

**Documentação – Trabalho Prático**  
**“Star Jam”**  
**Universidade Federal de Minas Gerais**  
**Programação e Desenvolvimento de Software I**

## **1 INTRODUÇÃO AO JOGO**

### **1.1 OVERVIEW**

Antes de tudo, você pode ter acesso ao código atualizado do jogo em:  
<https://github.com/Brendo-Lino/Star-Jam> (Somente a partir do dia 18/02/2022)

Star Jam é um jogo de resistência, ou seja, uma sobrevivência sem fim! O jogador inicia o jogo com 5 pontos de vida, aos quais são descontados ao colidir com inimigos. A principal mecânica é o tiro especial, uma variação do tiro normal, porém que precisa ser carregado segurando a barra de espaço por um tempo.

Ao abrir o jogo, pode-se iniciar uma partida apertando ENTER, o mesmo vale para quando perder.

O jogador pode destruir naves inimigas com seus tiros, apertando espaço para dispará-los, gerando pontuação para seu “score”, ao qual é salvo ao fim de cada partida em um arquivo de texto.

Caso o jogador mantenha o espaço pressionado, uma barra de energia começará a ser carregada, ao atingir o seu valor máximo, o jogador pode disparar essa energia em um tiro concentrado que não é destruído no primeiro contato com um inimigo. Se o jogador tiver moedas, elas serão gastas para lançar um tiro ainda mais poderoso que dizimará tudo em sua frente.

Colidir com um inimigo descontará um ponto de vida, já um planeta descontará todos os pontos de vida, terminando o jogo imediatamente.

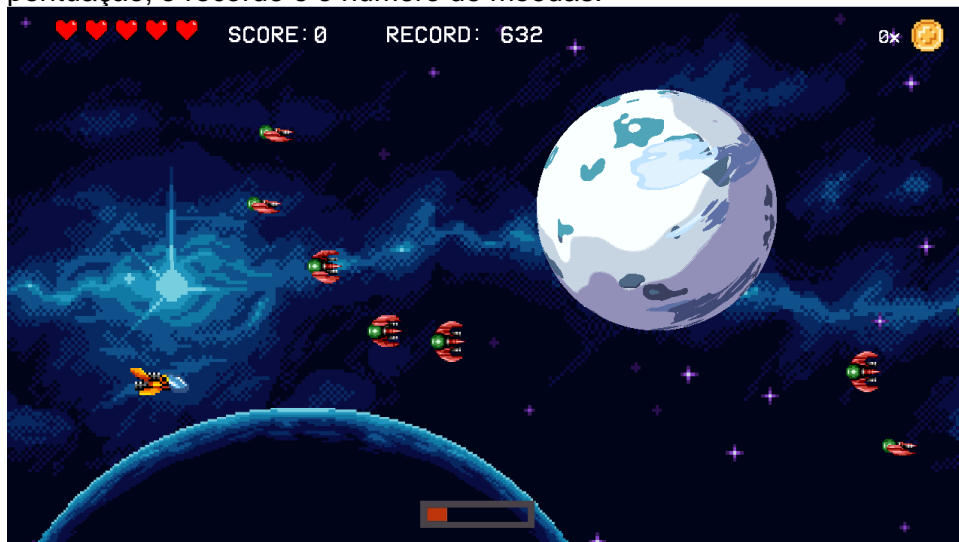
Planetas são esferas arredondadas de matéria condensada que viajam o espaço em busca de sua próxima vítima. Rápidos, de tamanhos variados e sorrateiros, eles aparecem de surpresa com uma velocidade alarmante, prontos para destruir tudo que está em sua frente.

Inimigos colidem entre si e são absorvidos pelos planetas, porém não geram pontuação nesses casos.

Moedas e Pontos de Vida podem ser dropados pelos inimigos quando mortos por um tiro ordinário não carregado.

## 1.2 TELA DO JOGO

Na parte superior, da esquerda para a direita, encontram-se os pontos de vida, a pontuação, o recorde e o número de moedas.



Na parte inferior, encontra-se a barra de energia, a qual é carregada ao segurar o botão SPACEBAR.

## 1.3 CONTROLES

Movimentação da Nave: Setas CIMA/ESQUERDA/BAIXO/DIREITA.

Atirar: SPACEBAR. Manter pressionado para o tiro especial.

Iniciar/Reiniciar o Jogo: ENTER.

## 2 ENTENDENDO O CÓDIGO

Escrito utilizando as melhores práticas possíveis, ou quase, o programa é separado em módulos, cada um responsável por uma parte vital do jogo.

As entidades e objetos do jogo foram declaradas, sempre que possível como structs, algumas usando outras structs dentro dela mesma, para uma boa organização do código e facilidade na modularidade. Dentre elas:

- Box – Uma estrutura de caixa, guarda coordenadas que facilitam em testes de colisão.
- Movable\_Object – Principal estrutura do jogo, guarda um pseudo-objeto. Pode ser enviada para o módulo “motion.c” para atualizar sua caixa, coordenadas, estado de animação, imagem, desenhar o objeto e testar colisões com outros objetos.
- Background – Usado para representar um plano de fundo. Pode ser enviada ao módulo “backgrounds.c” para atualizar seus valores, criando uma animação e desenhando o plano de fundo na tela.
- Drop – Usado para representar um objeto dropavel. Pode ser enviada ao módulo “drops.c” para ter seus valores atualizados, testar colisões ou pick-ups e aplicar seus respectivos efeitos para com a nave.

