                            PlaySound("..//..//sounds//hit_player.wav",  NULL,  SND_FILENAME  |
SND_ASYNC);
                        }

                        remove_from_blist(boss_bullet_array[line_number], check_bsp->bullet);

                        if ((line_number + 1) <= LINE_COUNT) {
                            remove_from_blist(boss_bullet_array[line_number  +  1],  check_bsp-
>bullet);
```

```
                    }

                    break;
                }
                check_bsp = check_bsp->next;
            }

            // between lines
            if ((line_number * BA_SIZE + BORDERS_SIZE + player.playerSize) / BA_SIZE !=
line_number) {
                struct Bullet_list *check_bsp = boss_bullet_array[line_number + 1];
                while (check_bsp != NULL) {
                    if (is_colliding_bp(check_bsp->bullet, player)) {
                        if (!player.godMode) {
                            player.playerLives -= 1;
                        }

                        if (!megalovania_is_playing) {
                            PlaySound("..//..//sounds//hit_player.wav", NULL, SND_FILENAME
| SND_ASYNC);
                        }

                        remove_from_blist(boss_bullet_array[line_number],       check_bsp-
>bullet);

                        if ((line_number + 1) <= LINE_COUNT) {
                            remove_from_blist(boss_bullet_array[line_number     +     1],
check_bsp->bullet);
                        }

                        break;
                    }
                    check_bsp = check_bsp->next;
                }
            }

            //bullet - boss collision
            line_number = (boss.bossY - BORDERS_SIZE) / BA_SIZE;
            struct Bullet_list *check_bbs = bullet_array[line_number];
            while (check_bbs != NULL) {
                if (is_colliding_bbs(check_bbs->bullet, boss)) {
                    boss.bossLives -= 1;
                    if (!megalovania_is_playing) {
                        PlaySound("..//..//sounds//hit_boss.wav",   NULL,   SND_FILENAME   |
SND_ASYNC);
                    }

                    remove_from_blist(bullet_array[line_number], check_bbs->bullet);

                    if ((line_number + 1) <= LINE_COUNT) {
                        remove_from_blist(bullet_array[line_number    +    1],    check_bsp-
>bullet);
                    }

                    break;
                }
                check_bbs = check_bbs->next;
            }

            line_number++;
            check_bbs = bullet_array[line_number];
            while (check_bbs != NULL) {
                if (is_colliding_bbs(check_bbs->bullet, boss)) {
                    boss.bossLives -= 1;
                    if (!megalovania_is_playing) {
                        PlaySound("..//..//sounds//hit_boss.wav",   NULL,   SND_FILENAME   |
SND_ASYNC);
                    }

                    remove_from_blist(bullet_array[line_number], check_bbs->bullet);
```

```
                    if ((line_number + 1) <= LINE_COUNT) {
                        remove_from_blist(bullet_array[line_number    +    1],    check_bsp-
>bullet);
                    }

                    break;
                }
                check_bbs = check_bbs->next;
            }

            if ((line_number+1) <= LINE_COUNT && \
                ((boss.bossY + boss.bossSize - BORDERS_SIZE) / BA_SIZE - \
                (boss.bossY - BORDERS_SIZE) / BA_SIZE ) == 2) {

                line_number++;
                check_bbs = bullet_array[line_number];
                while (check_bbs != NULL) {
                    if (is_colliding_bbs(check_bbs->bullet, boss)) {
                        boss.bossLives -= 1;
                        if (!megalovania_is_playing) {
                            PlaySound("..//..//sounds//hit_boss.wav", NULL, SND_FILENAME |
SND_ASYNC);
                        }

                        remove_from_blist(bullet_array[line_number], check_bbs->bullet);

                        if ((line_number + 1) <= LINE_COUNT) {
                            remove_from_blist(bullet_array[line_number  +  1],  check_bsp-
>bullet);
                        }

                        break;
                    }
                    check_bbs = check_bbs->next;
                }
            }
        }
    }

    void maybe_spawn_heart(struct Asteroid_list *a) {
        int rand_int = get_random_number() % 100;
        if (rand_int > 70 && player.playerLives < 3 && heart.heartX < 0) {
            if (!megalovania_is_playing) {
                PlaySound("..//..//sounds//heart_spawn.wav",    NULL,    SND_FILENAME    |
SND_ASYNC);
            }

            heart.heartX = a->asteroid->asteroidX;
            heart.heartY = a->asteroid->asteroidY;
        }
    }

    void check_bullet_asteroid_collisions(struct Asteroid_list **asteroids) {
        for (int i = 0; i < LINE_COUNT; i++) {
            struct Asteroid_list *current_asteroid = asteroids[i]->next;
            struct Bullet_list *current_bullet = bullet_array[i];
            while (current_bullet != NULL) {
                while (current_asteroid != NULL) {
                    if (is_colliding_ba(current_bullet->bullet, current_asteroid->asteroid)
&&
                        is_bullet_on_screen(current_bullet->bullet) &&
                        is_asteroid_on_screen(current_asteroid->asteroid)) {

                        player.playerScore++;
                        if (!megalovania_is_playing) {
                            PlaySound("..//..//sounds//hit_asteroid.wav",            NULL,
SND_FILENAME | SND_ASYNC);
                            if (asteroids == small_asteroids_array) {
                                maybe_spawn_heart(current_asteroid);
```

```
                            }
                        }

                        remove_from_blist(bullet_array[i], current_bullet->bullet);

                        if ((i + 1) < LINE_COUNT && current_bullet->bullet->between) {
                            remove_from_blist(bullet_array[i + 1], current_bullet->bullet);
                        }
                        bullet_count--;

                        int new_line_number = get_random_number() % LINE_COUNT;

                        remove_from_alist(asteroids[i], current_asteroid->asteroid);
                        add_to_alist(asteroids[new_line_number], current_asteroid);

                        current_asteroid->asteroid->line = new_line_number;
                        current_asteroid->asteroid->asteroidY = (new_line_number * BA_SIZE
+ BORDERS_SIZE);//!!!
                        current_asteroid->asteroid->asteroidX     =     WINDOW_WIDTH     -
current_asteroid->asteroid->asteroidSize;

                        break;
                    }
                    current_asteroid = current_asteroid->next;
                }
                current_bullet = current_bullet->next;
                current_asteroid = asteroids[i]->next;
            }
        }
    }

    void check_asteroid_player_collisions(struct Asteroid_list **asteroid_array) {
        int line_number = (player.playerY - BORDERS_SIZE) / BA_SIZE;
        struct Asteroid_list *current_asteroid = asteroid_array[line_number]->next;
        struct Asteroid_list *next_asteroid;
        while (current_asteroid != NULL) {
            next_asteroid = current_asteroid->next;
            if (is_colliding_ap(current_asteroid->asteroid, player) \
                && is_asteroid_on_screen(current_asteroid->asteroid)) {
                if (!player.godMode) {
                    player.playerLives -= 1;
                }

                if (!megalovania_is_playing) {
                    PlaySound("..//..//sounds//hit_player.wav",   NULL,   SND_FILENAME   |
SND_ASYNC);
                }

                int new_line_number = get_random_number() % LINE_COUNT;

                remove_from_alist(asteroid_array[line_number],           current_asteroid-
>asteroid);
                add_to_alist(asteroid_array[new_line_number], current_asteroid);

                current_asteroid->asteroid->line = new_line_number;
                current_asteroid->asteroid->asteroidY  =  (new_line_number  *  BA_SIZE  +
BORDERS_SIZE);//!!
                current_asteroid->asteroid->asteroidX  =  WINDOW_WIDTH  -  current_asteroid-
>asteroid->asteroidSize;
            }
            current_asteroid = next_asteroid;
        }

