Trabalho Prático AEDS I
Ikaruga

Danilo Pimentel de Carvalho Costa
E-mail: danilo-p@ufmg.br
Matrícula: 2016058077

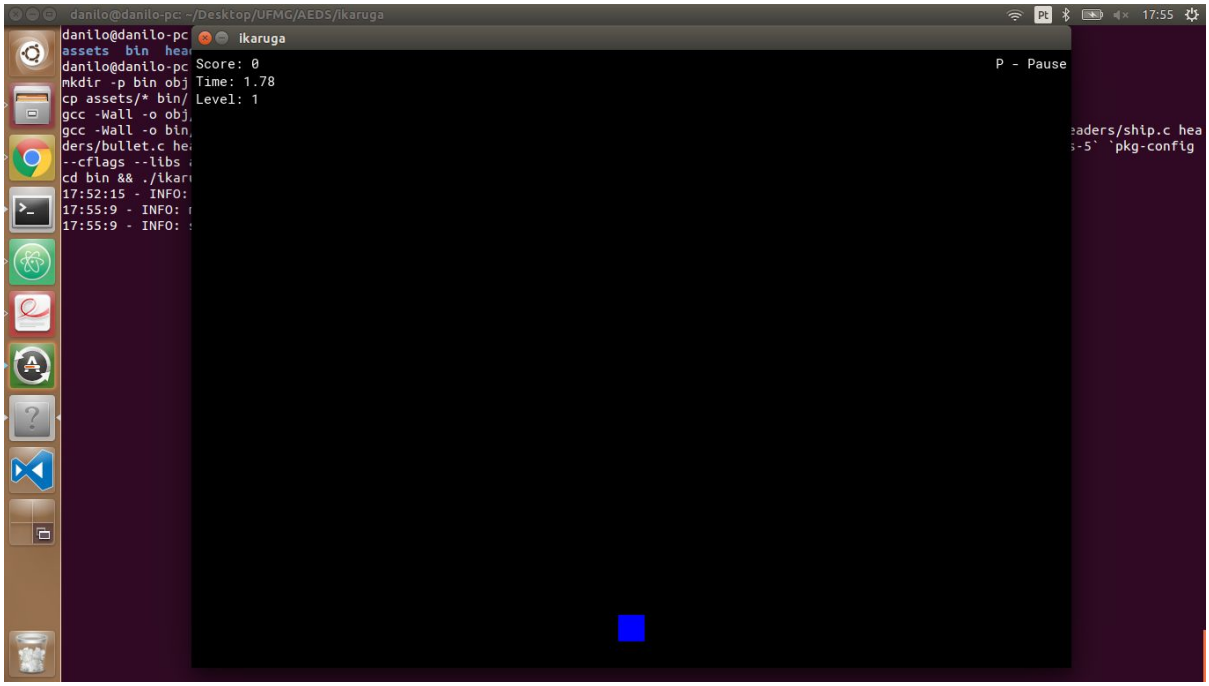
1 - Manual de Uso

Depois de obter o código do jogo, via download ou repositório git, deve-se executar o comando "*make*". Ele irá executar todas as tarefas necessárias para construir o jogo o iniciará logo em seguida.

Resultado esperado

O jogo inicia com um menu que dispõe de duas opções: jogar e sair. Para jogar, pressione a tecla **ENTER**. Para sair, pressione a tecla **Q**. Pressionando a tecla **Q**, o jogo se encerra. Pressionando a tecla **ENTER**, o jogo se inicia, como mostra a imagem a seguir.

Resultado esperado

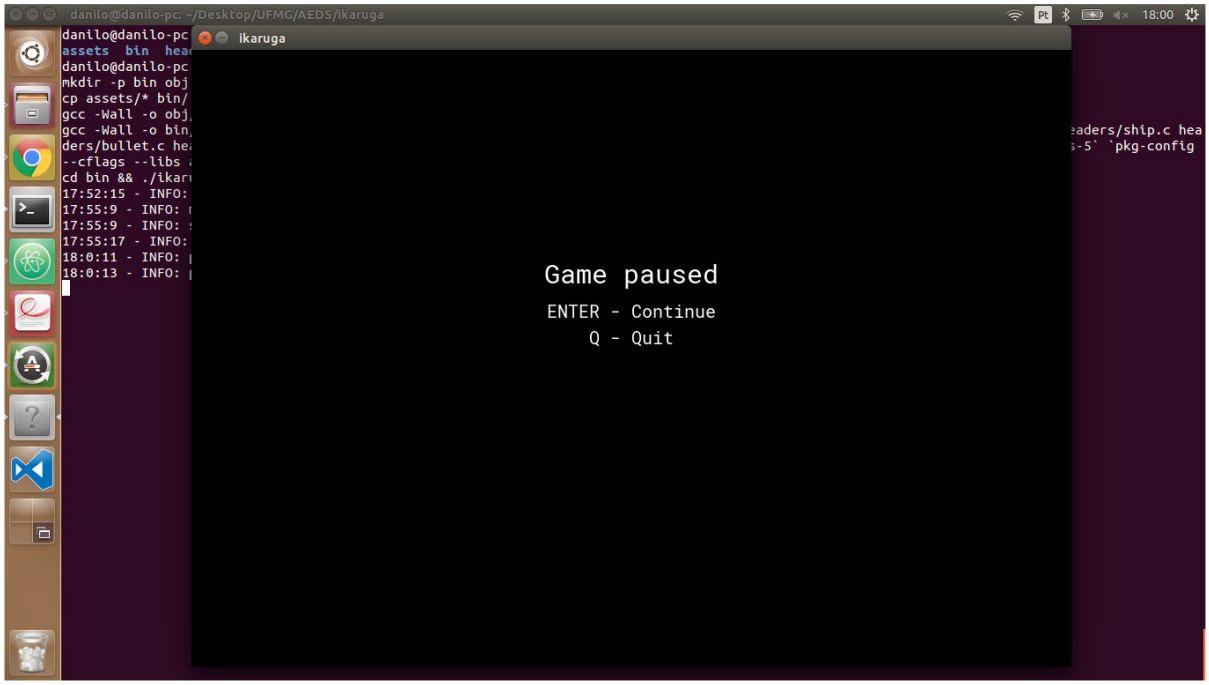


Os controles do *ikaruga* (sua nave) são:

- **W** - Movimenta a nave para cima
- **S** - Movimenta a nave para baixo
- **A** - Movimenta a nave para a esquerda
- **D** - Movimenta a nave para a direita
- **G** - Muda a cor da nave
- **H** - Atira a nave

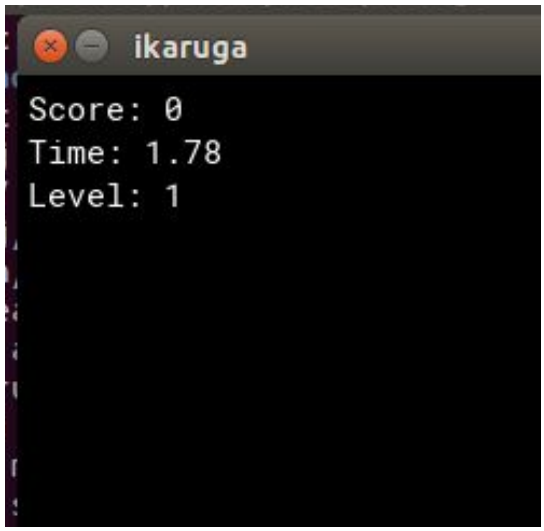
Além das movimentações, a tela de jogo dispõe da opção de pausa. Para pausar o jogo, pressione **P**.

Resultado esperado



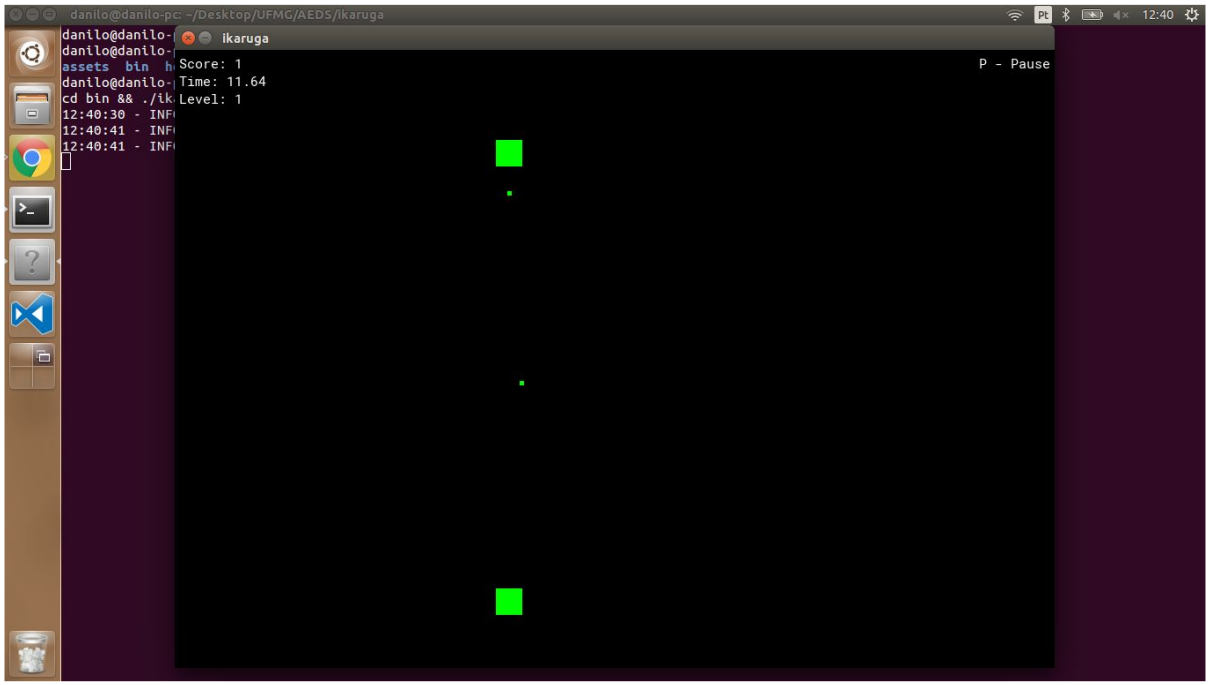
O menu de pausa dispõe de duas opções: voltar ao jogo e sair da partida. Para voltar ao jogo, pressione **ENTER.** Para sair da partida, pressione **Q.**

No canto superior direito da tela são mostradas informações do jogo, como pontuação (*Score*), tempo decorrido (*Time*) e nível de dificuldade (*Level*).