        //last line
        if (line_number == LINE_COUNT) {
            return;
        }

        current_asteroid = asteroid_array[line_number + 1];
        while (current_asteroid != NULL) {
```

```
                next_asteroid = current_asteroid->next;
                if (is_colliding_ap(current_asteroid->asteroid, player) \
                    && is_asteroid_on_screen(current_asteroid->asteroid)) {
                    if (!player.godMode) {
                        player.playerLives -= 1;
                    }

                    if (!megalovania_is_playing) {
                        PlaySound("..//..//sounds//hit_player.wav",   NULL,   SND_FILENAME   |
SND_ASYNC);
                    }

                    int new_line_number = get_random_number() % LINE_COUNT;

                    remove_from_alist(asteroid_array[line_number     +     1],current_asteroid-
>asteroid);
                    add_to_alist(asteroid_array[new_line_number], current_asteroid);

                    current_asteroid->asteroid->line = new_line_number;
                    current_asteroid->asteroid->asteroidY  =  (new_line_number  *  BA_SIZE  +
BORDERS_SIZE);//!!
                    current_asteroid->asteroid->asteroidX  =  WINDOW_WIDTH  -  current_asteroid-
>asteroid->asteroidSize;
                }
                current_asteroid = next_asteroid;
            }
        }

        void update(int aux) {
            if (megalovania_is_playing) {
                add_bullet(bullet_array);
            }

            if (!player.godMode && update_count % 16 == 0) {
                update_player_state();
            }

            if (!player.godMode && player.currentLevel == 4) {
                //TODO
                //update_boss_state();
            }

            //update player bullet position
            for(int i = 0; i < LINE_COUNT; i++) {
                for_bullet_list(bullet_array[i], update_bullet_position);
            }

            if (!player.godMode) {
                //update heart position
                if (heart.heartX >= 0) {
                    heart.heartX -= heart.heartSpeed;
                    if (heart.heartX <= 0) {
                        heart.heartY = -heart.heartSize;
                        heart.heartX = -heart.heartSize;
                    }
                }

                //check heart-player collision
                if (update_count % 4 == 0 && is_colliding_hp(heart, player)) {
                    player.playerLives += 1;
                    if (!megalovania_is_playing) {
                        PlaySound("..//..//sounds//heal.wav", NULL, SND_FILENAME | SND_ASYNC);
                    }

                    heart.heartY = -heart.heartSize;
                    heart.heartX = -heart.heartSize;
                }
            }

            // collisions check
```

```
            if (spawn_asteroids == true && player.currentLevel != 4) {
                for(int i = 0; i < LINE_COUNT; i++) {
                    for_asteroid_list(small_asteroids_array,          small_asteroids_array[i],
update_asteroid_position);
                    for_asteroid_list(medium_asteroids_array,        medium_asteroids_array[i],
update_asteroid_position);
                    for_asteroid_list(big_asteroids_array,             big_asteroids_array[i],
update_asteroid_position);
                }

                if (update_count % 2 == 0 ){
                    if(bullet_count != 0){
                        check_bullet_asteroid_collisions(small_asteroids_array);
                        check_bullet_asteroid_collisions(medium_asteroids_array);
                        check_bullet_asteroid_collisions(big_asteroids_array);
                    }
                    check_asteroid_player_collisions(small_asteroids_array);
                    check_asteroid_player_collisions(medium_asteroids_array);
                    check_asteroid_player_collisions(big_asteroids_array);
                }
            }

            update_count++;

            glutPostRedisplay();
            glutTimerFunc(25, update, 0);
        }
```

- Код файла lists.c

```
        #include <stdlib.h>
        #include <stdio.h>

        #include "structures.h"
        #include "constants.h"

        struct Asteroid_list* init_asteroid_list_elem(char type) {
            struct    Asteroid_list    *asteroid_list_elem    =    (struct    Asteroid_list
*)calloc(1 ,sizeof(struct Asteroid_list));
            struct Asteroid *asteroid = (struct Asteroid *)calloc(1 ,sizeof(struct Asteroid));

            switch (type) {
            case 's':
                asteroid->asteroidSize = SA_SIZE;
                asteroid->asteroidSpeed = SA_SPEED;
                asteroid->asteroidY = -asteroid->asteroidSize;
                asteroid->asteroidX = WINDOW_WIDTH - asteroid->asteroidSize;
                asteroid->line = -1;
                break;
            case 'm':
                asteroid->asteroidSize = MA_SIZE;
                asteroid->asteroidSpeed = MA_SPEED;
                asteroid->asteroidY = -asteroid->asteroidSize;
                asteroid->asteroidX = WINDOW_WIDTH - asteroid->asteroidSize;
                asteroid->line = -1;
                break;
            case 'b':
                asteroid->asteroidSize = BA_SIZE;
                asteroid->asteroidSpeed = BA_SPEED;
                asteroid->asteroidY = -asteroid->asteroidSize;
                asteroid->asteroidX = WINDOW_WIDTH - asteroid->asteroidSize;
                asteroid->line = -1;
                break;
            default:
                break;
            }

            asteroid_list_elem->asteroid = asteroid;
            asteroid_list_elem->next = NULL;
```

```
            return asteroid_list_elem;
        }

        struct Bullet_list * init_bullet_list_elem() {
            struct Bullet_list *bullet_list_elem = (struct Bullet_list *)calloc(1 ,sizeof(struct
Bullet_list));
            struct Bullet *bullet = (struct Bullet *)calloc(1 ,sizeof(struct Bullet));

            bullet->bulletSize = B_SIZE;
            bullet->bulletSpeed = B_SPEED;
            bullet->bulletY = -bullet->bulletSize;
            bullet->bulletX = -bullet->bulletSize;

            bullet_list_elem->bullet = bullet;
            bullet_list_elem->next = NULL;

            return bullet_list_elem;
        }


        void add_to_alist(struct Asteroid_list *list_head, struct Asteroid_list *a){
            a->next = list_head->next;
            list_head->next = a;
        }

        void add_to_blist(struct Bullet_list *list_head, struct Bullet_list *b){
            b->next = list_head->next;
            list_head->next = b;
        }

        struct  Asteroid_list  *  remove_from_alist(struct  Asteroid_list  *list_head,  struct
Asteroid *a) {
            if (list_head->asteroid == a) {
                list_head->asteroid->asteroidX = -a->asteroidSize;
                list_head->asteroid->asteroidY = -a->asteroidSize;

                return list_head;
            }

            struct Asteroid_list *temp = list_head->next;
            struct Asteroid_list *prev = list_head;
            while(temp != NULL)
            {
                if (temp->asteroid == a){
                    prev->next = temp->next;

                    return temp;
                }
                prev = temp;
                temp = temp->next;
            }
            return NULL;
        }

        struct Bullet_list * remove_from_blist(struct Bullet_list *list_head, struct Bullet *b)
{
            if (list_head->bullet == b) {
                list_head->bullet->bulletX = -b->bulletSize;
                list_head->bullet->bulletY = -b->bulletSize;

                return list_head;
            }

            struct Bullet_list *temp = list_head->next;
            struct Bullet_list *prev = list_head;
            while(temp != NULL)
            {
                if (temp->bullet == b){
                    prev->next = temp->next;
                    return temp;
```