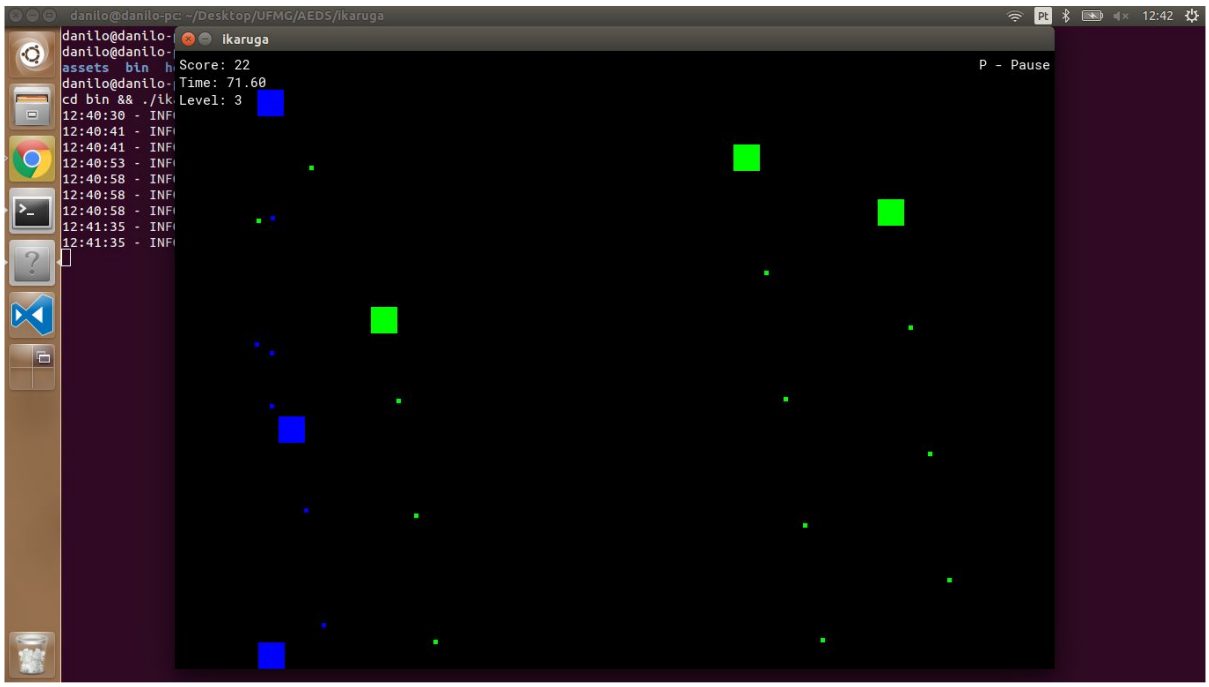


Ao decorrer do jogo, o nível de dificuldade é aumentado de acordo com o tempo. Os inimigos começam a nascer com maior frequência, e a cadência de seus tiros aumenta. Os pontos correspondem a quantos inimigos foram eliminados.

Dificuldade baixa

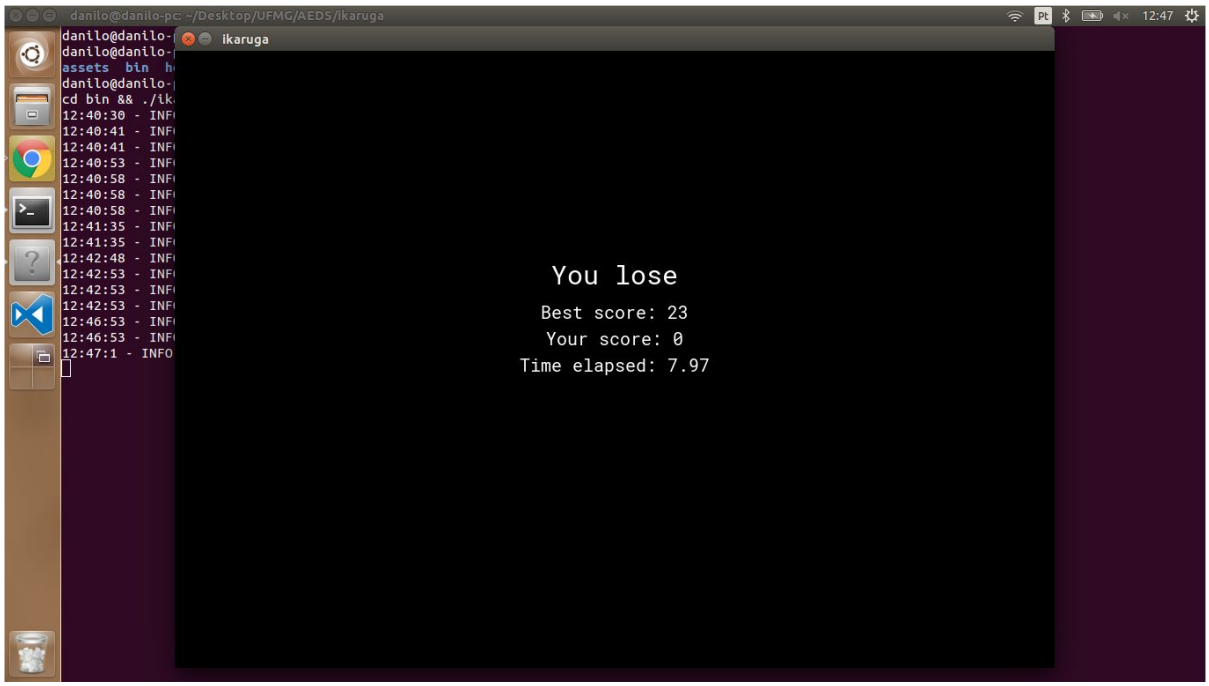


Dificuldade aumentada



O jogo termina quando um dos inimigos consegue eliminar o ikaruga através de seus tiros ou de uma colisão entre as naves. Ao final do jogo são mostradas informações da partida como tempo (*Time elapsed*) e pontuação (*Your score*) e informações da melhor pontuação do jogo (*Best score*). Depois de 5 segundos o jogador é levado à tela inicial, onde pode começar outra partida ou sair.

Final da partida



2 - Código do Jogo

2.1 - Make

O executável do jogo é gerado através do *make*, que realiza as seguintes tarefas:

- ***setup***: São criadas as pastas necessárias;
- ***copy***: São copiados os arquivos da pasta *assets/*, necessários para o funcionamento do jogo, para a pasta *bin/*, destino do arquivo executável do jogo;
- ***ikaruga***: Compila o programa utilizando os *headers* do próprio jogo e da biblioteca *Allegro*;

- ***run***: Entra na pasta bin, executa o jogo e volta à raíz do projeto.

2.2 - Organização do projeto

A fim de organizar o projeto, o código do jogo foi dividido em *headers* contendo funções para cada tipo de tarefa necessária para o funcionamento do jogo. Assim, o arquivo principal *ikaruga.c* está na raíz do projeto, e os *headers* se encontram na pasta *headers/*.

O arquivo principal contém poucas linhas e é responsável pela inicialização e lógica de transição entre o menu principal e a execução do jogo em si.

ikaruga.c

```c
#include "headers/common.h"
#include "headers/game.h"
#include "headers/display.h"
#include "headers/menu.h"

#include <allegro5/allegro.h>

int main () {

    ALLEGRO_DISPLAY *display = NULL;
    bool quit = false;

    if(!initGame()) {
        logerror("[FATAL] Failed to init game. Exiting...\n");
        return 0;
    }

    display = createDisplay();
    if(!display) {
        logerror("[FATAL] Failed to create display. Exiting...\n");
        return 0;
    }

    while(!quit) {
        switch (mainMenu(display)) {
            case 1:
                startGame(display);
            break;
            case 0:
                quit = true;
            break;
        }
    }

    destroyDisplay(&display);

    return 0;
}
```

Funções como *initGame(), createDisplay()* entre outras utilizadas no arquivo principal são declaradas nos *headers* do jogo. Os *headers* são:

- ***config.h***: Configurações gerais do jogo;
- ***common.h***: Funções e estruturas comuns a todas as partes do jogo;
- ***game.h***: Funções relacionadas à lógica do jogo;
- ***menu.h***: Funções que exibem menus do jogo;
- ***display.h***: Funções relacionadas à manipulação do display;
- ***score.h***: Contém a lógica de manipulação do arquivo que armazena a melhor pontuação do jogo;
- ***element.h***: Funções e estrutura que servirá de base para todos os elementos que fazem parte do jogo.
- ***ship.h***: Funções e estrutura relacionadas às naves do jogo, tanto para o *ikaruga* quanto para os inimigos.
- ***bullet.h***: Funções e estrutura relacionadas às balas disparadas pelas naves.

IMPORTANTE: A fim de resumir a documentação do jogo, mantendo as explicações focadas, aqui somente será explicada a **lógica do jogo**. Todos os *headers* possuem uma descrição acima de cada estrutura, elemento ou função contidas no arquivo fonte correspondente. Caso haja dúvida no objetivo de alguma função, deve-se consultar o *header* de origem a fim de procurar uma explicação.

2.3 - O Jogo

O funcionamento do jogo, que compreende o controle de ações do jogador, ações dos inimigos, geração de inimigos, movimentações dos elementos, pontuação, níveis, dificuldade do jogo e renderização estão contidos no *header* ***game.h***, especificamente na função ***startGame()***.

A execução da função possui 3 passos, descritos nas sub-seções a baixo.

2.3.1 - Inicialização

Este passo inicializa variáveis necessárias para a execução da função. Algumas das mais importantes são *ALLEGRO_TIMER* ****timer*** *e ALLEGRO_EVENT_QUEUE* ****event_queue***.

timer é o temporizador da biblioteca *Allegro* responsável pela geração de eventos do tipo timer, utilizados para a execução das ações do jogo com sincronia.

event_queue é a variável utilizada como *handler* para os eventos interessantes ao jogo, como eventos relacionados ao teclado e ao temporizador.

Variáveis utilizadas ao longo da execução

```
30  bool startGame(ALLEGRO_DISPLAY *display) {
31
32      // Create the game timer and the event queue
33      ALLEGRO_TIMER *timer = createTimer();
34      ALLEGRO_EVENT_QUEUE *event_queue = createEventQueue(display, timer);
35
36      // The ikaruga ship, called hero
37      Ship hero;
38      // The enemies array
39      Ship *enemies = NULL;
40
41      // The bullets array and the aux variable new_bullet
42      Bullet *bullets = NULL, new_bullet;
43
44      // The timestamp of the last enemy created, initialized with the
45      // configurated offset
46      double last_enemy_created = FIRST_ENEMY_OFFSET;
47      // Aux variable to control the levels difficulty
48      double level_factor;
49      // Time playing the game
50      double time_elapsed;
51      // Store the moment that the game started
52      double start_time = al_get_time();
53
54      // Aux variable for quit the game
55      bool quit = false;
56      // Flag for indicating the hero fire
57      bool fire_key = false;
58
59      // Counters
60      int i, j;
61      int bullets_count = 0;
62      int enemies_count = 0;
63      int score = 0;
64      int level = 1;
65
66      // Store the code of the move key pressed
67      int move_key_1 = -1;
68      int move_key_2 = -1;
69
```

Neste passo variáveis relacionadas aos elementos são declaradas, como *Ship* ***hero***, a estrutura do *ikaruga, o* vetor que contém as balas de todo o jogo (*Bullet* ****bullets***) e o vetor que contém todos os inimigos do jogo (*Ship* ****enemies***).