```
            }
            prev = temp;
            temp = temp->next;
        }
    }

    void for_asteroid_list(struct Asteroid_list **array, struct Asteroid_list *list_head,
void (*func)(struct Asteroid_list **list_head, struct Asteroid *asteroid)) {
        struct Asteroid_list *temp = list_head->next;
        while(temp != NULL)
        {
            func(array, temp->asteroid);
            temp = temp->next;
        }
    }

    void for_bullet_list(struct Bullet_list *list_head, void (*func)(struct Bullet*)) {
        struct Bullet_list *temp = list_head;
        while(temp != NULL)
        {
            func(temp->bullet);
            temp = temp->next;
        }
    }
```

- Код файла init.c

```
    #include <GL/glut.h>
    #include <stdio.h>

    #define STB_IMAGE_IMPLEMENTATION
    #include <stb_image.h>

    #include "constants.h"
    #include "lists.h"
    #include "extern_pointers.h"
    #include "rand.h"

    void main_menu_init() {
        glClearColor(1.0, 1.0, 1.0, 1.0);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluOrtho2D(0, WINDOW_WIDTH, 0, WINDOW_HEIGHT);

        //?player -> bullet -> boss_bullet -> s_asteroid -> m_asteroid -> b_asteroid -> boss
-> background -> main_menu_start -> main_menu_exit
        glGenTextures(TEXTURES_AMT, textures);

        //!main_menu_start - start
        glBindTexture(GL_TEXTURE_2D, textures[11]);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
        int menus_width = 0, menus_height = 0, menus_channels = 0;
        unsigned   char   *menus_texture_img   =   stbi_load(MENUS_FILENAME,   &menus_width,
&menus_height, &menus_channels, 0);
        if(menus_texture_img == NULL) {
            printf("error in loading boss_texture_img\n");
            // exit(1);
        }
        printf("%s - %dx%d with %d channels\n", MENUS_FILENAME, menus_width, menus_height,
menus_channels);
        if (menus_channels == 3) {
            glTexImage2D(GL_TEXTURE_2D,  0,  GL_RGB,  menus_width,  menus_height,  0,  GL_RGB,
GL_UNSIGNED_BYTE, menus_texture_img);
        } else if (menus_channels == 4) {
            glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, menus_width, menus_height, 0, GL_RGBA,
GL_UNSIGNED_BYTE, menus_texture_img);
        }
```

```
            stbi_image_free(menus_texture_img);
            glBindTexture(GL_TEXTURE_2D, 0);
            //!main_menu_start - end

            //!main_menu_exit - start
            glBindTexture(GL_TEXTURE_2D, textures[12]);
            glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
            glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
            glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
            glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
            int menue_width = 0, menue_height = 0, menue_channels = 0;
            unsigned char *menue_texture_img = stbi_load(MENUE_FILENAME, &menue_width,
&menue_height, &menue_channels, 0);
            if(menue_texture_img == NULL) {
                printf("error in loading boss_texture_img\n");
                // exit(1);
            }
            printf("%s - %dx%d with %d channels\n", MENUE_FILENAME, menue_width, menue_height,
menue_channels);
            if (menue_channels == 3) {
                glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, menue_width, menue_height, 0, GL_RGB,
GL_UNSIGNED_BYTE, menue_texture_img);
            } else if (menue_channels == 4) {
                glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, menue_width, menue_height, 0, GL_RGBA,
GL_UNSIGNED_BYTE, menue_texture_img);
            }
            stbi_image_free(menue_texture_img);
            glBindTexture(GL_TEXTURE_2D, 0);
            //!main_menu_exit - end

            //!objects initialization
            main_menu.option = 1;
        }

        void init_game() {
            //!player texture - start
            glBindTexture(GL_TEXTURE_2D, textures[0]);
            glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
            glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
            glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
            glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

            int pt_width = 0, pt_height = 0, pt_channels = 0;
            unsigned char *player_texture_img = stbi_load(PT_FILENAME, &pt_width, &pt_height,
&pt_channels, 0);
            if(player_texture_img == NULL) {
                printf("error in loading player_texture_img\n");
                // exit(1);
            }
            printf("%s - %dx%d with %d channels\n", PT_FILENAME, pt_width, pt_height,
pt_channels);
            if (pt_channels == 3) {
                glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, pt_width, pt_height, 0, GL_RGB,
GL_UNSIGNED_BYTE, player_texture_img);
            } else if (pt_channels == 4) {
                glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, pt_width, pt_height, 0, GL_RGBA,
GL_UNSIGNED_BYTE, player_texture_img);
            }
            stbi_image_free(player_texture_img);
            glBindTexture(GL_TEXTURE_2D, 0);
            //!player texture - end

            //!bullet texture - start
            glBindTexture(GL_TEXTURE_2D, textures[1]);
            glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
            glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
            glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
            glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

            int bt_width = 0, bt_height = 0, bt_channels = 0;
```

```
            unsigned char *bullet_texture_img = stbi_load(BT_FILENAME, &bt_width, &bt_height,
&bt_channels, 0);
            if(bullet_texture_img == NULL) {
                printf("error in loading bullet_texture_img\n");
                // exit(1);
            }
            printf("%s  -  %dx%d  with  %d  channels\n",  BT_FILENAME,  bt_width,  bt_height,
bt_channels);
            if (bt_channels == 3) {
                glTexImage2D(GL_TEXTURE_2D,  0,  GL_RGB,  bt_width,  bt_height,  0,  GL_RGB,
GL_UNSIGNED_BYTE, bullet_texture_img);
            } else if (bt_channels == 4) {
                glTexImage2D(GL_TEXTURE_2D,  0,  GL_RGBA,  bt_width,  bt_height,  0,  GL_RGBA,
GL_UNSIGNED_BYTE, bullet_texture_img);
            }
            stbi_image_free(bullet_texture_img);
            glBindTexture(GL_TEXTURE_2D, 0);
            //!bullet texture - end