Aqui estão presentes também variáveis de controle como os contadores *int* ***score*** e *int* ***level*** são responsáveis por armazenar a pontuação e o nível que o jogador se encontra. Além destas, outra importante é a variável que corresponde à dificuldade do jogo, a *double* ***level_factor***, calculada e utilizada no próximo passo da execução.

2.3.2 - Loop

Neste passo, a lógica do jogo é executada através de um loop que utiliza uma "*flag" bool* ***quit***, que indica a necessidade de finalização do jogo, seja pela eliminação do jogador ou pelo desejo de saída indicado no menu de pausa.

No início do loop é declarada uma variável *ALLEGRO_EVENT* ***e***, que é utilizada juntamente com a variável ***event_queue*** para esperar um evento. Quanto identificado um evento, verifica-se se este é interessante ao jogo.

Os eventos utilizados pela lógica do jogo são relacionados ao temporizador (*ALLEGRO_EVENT_TIMER*) e ao teclado (*ALLEGRO_EVENT_KEY_DOWN* e *ALEGRO_EVENT_KEY_UP*).

Game loop

```
89
90      // Game loop
91      while(!quit) {
92          ALLEGRO_EVENT e;
93          al_wait_for_event(event_queue, &e);
94
95          // Handler for the timer event
96 ›        if(e.type == ALLEGRO_EVENT_TIMER) {⋯
199 ›       else if(e.type == ALLEGRO_EVENT_KEY_DOWN) {⋯
244 ›       else if(e.type == ALLEGRO_EVENT_KEY_UP) {⋯
263     }
264
```

Os procedimentos realizados nos eventos relacionados ao teclado são para identificação de **movimentação, disparo e troca de cor do *ikaruga*** bem como os procedimentos relacionados ao **menu de pausa**.

São utilizadas as variáveis *int* ***move_key_1*** e *int* ***move_key_2*** para armazenar o código das teclas de movimento pressionadas pelo jogador. Estas são utilizadas no controle de **movimentação do *ikaruga***, presente nos procedimentos do evento do temporizador.

A variável *bool* ***fire_key*** é utilizada como *"flag"* para indicar se o jogador está pressionando a **tecla de disparo H**.

```
else if(e.type == ALLEGRO_EVENT_KEY_DOWN) {
    switch(e.keyboard.keycode) {
        case ALLEGRO_KEY_W:
        case ALLEGRO_KEY_S:
        case ALLEGRO_KEY_A:
        case ALLEGRO_KEY_D:
            // If both keys are different from the pressed key
            if(move_key_1 != e.keyboard.keycode &&
                    move_key_2 != e.keyboard.keycode) {

                // If the first key is not pressed, give it a value.
                if(move_key_1 == -1) {
                    move_key_1 = e.keyboard.keycode;
                }
                // If the second key is not pressed, give it a value.
                else if(move_key_2 == -1) {
                    move_key_2 = e.keyboard.keycode;
                }
                // If both keys are pressed, discard the second value.
                else {
                    move_key_2 = e.keyboard.keycode;
                }
            }
        break;

        case ALLEGRO_KEY_H:
            fire_key = true;
        break;
```

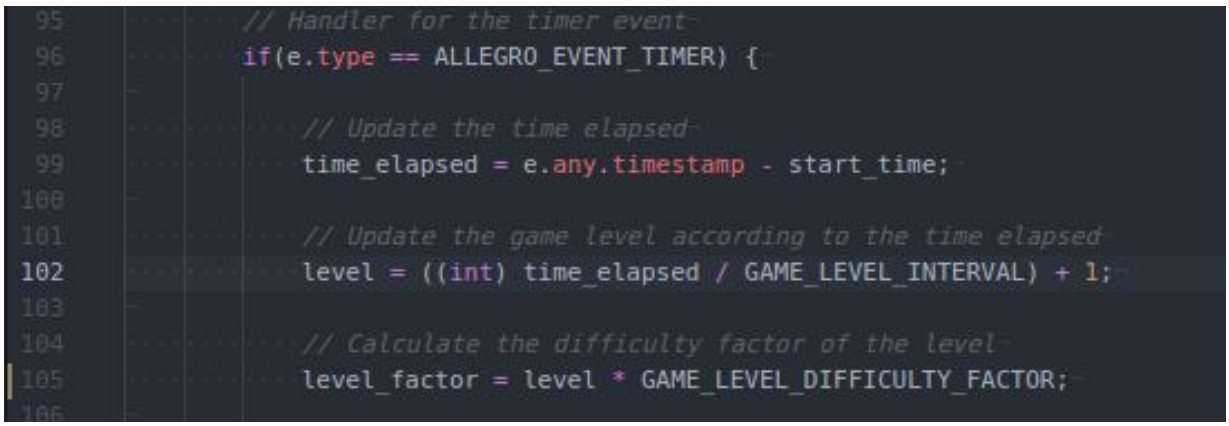
```
else if(e.type == ALLEGRO_EVENT_KEY_UP) {
    switch(e.keyboard.keycode) {
        case ALLEGRO_KEY_W:
        case ALLEGRO_KEY_S:
        case ALLEGRO_KEY_A:
        case ALLEGRO_KEY_D:
            if(move_key_1 == e.keyboard.keycode)
                move_key_1 = -1;

            if(move_key_2 == e.keyboard.keycode)
                move_key_2 = -1;

        break;

        case ALLEGRO_KEY_H:
            fire_key = false;
        break;
    }
```