            //!boss_bullet texture - start
            glBindTexture(GL_TEXTURE_2D, textures[2]);
            glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
            glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
            glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
            glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

            int bbt_width = 0, bbt_height = 0, bbt_channels = 0;
            unsigned  char  *boss_bullet_texture_img  =  stbi_load(BBT_FILENAME,  &bbt_width,
&bbt_height, &bbt_channels, 0);
            if(boss_bullet_texture_img == NULL) {
                printf("error in loading boss_bullet_texture_img\n");
                // exit(1);
            }
            printf("%s  -  %dx%d  with  %d  channels\n",  BBT_FILENAME,  bbt_width,  bbt_height,
bbt_channels);
            if (bbt_channels == 3) {
                glTexImage2D(GL_TEXTURE_2D,  0,  GL_RGB,  bbt_width,  bbt_height,  0,  GL_RGB,
GL_UNSIGNED_BYTE, boss_bullet_texture_img);
            } else if (bbt_channels == 4) {
                glTexImage2D(GL_TEXTURE_2D,  0,  GL_RGBA,  bbt_width,  bbt_height,  0,  GL_RGBA,
GL_UNSIGNED_BYTE, boss_bullet_texture_img);
            }
            stbi_image_free(boss_bullet_texture_img);
            glBindTexture(GL_TEXTURE_2D, 0);
            //!boss_bullet texture - end

            //!small_asteroid_texture - start
            glBindTexture(GL_TEXTURE_2D, textures[3]);
            glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
            glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
            glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
            glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
            int sat_width = 0, sat_height = 0, sat_channels = 0;
            unsigned  char  *small_asteroid_texture_img  =  stbi_load(SAT_FILENAME,  &sat_width,
&sat_height, &sat_channels, 0);
            if(small_asteroid_texture_img == NULL) {
                printf("error in loading small_asteroid_texture_img\n");
                // exit(1);
            }
            printf("%s  -  %dx%d  with  %d  channels\n",  SAT_FILENAME,  sat_width,  sat_height,
sat_channels);
            if (sat_channels == 3) {
                glTexImage2D(GL_TEXTURE_2D,  0,  GL_RGB,  sat_width,  sat_height,  0,  GL_RGB,
GL_UNSIGNED_BYTE, small_asteroid_texture_img);
            } else if (sat_channels == 4) {
                glTexImage2D(GL_TEXTURE_2D,  0,  GL_RGBA,  sat_width,  sat_height,  0,  GL_RGBA,
GL_UNSIGNED_BYTE, small_asteroid_texture_img);
            }
            stbi_image_free(small_asteroid_texture_img);
            glBindTexture(GL_TEXTURE_2D, 0);
```

```
            //!small_asteroid_texture - end

            //!medium_asteroid_texture - start
            glBindTexture(GL_TEXTURE_2D, textures[4]);
            glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
            glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
            glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
            glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
            int mat_width = 0, mat_height = 0, mat_channels = 0;
            unsigned char *medium_asteroid_texture_img = stbi_load(MAT_FILENAME, &mat_width,
&mat_height, &mat_channels, 0);
            if(medium_asteroid_texture_img == NULL) {
                printf("error in loading medium_asteroid_texture_img\n");
                // exit(1);
            }
            printf("%s - %dx%d with %d channels\n", MAT_FILENAME, mat_width, mat_height,
mat_channels);
            if (mat_channels == 3) {
                glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, mat_width, mat_height, 0, GL_RGB,
GL_UNSIGNED_BYTE, medium_asteroid_texture_img);
            } else if (mat_channels == 4) {
                glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, mat_width, mat_height, 0, GL_RGBA,
GL_UNSIGNED_BYTE, medium_asteroid_texture_img);
            }
            stbi_image_free(medium_asteroid_texture_img);
            glBindTexture(GL_TEXTURE_2D, 0);
            //!medium_asteroid_texture - start

            //!big_asteroid_texture - start
            glBindTexture(GL_TEXTURE_2D, textures[5]);
            glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
            glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
            glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
            glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
            int bat_width = 0, bat_height = 0, bat_channels = 0;
            unsigned char *big_asteroid_texture_img = stbi_load(BAT_FILENAME, &bat_width,
&bat_height, &bat_channels, 0);
            if(big_asteroid_texture_img == NULL) {
                printf("error in loading big_asteroid_texture_img\n");
                // exit(1);
            }
            printf("%s - %dx%d with %d channels\n", BAT_FILENAME, bat_width, bat_height,
bat_channels);
            if (bat_channels == 3) {
                glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, bat_width, bat_height, 0, GL_RGB,
GL_UNSIGNED_BYTE, big_asteroid_texture_img);
            } else if (bat_channels == 4) {
                glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, bat_width, bat_height, 0, GL_RGBA,
GL_UNSIGNED_BYTE, big_asteroid_texture_img);
            }
            stbi_image_free(big_asteroid_texture_img);
            glBindTexture(GL_TEXTURE_2D, 0);
            //!big_asteroid_texture - end

            //!boss_texture - start
            glBindTexture(GL_TEXTURE_2D, textures[6]);
            glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
            glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
            glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
            glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
            int bosst_width = 0, bosst_height = 0, bosst_channels = 0;
            unsigned char *boss_texture_img = stbi_load(BOSST_FILENAME, &bosst_width,
&bosst_height, &bosst_channels, 0);
            if(boss_texture_img == NULL) {
                printf("error in loading boss_texture_img\n");
                // exit(1);
            }
            printf("%s - %dx%d with %d channels\n", BOSST_FILENAME, bosst_width, bosst_height,
bosst_channels);
            if (bosst_channels == 3) {
```

```
            glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, bosst_width, bosst_height, 0, GL_RGB,
GL_UNSIGNED_BYTE, boss_texture_img);
        } else if (bosst_channels == 4) {
            glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, bosst_width, bosst_height, 0, GL_RGBA,
GL_UNSIGNED_BYTE, boss_texture_img);
        }
        stbi_image_free(boss_texture_img);
        glBindTexture(GL_TEXTURE_2D, 0);
        //!boss_texture - end

        //!backgorund_1heart - start
        glBindTexture(GL_TEXTURE_2D, textures[7]);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
        int bgr1_width = 0, bgr1_height = 0, bgr1_channels = 0;
        unsigned   char   *bgr1_texture_img   =   stbi_load(BGR1_FILENAME,   &bgr1_width,
&bgr1_height, &bgr1_channels, 0);
        if(bgr1_texture_img == NULL) {
            printf("error in loading bgr1_texture_img\n");
            // exit(1);
        }
        printf("%s - %dx%d with %d channels\n", BGR1_FILENAME, bgr1_width, bgr1_height,
bgr1_channels);
        if (bgr1_channels == 3) {
            glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, bgr1_width, bgr1_height, 0, GL_RGB,
GL_UNSIGNED_BYTE, bgr1_texture_img);
        } else if (bgr1_channels == 4) {
            glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, bgr1_width, bgr1_height, 0, GL_RGBA,
GL_UNSIGNED_BYTE, bgr1_texture_img);
        }
        stbi_image_free(bgr1_texture_img);
        glBindTexture(GL_TEXTURE_2D, 0);
        //!background_1heart - end

        //!backgorund_2heart - start
        glBindTexture(GL_TEXTURE_2D, textures[8]);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
        int bgr2_width = 0, bgr2_height = 0, bgr2_channels = 0;
        unsigned   char   *bgr2_texture_img   =   stbi_load(BGR2_FILENAME,   &bgr2_width,
&bgr2_height, &bgr2_channels, 0);
        if(bgr2_texture_img == NULL) {
            printf("error in loading bgr2_texture_img\n");
            // exit(1);
        }
        printf("%s - %dx%d with %d channels\n", BGR1_FILENAME, bgr2_width, bgr2_height,
bgr2_channels);
        if (bgr2_channels == 3) {
            glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, bgr2_width, bgr2_height, 0, GL_RGB,
GL_UNSIGNED_BYTE, bgr2_texture_img);
        } else if (bgr2_channels == 4) {
            glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, bgr2_width, bgr2_height, 0, GL_RGBA,
GL_UNSIGNED_BYTE, bgr2_texture_img);
        }
        stbi_image_free(bgr2_texture_img);
        glBindTexture(GL_TEXTURE_2D, 0);
        //!background_2heart - end

        //!backgorund_3heart - start
        glBindTexture(GL_TEXTURE_2D, textures[9]);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
        int bgr3_width = 0, bgr3_height = 0, bgr3_channels = 0;
        unsigned   char   *bgr3_texture_img   =   stbi_load(BGR3_FILENAME,   &bgr3_width,
```

```
&bgr3_height, &bgr3_channels, 0);
            if(bgr3_texture_img == NULL) {
                printf("error in loading bgr3_texture_img\n");
                // exit(1);
            }
            printf("%s - %dx%d with %d channels\n", BGR3_FILENAME, bgr3_width, bgr3_height,
bgr3_channels);
            if (bgr3_channels == 3) {
                glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, bgr3_width, bgr3_height, 0, GL_RGB,
GL_UNSIGNED_BYTE, bgr3_texture_img);
            } else if (bgr3_channels == 4) {
                glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, bgr3_width, bgr3_height, 0, GL_RGBA,
GL_UNSIGNED_BYTE, bgr3_texture_img);
            }
            stbi_image_free(bgr3_texture_img);
            glBindTexture(GL_TEXTURE_2D, 0);
            //!background_3heart - end

            //!backgorund_0heart - start
            glBindTexture(GL_TEXTURE_2D, textures[10]);
            glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
            glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
            glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
            glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
            int bgr0_width = 0, bgr0_height = 0, bgr0_channels = 0;
            unsigned char *bgr0_texture_img = stbi_load(BGR0_FILENAME, &bgr0_width,
&bgr0_height, &bgr0_channels, 0);
            if(bgr0_texture_img == NULL) {
                printf("error in loading bgr0_texture_img\n");
                // exit(1);
            }
            printf("%s - %dx%d with %d channels\n", BGR0_FILENAME, bgr0_width, bgr0_height,
bgr0_channels);
            if (bgr0_channels == 3) {
                glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, bgr0_width, bgr0_height, 0, GL_RGB,
GL_UNSIGNED_BYTE, bgr0_texture_img);
            } else if (bgr0_channels == 4) {
                glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, bgr0_width, bgr0_height, 0, GL_RGBA,
GL_UNSIGNED_BYTE, bgr0_texture_img);
            }
            stbi_image_free(bgr0_texture_img);
            glBindTexture(GL_TEXTURE_2D, 0);
            //!background_0heart - end

            //!heart_texture - start
            glBindTexture(GL_TEXTURE_2D, textures[13]);
            glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
            glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
            glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
            glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
            int heart_width = 0, heart_height = 0, heart_channels = 0;
            unsigned char *heart_texture_img = stbi_load(HEART_FILENAME, &heart_width,
&heart_height, &heart_channels, 0);
            if(heart_texture_img == NULL) {
                printf("error in loading heart_texture_img\n");
                // exit(1);
            }
            printf("%s - %dx%d with %d channels\n", HEART_FILENAME, heart_width, heart_height,
heart_channels);
            if (heart_channels == 3) {
                glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, heart_width, heart_height, 0, GL_RGB,
GL_UNSIGNED_BYTE, heart_texture_img);
            } else if (heart_channels == 4) {
                glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, heart_width, heart_height, 0, GL_RGBA,
GL_UNSIGNED_BYTE, heart_texture_img);
            }
            stbi_image_free(heart_texture_img);
            glBindTexture(GL_TEXTURE_2D, 0);
            //!heart_texture - end
```

```
    //!objects initialization
    player.currentLevel = PLAYER_START_LEVEL;
    player.playerSize = PLAYER_SIZE;
    player.playerY = WINDOW_HEIGHT / 2;
    player.playerX = player.playerSize;
    player.playerScore = PLAYER_SCORE;
    player.playerLives = PLAYER_LIVES;
    player.godMode = false;

    boss.bossLives = BOSS_LIVES;
    boss.bossSize = BOSS_SIZE;
    boss.bossY = WINDOW_HEIGHT / 2;
    boss.bossX = WINDOW_WIDTH - boss.bossSize * 2;

    heart.heartSize = HEART_SIZE;
    heart.spawn = true;
    heart.heartX = WINDOW_WIDTH - heart.heartSize;
    heart.heartY = -heart.heartSize;
    heart.heartSpeed = HEART_SPEED;

    boss.reached_top = true;
    boss.reached_bot = false;

    init_random_pool();
}
```

- Код файла keyboard.c

```
#include <GL/glut.h>
#include <unistd.h>
#include <stdio.h>
#include <windows.h>
#include <mmsystem.h>

#include "constants.h"
#include "extern_pointers.h"
#include "update.h"
#include "draw.h"
#include "init.h"

void handle_keyboard(unsigned char key, int x, int y) {
    switch (key) {
        case 32: //SPACE
            if (player.playerLives > 0) {
                add_bullet(bullet_array);
                if (!megalovania_is_playing) {
                    PlaySound("..//..//sounds//shot.wav",    NULL,    SND_FILENAME    |
SND_ASYNC);
                }
            }
            break;
        case 27: //ESC
            PlaySound("..//..//sounds//exit.wav", NULL, SND_FILENAME | SND_ASYNC);
            sleep(1);
            exit(0);
            break;
        case 97: //a
            megalovania_is_playing = true;
            player.godMode = true;
            player.playerScore = -9999;
            PlaySound("..//..//sounds//megalovania.wav",    NULL,    SND_FILENAME    |
SND_ASYNC);

            for (int amt = 0; amt < MAX_ASTEROIDS_IN_BENCH_MODE/3; amt++) {
                add_asteroid(small_asteroids_array);
                add_asteroid(medium_asteroids_array);
                add_asteroid(big_asteroids_array);
            }

            break;
```

```
            case 105: //i
                //TODO infinity mode
                break;
        }
    }