O tratamento e os procedimentos relacionados à **troca de cor do *ikaruga***, identificada pelo pressionamento da tecla **G**, e a abertura do **menu de pausa** do jogo, disparada pelo pressionamento da tecla **P** são executados também nesta parte da função, como mostra a imagem a seguir:

```
228                 case ALLEGRO_KEY_G:
229                     // Change the hero target
230                     if(hero.target == alpha)
231                         setShipTarget(&hero, gama);
232                     else if(hero.target == gama)
233                         setShipTarget(&hero, alpha);
234                 break;
235
236                 case ALLEGRO_KEY_P:
237                     // Pause the game and stop the timers
238                     al_stop_timer(timer);
239                     quit = (pauseMenu(display) == 0);
240                     al_start_timer(timer);
241                 break;
242             }
```

Os procedimentos realizados no evento relacionado ao temporizador são para controle de **níveis, renderização, movimentação dos elementos, disparos das naves e colisões**.

Controle de níveis e de tempo decorrido



```
95          // Handler for the timer event
96          if(e.type == ALLEGRO_EVENT_TIMER) {
97
98              // Update the time elapsed
99              time_elapsed = e.any.timestamp - start_time;
100
101             // Update the game level according to the time elapsed
102             level = ((int) time_elapsed / GAME_LEVEL_INTERVAL) + 1;
103
104             // Calculate the difficulty factor of the level
105             level_factor = level * GAME_LEVEL_DIFFICULTY_FACTOR;
106
```

O nível é calculado a partir da divisão do tempo decorrido pelo valor *GAME_LEVE_INTERVAL* configurado no *header config.h.*

A dificuldade aumentada pelo nível é calculada a partir da multiplicação do nível atual pela configuração do fator *GAME_LEVEL_DIFFICULTY_FACTOR*, que é utilizado para o aumento da frequência de geração de inimigos e intervalo entre os disparos destes.

Logo após os cáculos, acontece a renderização dos elementos e das marcas no display, utilizando a variável ***hero*** e iterando sobre o vetor de inimigos e balas, como mostra a imagem a seguir:

```
106
107         /***** Rendering *****/
108
109         clearDisplay(display, al_map_rgb(0,0,0));
110
111         renderShip(hero, display);
112
113         for(i=0; i<bullets_count; i++)
114             renderBullet(bullets[i], display);
115
116         for(i=0; i<enemies_count; i++)
117             renderShip(enemies[i], display);
118
119         renderGameDisplay(display, level, score, time_elapsed, size, FONT_SIZE_SM);
120
```

Após a renderização são tratadas as teclas de controle do *ikaruga e disparo do* ikaruga, a geração de novos inimigos e os disparos destes.

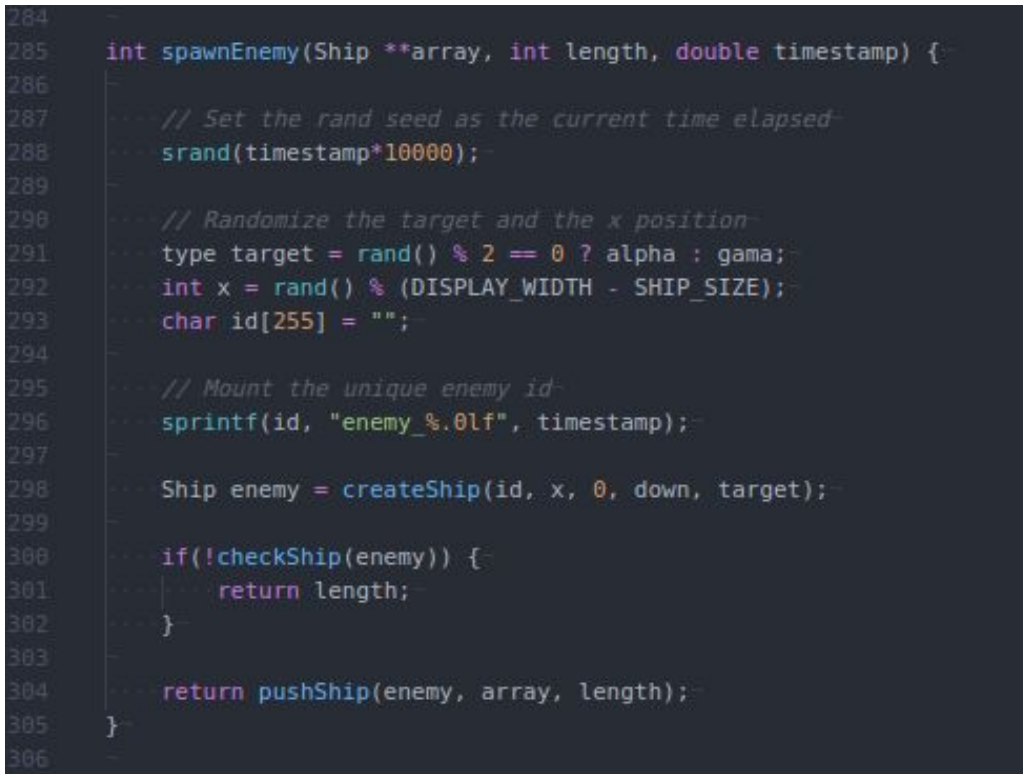
São utilizadas as variáveis mencionadas anteriormente **move_key_1** e **move_key_2** para a movimentação do *ikaruga. A flag* ***fire_key*** é utilizada para tratar a intenção do jogador em atirar. Caso seja verdadeira, é criada uma nova bala, contendo informações sobre a cor e de que foi o *ikaruga* que a disparou.

A criação de novos inimigos é gerenciada pela função *spawnEnemy()*, presente no *header game.h.* Esta função realiza os procedimentos de cálculo randômico da posição e da cor do novo inimigo, bem como a inicialização e a inserção no vetor de inimigos ***enemies***.

O próximo procedimento realiza os disparos de todos os inimigos presentes na tela. São criadas novas balas, contendo a informação sobre a cor do inimigo e de que foi este inimigo que disparou esta bala.

```
120
121         /***** Game action *****/
122
123         // Move the ship with the pressed keys
124         moveShip(&hero, move_key_1, SHIP_STEP_SIZE);
125         moveShip(&hero, move_key_2, SHIP_STEP_SIZE);
126
127         // Check if the fire key is pressed
128         if(fire_key) {
129             // Try to fire the ship and catch the result
130             new_bullet = fireShip(&hero, SHIP_FIRE_INTERVAL, e);
131
132             // Check if the bullet is valid
133             if(checkBullet(new_bullet))
134                 bullets_count = pushBullet(new_bullet, &bullets, bullets_count);
135         }
136
137         // Check if is time to spawn a new enemy
138         if(last_enemy_created + ENEMY_SPAWN_INTERVAL / level_factor < time_elapsed) {
139             // Spawn a new enemy
140             enemies_count = spawnEnemy(&enemies, enemies_count, time_elapsed);
141             last_enemy_created = time_elapsed;
142         }
143
144         // Fire all the enemies
145         for(i=0; i<enemies_count; i++) {
146             new_bullet = fireShip(&enemies[i], ENEMY_FIRE_INTERVAL / level_factor, e);
147             if(checkBullet(new_bullet))
148                 bullets_count = pushBullet(new_bullet, &bullets, bullets_count);
149         }
150
```

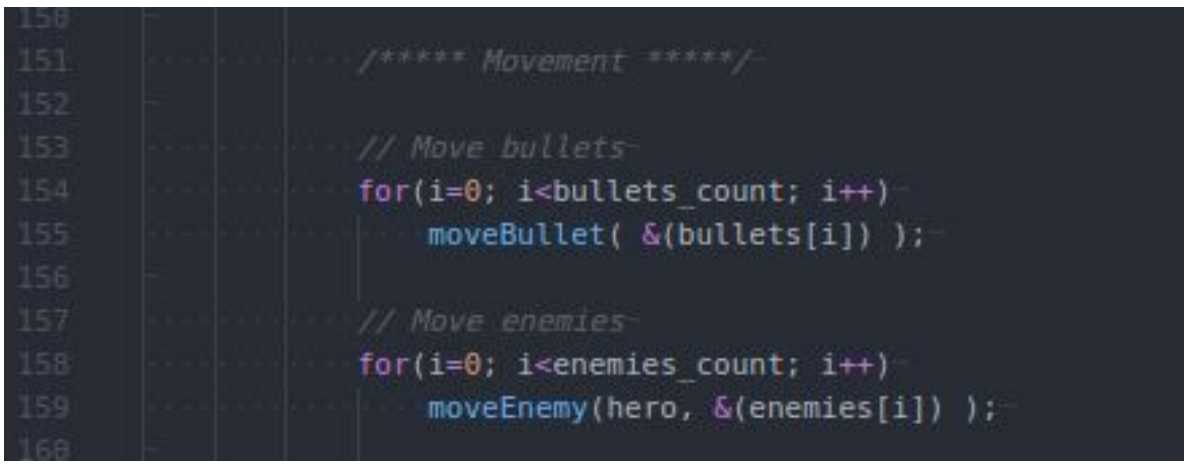
Criação e inicialização de um novo inimigo



```
int spawnEnemy(Ship **array, int length, double timestamp) {

    // Set the rand seed as the current time elapsed
    srand(timestamp*10000);

    // Randomize the target and the x position
    type target = rand() % 2 == 0 ? alpha : gama;
    int x = rand() % (DISPLAY_WIDTH - SHIP_SIZE);
    char id[255] = "";

    // Mount the unique enemy id
    sprintf(id, "enemy_%.0lf", timestamp);

    Ship enemy = createShip(id, x, 0, down, target);

    if(!checkShip(enemy)) {
        return length;
    }

    return pushShip(enemy, array, length);
}
```

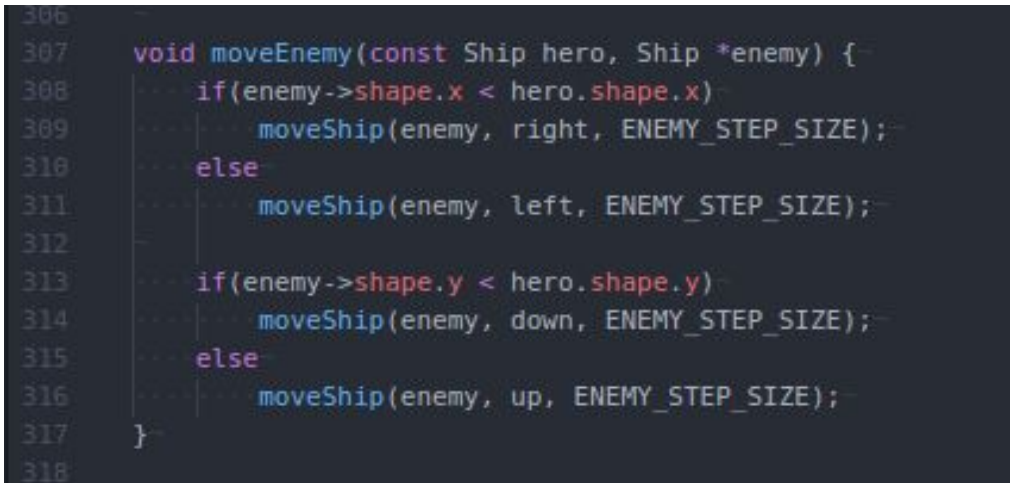
Após a criação de novos inimigos é realizada a movimentação das balas disparadas tanto pelos inimigos quanto pelo *ikaruga* e a movimentação dos inimigos em direção ao *ikaruga.* A lógica para a movimentação dos inimigos está presente na função *moveEnemy()*, declarada no *header game.h.*