    void handle_movement_keys(int key, int x, int y) {
        switch (key) {
            case GLUT_KEY_DOWN:
                player.playerY -= PLAYER_MOVE_STEP;
                if (player.playerY < BORDERS_SIZE) {
                    player.playerY = BORDERS_SIZE;
                }
                break;
            case GLUT_KEY_UP:
                player.playerY += PLAYER_MOVE_STEP;
                if (player.playerY + player.playerSize + BORDERS_SIZE > WINDOW_HEIGHT) {
                    player.playerY = WINDOW_HEIGHT - player.playerSize - BORDERS_SIZE;
                }
                break;
            case GLUT_KEY_RIGHT:
                player.playerX += PLAYER_MOVE_STEP;
                if (player.playerX > player.playerSize + PLAYER_MOVEMENTS_WIDTH) {
                    player.playerX = player.playerSize + PLAYER_MOVEMENTS_WIDTH;
                }
                break;
            case GLUT_KEY_LEFT:
                player.playerX -= PLAYER_MOVE_STEP;
                if (player.playerX < player.playerSize) {
                    player.playerX = player.playerSize;
                }
                break;
        }
    }

    void handle_menu_keyboard(unsigned char key, int x, int y) {
        switch (key) {
            case 27: //ESC
                PlaySound("..//..//sounds//exit.wav", NULL, SND_FILENAME | SND_ASYNC);
                sleep(1);
                exit(0);
                break;
            case 13: //ENTER
                if (main_menu.option == 1) {
                    printf("Player started the game, initializing rest of things...\n");
                    if (!megalovania_is_playing) {
                        PlaySound("..//..//sounds//start.wav",   NULL,   SND_FILENAME   |
SND_ASYNC);
                    }

                    init_game();

                    glutDisplayFunc(draw_scene);
                    glutKeyboardFunc(handle_keyboard);
                    glutSpecialFunc(handle_movement_keys);
                    glutTimerFunc(25, update, 0);
                    break;
                } else if (main_menu.option == 0) {
                    PlaySound("..//..//sounds//exit.wav", NULL, SND_FILENAME | SND_ASYNC);
                    sleep(1);
                    exit(0);
                    break;
                }
        }
    }

    void handle_menu_special_keyboard(int key, int x, int y) {
        switch (key) {
            case GLUT_KEY_DOWN:
                // PlaySound("..//..//sounds//select.wav", NULL, SND_FILENAME | SND_ASYNC);
```

```
            main_menu.option = 0;
            break;
        case GLUT_KEY_UP:
            // PlaySound("..//..//sounds//select.wav", NULL, SND_FILENAME | SND_ASYNC);
            main_menu.option = 1;
            break;
    }

    glutPostRedisplay();
}
```

- Код файла draw.c

```
#include <GL/glut.h>
#include <stdio.h>

#include "constants.h"
#include "extern_pointers.h"
#include "update.h"

static int last_texture_id = -1;

void frame_counter () {
    frameCount++;
    double currentTime = (double)glutGet(GLUT_ELAPSED_TIME) / 1000.0;
    if (currentTime - previousTime >= 1.0) {
        frameCountPerSecond = frameCount;
        frameCount = 0;
        previousTime = currentTime;
    }
}

void draw_rectangle(int x, int y, int width, int height, int texture_id) {
    if (texture_id != last_texture_id) {
        glBindTexture(GL_TEXTURE_2D, textures[texture_id]);
        last_texture_id = texture_id;
    }

    glBegin(GL_QUADS);
    glTexCoord2f(0.0f, 1.0f); glVertex2f(x, y);
    glTexCoord2f(1.0f, 1.0f); glVertex2f(x + width, y);
    glTexCoord2f(1.0f, 0.0f); glVertex2f(x + width, y + height);
    glTexCoord2f(0.0f, 0.0f); glVertex2f(x, y + height);
    glEnd();
}

void draw_text(int x, int y, char* string) {
    glColor3f(1.0, 1.0, 1.0);
    glRasterPos2f(x, y);
    for (int i = 0; string[i] != '\0'; ++i) {
        glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, string[i]);
    }
}

void draw_main_menu() {
    glClear(GL_COLOR_BUFFER_BIT);
    glEnable(GL_TEXTURE_2D);

    //draw main menu
    glColor3f(1.0, 1.0, 1.0);
    if (main_menu.option == 1) {
        draw_rectangle(0, 0, WINDOW_WIDTH, WINDOW_HEIGHT, 11);
    } else if (main_menu.option == 0) {
        draw_rectangle(0, 0, WINDOW_WIDTH, WINDOW_HEIGHT, 12);
    }

    glDisable(GL_TEXTURE_2D);
    glutSwapBuffers();
}
```

```
        void draw_scene() {
            glClear(GL_COLOR_BUFFER_BIT);
            glEnable(GL_TEXTURE_2D);

            //draw background
            if (player.playerLives == 3) {
                draw_rectangle(0, 0, WINDOW_WIDTH, WINDOW_HEIGHT, 9);
            } else if (player.playerLives == 2) {
                draw_rectangle(0, 0, WINDOW_WIDTH, WINDOW_HEIGHT, 8);
            } else if (player.playerLives == 1) {
                draw_rectangle(0, 0, WINDOW_WIDTH, WINDOW_HEIGHT, 7);
            }

            glDisable(GL_TEXTURE_2D);

            if (player.playerLives == 0) {

                //draw background
                glEnable(GL_TEXTURE_2D);
                draw_rectangle(0, 0, WINDOW_WIDTH, WINDOW_HEIGHT, 10);
                glDisable(GL_TEXTURE_2D);

                char game_over[DRAW_TEXT_LENGTH];
                sprintf(game_over, "Game over!");
                draw_text((WINDOW_WIDTH / 2) - 50, WINDOW_HEIGHT / 2, game_over);

                char score[DRAW_TEXT_LENGTH];
                sprintf(score, "Score: %d", player.playerScore);
                draw_text((WINDOW_WIDTH / 2) - 45, WINDOW_HEIGHT / 2 - 40, score);

                char level[DRAW_TEXT_LENGTH];
                sprintf(level, "Level: %d", player.currentLevel);
                draw_text((WINDOW_WIDTH / 2) - 40, WINDOW_HEIGHT / 2 - 60, level);

                char closing[DRAW_TEXT_LENGTH];
                sprintf(closing, "To close game press ESC");
                draw_text((WINDOW_WIDTH / 2) - 120, WINDOW_HEIGHT / 2 - 80, closing);

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

                //draw background
                glEnable(GL_TEXTURE_2D);
                draw_rectangle(0, 0, WINDOW_WIDTH, WINDOW_HEIGHT, 10);
                glDisable(GL_TEXTURE_2D);

                char game_end[DRAW_TEXT_LENGTH];
                sprintf(game_end, "You won!");
                draw_text((WINDOW_WIDTH / 2) - 50, WINDOW_HEIGHT / 2, game_end);

                char score[DRAW_TEXT_LENGTH];
                sprintf(score, "Score: %d", player.playerScore);
                draw_text((WINDOW_WIDTH / 2) - 45, WINDOW_HEIGHT / 2 - 40, score);

                char level[DRAW_TEXT_LENGTH];
                sprintf(level, "Level: %d", player.currentLevel);
                draw_text((WINDOW_WIDTH / 2) - 40, WINDOW_HEIGHT / 2 - 60, level);

                char closing[DRAW_TEXT_LENGTH];
                sprintf(closing, "To close game press ESC");
                draw_text((WINDOW_WIDTH / 2) - 120, WINDOW_HEIGHT / 2 - 80, closing);

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

                //draw player
                glEnable(GL_TEXTURE_2D);
                draw_rectangle(player.playerX - player.playerSize / 2, player.playerY,
player.playerSize, player.playerSize, 0);

                //draw player_bullet
                for (int i = 0; i < LINE_COUNT; i++) {
```

```
                    struct Bullet_list *bullet_elem = bullet_array[i];
                    while (bullet_elem != NULL) {
                        if (is_bullet_on_screen(bullet_elem->bullet)) {
                            if (!bullet_elem->bullet->between || (bullet_elem->bullet->between
&& i % 2 == 0)) {
                                draw_rectangle(bullet_elem->bullet->bulletX, \
                                               bullet_elem->bullet->bulletY, \
                                               bullet_elem->bullet->bulletSize, \
                                               bullet_elem->bullet->bulletSize, 1);
                            }
                        }
                        bullet_elem = bullet_elem->next;
                    }
                }

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

                //TODO
                //draw boss_bullet
                // for (int i = 0; i < LINE_COUNT; i++) {
                //     struct Bullet_list *boss_bullet_elem = boss_bullet_array[i];
                //     while (boss_bullet_elem != NULL) {
                //         if (is_bullet_on_screen(boss_bullet_elem->bullet)) {
                //             draw_rectangle(boss_bullet_elem->bullet->bulletX, \
                //                            boss_bullet_elem->bullet->bulletY, \
                //                            boss_bullet_elem->bullet->bulletSize, \
                //                            boss_bullet_elem->bullet->bulletSize, 2);
                //         }
                //         boss_bullet_elem = boss_bullet_elem->next;
                //     }
                // }

                glDisable(GL_TEXTURE_2D);

                //draw level
                char level[DRAW_TEXT_LENGTH];
                sprintf(level, "Level: Boss");
                draw_text(10, WINDOW_HEIGHT - 30, level);

                //draw boss_lives
                char boss_lives[DRAW_TEXT_LENGTH];
                sprintf(boss_lives, "Boss lives: %d", boss.bossLives);
                draw_text(10, WINDOW_HEIGHT - 50, boss_lives);

            } else {

                //draw player
                glEnable(GL_TEXTURE_2D);
                draw_rectangle(player.playerX - player.playerSize / 2, player.playerY,
player.playerSize, player.playerSize, 0);

                //draw heart
                draw_rectangle(heart.heartX, heart.heartY, heart.heartSize, heart.heartSize,
13);

                //draw bullet
                for (int i = 0; i < LINE_COUNT; i++) {
                    struct Bullet_list *bullet_elem = bullet_array[i];
                    while (bullet_elem != NULL) {
                        if (is_bullet_on_screen(bullet_elem->bullet)) {
                            if (!bullet_elem->bullet->between) {
                                draw_rectangle(bullet_elem->bullet->bulletX, \
                                               bullet_elem->bullet->bulletY, \
                                               bullet_elem->bullet->bulletSize, \
                                               bullet_elem->bullet->bulletSize, 1);
                            } else if (i % 2 == 0) {
                                draw_rectangle(bullet_elem->bullet->bulletX, \
                                               bullet_elem->bullet->bulletY, \
                                               bullet_elem->bullet->bulletSize, \
```

```
                                    bullet_elem->bullet->bulletSize, 1);
                }
            }
            bullet_elem = bullet_elem->next;
        }
    }

    //draw small_asteroid
    for (int i = 0; i < LINE_COUNT; i++) {
        struct Asteroid_list *sa_elem = small_asteroids_array[i]->next;
        while (sa_elem != NULL) {
            if (is_asteroid_on_screen(sa_elem->asteroid)) {
                draw_rectangle(sa_elem->asteroid->asteroidX, \
                               sa_elem->asteroid->asteroidY, \
                               sa_elem->asteroid->asteroidSize, \
                               sa_elem->asteroid->asteroidSize, 3);
            }
            sa_elem = sa_elem->next;
        }
    }

    //draw medium_asteroid
    for (int i = 0; i < LINE_COUNT; i++) {
        struct Asteroid_list *ma_elem = medium_asteroids_array[i]->next;
        while (ma_elem != NULL) {
            if (is_asteroid_on_screen(ma_elem->asteroid)) {
                draw_rectangle(ma_elem->asteroid->asteroidX, \
                               ma_elem->asteroid->asteroidY, \
                               ma_elem->asteroid->asteroidSize, \
                               ma_elem->asteroid->asteroidSize, 4);
            }
            ma_elem = ma_elem->next;
        }
    }

    //draw big_asteroid
    for (int i = 0; i < LINE_COUNT; i++) {
        struct Asteroid_list *ba_elem = big_asteroids_array[i]->next;
        while (ba_elem != NULL) {
            if (is_asteroid_on_screen(ba_elem->asteroid)) {
                draw_rectangle(ba_elem->asteroid->asteroidX, \
                               ba_elem->asteroid->asteroidY, \
                               ba_elem->asteroid->asteroidSize, \
                               ba_elem->asteroid->asteroidSize, 5);
            }
            ba_elem = ba_elem->next;
        }
    }

    glDisable(GL_TEXTURE_2D);

    //draw level
    char level[DRAW_TEXT_LENGTH];
    sprintf(level, "Level: %d", player.currentLevel);
    draw_text(10, WINDOW_HEIGHT - 30, level);

    //draw score
    char score[DRAW_TEXT_LENGTH];
    sprintf(score, "Score: %d", player.playerScore);
    draw_text(10, WINDOW_HEIGHT - 50, score);

    //draw fps
    char fps[DRAW_TEXT_LENGTH];
    sprintf(fps, "fps: %d", frameCountPerSecond);
    draw_text(10, 30, fps);

}

frame_counter();
glutSwapBuffers();
```

```
}
```

- Код файла constants.h

```
#ifndef CONSTANTS_H
#define CONSTANTS_H

// Window constants
#define WINDOW_CAPTION "Space Impact - 5131001/30002"
#define WINDOW_WIDTH 1200
#define WINDOW_HEIGHT 700
#define DRAW_TEXT_LENGTH 15

// rand.c constants
#define RAND_POOL_SIZE 200

// Game canvas constants
#define LINE_COUNT ((WINDOW_HEIGHT - 2 * BORDERS_SIZE) / BA_SIZE)
#define BORDERS_SIZE 75

// Game points constants
#define FOR_SECOND_LEVEL 5
#define FOR_THIRD_LEVEL 15
#define FOR_BOSS_LEVEL 35
#define BOSS_DELAY_VALUE 35