Movimentação dos disparos e dos inimigos



```
            /***** Movement *****/

            // Move bullets
            for(i=0; i<bullets_count; i++)
                moveBullet( &(bullets[i]) );

            // Move enemies
            for(i=0; i<enemies_count; i++)
                moveEnemy(hero, &(enemies[i]) );
```

Movimentação do inimigo em direção ao *ikaruga*



```
306
307  void moveEnemy(const Ship hero, Ship *enemy) {
308      if(enemy->shape.x < hero.shape.x)
309          moveShip(enemy, right, ENEMY_STEP_SIZE);
310      else
311          moveShip(enemy, left, ENEMY_STEP_SIZE);
312
313      if(enemy->shape.y < hero.shape.y)
314          moveShip(enemy, down, ENEMY_STEP_SIZE);
315      else
316          moveShip(enemy, up, ENEMY_STEP_SIZE);
317  }
318
```

O último passo do tratamento do evento do temporizador consiste na **detecção de colisões entre os elementos do jogo**. Neste passo, são tratadas as seguintes colisões:

- ***Ikaruga* - Inimigos**: Caso um inimigo colida com o *ikaruga*, o jogador perde o jogo. Para sair do loop, utiliza-se a "*flag*" ***quit*** para sair do loop e encerrar o jogo**.**
- **Balas dos inimigos - *Ikaruga***: Caso uma das balas do inimigo atinja o *ikaruga*, e a cor da bala é a mesma da cor do *ikaruga*, o jogador perde o jogo. Para sair do loop, utiliza-se a "*flag*" ***quit*** para sair do loop e encerrar o jogo**.**
- **Balas do *Ikaruga* - Inimigos**: Caso uma das balas do *ikaruga* atinja um inimigo, e a cor da bala é a mesma do inimigo, o jogador acaba de marcar um ponto. Assim, é retirada a bala e o inimigo dos vetores ***bullets*** e ***enemies*** respectivamente, e é aumentada em 1 ponto a pontuação do jogador na variável ***score*.**
- **Balas - Bordas do mapa**: Caso uma das balas de qualquer nave atinja as bordas do mapa, esta é retirada do vetor ***bullets***.

Todo o procedimento de checagem de colisão pode ser visto na imagem abaixo:

Checagem de colisões

```
/***** Colisions check *****/

// Checking hero - enemy colisions
for(i=0; i<enemies_count; i++)
    if(checkShipsColision(hero, enemies[i])) quit = true;

// Checking enemy bullets - hero colisions
for(i=0; i<bullets_count; i++)
    if( hero.target == bullets[i].target &&
        checkBulletShipColision(bullets[i], hero) &&
        strstr(bullets[i].id, "enemy")
    ) quit = true;

// Checking hero's bullets - enemy colisions
for(i=0; i<enemies_count; i++)
    for(j=0; j<bullets_count; j++)
        if( enemies[i].target == bullets[j].target &&
            checkBulletShipColision(bullets[j], enemies[i]) &&
            strstr(bullets[j].id, "hero")
        ) {
            // Remove the enemy and the bullet from the game
            enemies_count = popShip(enemies[i], &enemies, enemies_count);
            bullets_count = popBullet(bullets[j], &bullets, bullets_count);

            // Increase the score
            score++;

            // Exit the first loop to avoid accessing invalid
            // pointers on the next iteration
            break;
        }

// Checking bullet - display colisions
for(i=0; i<bullets_count; i++)
    if(checkBulletDisplayColision(bullets[i]))
        bullets_count = popBullet(bullets[i], &bullets, bullets_count);
```

2.3.3 - Finalização da partida

Após o fim do jogo (saída do loop), a finalização da partida acontece. A tela de final de partida é exibida ao jogador através da função *gameFinishedMenu*, declarada no *header menu.h*.

Este passo contém os procedimentos de destruição dos elementos do jogo, como as balas criadas, os inimigos criados, o próprio *ikaruga*, bem como a destruição do temporizador ***timer*** e da fila de eventos ***event_queue***. São destruídos todos os *bitmaps* das naves e das balas, e liberada toda a memória alocada para os vetores e estruturas dos elementos do jogo. Todo o processo é mostrado na imagem abaixo:

Procedimentos de finalização da partida

```
265     // Game finished. Show the score, best score and the time_elapsed
266     gameFinishedMenu(display, score, time_elapsed);
267
268     /* Destroy the game elements */
269
270     destroyShip(&hero);
271
272     for(i = 0; i < enemies_count; i++)
273         destroyShip(&(enemies[i]));
274
275     for(i = 0; i < enemies_count; i++)
276         destroyBullet(&(bullets[i]));
277
278     free(enemies);
279     free(bullets);
280
281     destroyTimer(&timer);
282     destroyEventQueue(&event_queue);
283
284     loginfo("startGame finish");
285
```