// Game objects constants
#define MAX_ASTEROIDS_IN_BENCH_MODE 100000
#define SA_SPEED 4
#define SA_SIZE 30
#define MA_SPEED 3
#define MA_SIZE 40
#define BA_SPEED 2
#define BA_SIZE 50

#define B_SIZE 10
#define B_SPEED 10

#define PLAYER_SIZE 50
#define PLAYER_LIVES 3
#define PLAYER_SCORE 0
#define PLAYER_START_LEVEL 1
#define PLAYER_MOVE_STEP 10
#define PLAYER_MOVEMENTS_WIDTH 200

#define BOSS_LIVES 10
#define BOSS_SIZE 70

#define HEART_SIZE 30
#define HEART_SPEED 4

// Textures constants
#define TEXTURES_AMT 13
#define PT_FILENAME "../../themes/player.bmp"
#define BT_FILENAME "../../themes/bullet.bmp"
#define BBT_FILENAME "../../themes/boss_bullet.bmp"
#define SAT_FILENAME "../../themes/small_asteroid.bmp"
#define MAT_FILENAME "../../themes/medium_asteroid.bmp"
#define BAT_FILENAME "../../themes/big_asteroid.bmp"
#define BOSST_FILENAME "../../themes/boss.bmp"
#define BGR0_FILENAME "../../themes/background_0heart.bmp"
#define BGR1_FILENAME "../../themes/background_1heart.bmp"
#define BGR2_FILENAME "../../themes/background_2heart.bmp"
#define BGR3_FILENAME "../../themes/background_3heart.bmp"
#define MENUS_FILENAME "../../themes/main_menu_start.bmp"
#define MENUE_FILENAME "../../themes/main_menu_exit.bmp"
#define HEART_FILENAME "../../themes/heart.bmp"

#endif
```

- Код файла draw.h

```
#ifndef DRAW_H
#define DRAW_H

void frame_counter(void);
void draw_rectangle(int, int, int, int, int);
void draw_text(int, int, char*);
void draw_main_menu(void);
void draw_scene(void);

#endif
```

- Код файла extern_pointers.h

```
#ifndef EXTERN_P_H
#define EXTERN_P_H

#include "constants.h"
#include "structures.h"
#include "lists.h"

extern GLuint textures[TEXTURES_AMT+1];

extern bool changed_to_second;
extern bool changed_to_third;
extern bool changed_to_fourth;
extern bool player_is_dead;
extern bool boss_is_dead;
extern bool spawn_asteroids;
extern bool game_just_started;
extern bool megalovania_is_playing;

extern struct Player player;
extern struct Boss boss;
extern struct Heart heart;
extern struct Menu main_menu;

extern int boss_delay;

extern struct Bullet_list *bullet_array[LINE_COUNT];
extern struct Bullet_list *boss_bullet_array[LINE_COUNT];
extern struct Asteroid_list *small_asteroids_array[LINE_COUNT];
extern struct Asteroid_list *medium_asteroids_array[LINE_COUNT];
extern struct Asteroid_list *big_asteroids_array[LINE_COUNT];

extern struct Bullet_list *bullets;
extern struct Bullet_list *boss_bullets;
extern struct Asteroid_list *small_asteroids;
extern struct Asteroid_list *medium_asteroids;
extern struct Asteroid_list *big_asteroids;

extern int update_count;
extern int bullet_count;

extern int frameCountPerSecond;
extern int frameCount;
extern double previousTime;

extern int *random_pool;
extern int random_index;

#endif
```

- Код файла init.h

```
#ifndef INIT_H
#define INIT_H

void main_menu_init(void);
```

```
void init_game(void);

#endif
```

- Код файла keyboard.h

```
#ifndef KEYBOARD_H
#define KEYBOARD_H

void handle_keyboard(unsigned char, int, int);
void handle_movement_keys(int, int, int);
void handle_menu_keyboard(unsigned char, int, int);
void handle_menu_special_keyboard(int, int, int);

#endif
```

- Код файла lists.h

```
#ifndef LISTS_H
#define LISTS_H

#include "structures.h"

struct Asteroid_list * init_asteroid_list_elem(char type);
struct Bullet_list * init_bullet_list_elem();

void remove_from_alist(struct Asteroid_list *list_head, struct Asteroid *a);
void remove_from_blist(struct Bullet_list *list_head, struct Bullet *b);

void for_asteroid_list(struct Asteroid_list *list_head, void (*func)(struct Asteroid*));
void for_bullet_list(struct Bullet_list *list_head, void (*func)(struct Bullet*));

#endif
```

- Код файла rand.h

```
#ifndef RAND_H
#define RAND_H

void init_random_pool();
int get_random_number();

#endif
```

- Код файла structures.h

```
#ifndef STRUCTURES_H
#define STRUCTURES_H

#include <stdbool.h>

struct Player {
    int currentLevel;
    int playerSize;
    int playerY;
    int playerX;
    int playerLives;
    int playerScore;
    bool godMode;
    bool between;
};

struct Boss {
    int bossSize;
    int bossY;
    int bossX;
    int bossLives;
    bool reached_top;
    bool reached_bot;
};
```

```
struct Bullet {
    int bulletSize;
    int bulletX;
    int bulletY;
    int bulletSpeed;
    bool between;
};

struct Asteroid {
    int asteroidSize;
    int asteroidY;
    int asteroidX;
    int asteroidSpeed;
    int line;
};

struct Heart {
    bool spawn;
    int heartSize;
    int heartY;
    int heartX;
    int heartSpeed;
};

struct Menu {
    int option;
};

struct Bullet_list {
    struct Bullet *bullet;
    struct Bullet_list *next;
};

struct Asteroid_list {
    struct Asteroid *asteroid;
    struct Asteroid_list *next;
};

#endif
```

- Код файла update.h

```
#ifndef UPDATE_H
#define UPDATE_H

#include <stdbool.h>
#include "structures.h"

bool is_asteroid_on_screen(struct Asteroid *a);
bool is_bullet_on_screen(struct Bullet *b);

void add_asteroid(struct Asteroid_list **a_array);
void add_boss_bullet(struct Bullet_list **b_array);
void add_bullet(struct Bullet_list **b_array);
void maybe_spawn_heart(struct Asteroid_list *a);

bool is_colliding_ap(struct Asteroid *a, struct Player p);
bool is_colliding_hp(struct Heart h, struct Player p);
bool is_colliding_ba(struct Bullet *b, struct Asteroid *a);
bool is_colliding_bp(struct Bullet *b, struct Player p);
bool is_colliding_bbs(struct Bullet *b, struct Boss bs);
void check_bullet_asteroid_collisions(struct Asteroid_list **asteroids);
void check_asteroid_player_collisions(struct Asteroid_list **asteroid_array);

void update_asteroid_position(struct Asteroid_list **asteroid_array, struct Asteroid
*asteroid);
void update_bullet_position(struct Bullet *bullet);
void update_boss_state(void);
void player_state_update(void);
```

```
void update(int aux);

#endif
```