- Entity – Usado para representar uma entidade. Pode ser enviada ao módulo “entities.c” para atualizar seus valores, testar colisões e ser desenhada.
- Planet – Usado para representar um planeta. Pode ser enviada ao módulo “planets.c” para ter seus valores atualizados e testar colisões.
- Ship – Usado para representar a Nave do jogo. Pode ser enviada ao módulo “ships.c” para atualizar seus valores, disparar um tiro, testar colisões e atualizar seu sprite.
- Shot – Usado para representar os tiros – Pode ser enviada ao módulo “shots.c” para ter seus valores atualizados e testar colisões.
- 

Os módulos serão apresentados por ordem alfabética, com exceção do “game.c” e “game.h”, aos quais virão primeiro por serem os mais importantes, visto que são todos interligados e não há ordem sequencial de execução, sendo separados em tópicos, representando ambos o arquivo “.c” e o arquivo “.h”.

## 2.1 GAME

Módulo principal, responsável por conectar todos os outros módulos chamando suas funções e fazendo o jogo funcionar.

```
#include "game.h"

/* Structural Variables */
int running = 1;
int playing = 0;
int selecting = 1;
int waiting = 0;
int lost = 0;

/* Display and Event Queue */
ALLEGRO_DISPLAY *display = NULL;
ALLEGRO_EVENT_QUEUE *event_queue = NULL;

/* Timers */
ALLEGRO_TIMER *timer = NULL;

/* Fonts */
ALLEGRO_FONT *size_64 = NULL;
ALLEGRO_FONT *size_32 = NULL;
ALLEGRO_FONT *size_24 = NULL;
ALLEGRO_FONT *size_16 = NULL;

ALLEGRO_BITMAP *img_game_over = NULL;
ALLEGRO_BITMAP *img_press_start = NULL;

/* Sounds */
ALLEGRO_SAMPLE *soundtrack = NULL;
```

## Armazena:

- As variáveis estruturais, responsáveis por indicar se o jogo está rodando, se o jogador está em partida, se está no menu, se está esperando ou se perdeu.
- Ponteiros de display, fila de eventos, timer, fontes, imagens de início e fim de jogo e a música de fundo.

```
int main();

void game_start(void);

void game_stop(void);

void game_lose(void);

void game_reset(void);

void game_run(void);

void game_process_keyboard(ALLEGRO_EVENT *event);

void game_load(void);

void game_load_fonts(void);

int game_reset_timers(void);

int game_setup_allegro(void);
```

## Funções:

- int main(); primeira a ser chamada, carrega e roda o jogo.
- void game\_start(void); inicia todos os valores e inicia uma partida.
- void game\_stop(void); para o jogo.
- void game\_lose(void); chamada ao perder o jogo, mostra a tela de derrota.
- void game\_reset(void); reinicia o jogo, reseta todos os valores e salva a pontuação máxima.
- void game\_run(void); principal função, contém o loop principal que faz o jogo rodar e chama todas as outras funções, toca a música de fundo e reinicia os timers.
- void game\_process\_keyboard(ALLEGRO\_EVENT \*event); processa os eventos de teclado, como movimentação e tiros;
- void game\_load(void); carrega imagens, sons, objetos, inimigos e planetas.
- void game\_load\_fonts(void); carrega as fontes.
- int game\_reset\_timers(void); reinicia os timers.
- int game\_setup\_allegro(void); inicia os módulos do allegro.

## 2.2 BACKGROUNDS

Módulo responsável pelos planos de fundo.

```
#include "game.h"

ALLEGRO_TIMER *background_timer = NULL;

/* Backgrounds */
Background bg_default;
Background bg_planet;
Background bg_stars;

void backgrounds_load(void);

void backgrounds_reset(void);

void backgrounds_update(void);
void backgrounds_update_background(Background *bg);
```

**Armazena:**

- Planos de fundo do jogo;

**Funções:**

- void backgrounds\_load(void); carrega os planos de fundo.
- void backgrounds\_reset(void); reinicia os valores dos planos de fundo.
- void backgrounds\_update(void); atualiza todos os planos de fundo.
- void backgrounds\_update\_background(Background \*bg); atualiza um plano de fundo específico e desenha ele na tela.

## 2.3 COLLISIONS

Responsável por detectar colisões entre objetos ou planetas.

```
#include "game.h"

int collisions_check_box(Box box1, Box box2);

int collisions_check_planet(Planet *planet, Box box);
```

### Funções:

- int collisions\_check\_box(Box box1, Box box2); detecta uma colisão entre duas caixas.
- int collisions\_check\_planet(Planet \*planet, Box box); detecta uma colisão entre uma caixa e um planeta.

## 2.4 DROPS

Responsável pelos objetos dropaveis do jogo, tal como vidas e coins.

```
#include "game.h"

Drop drops[NUM_DROPS];

ALLEGRO_BITMAP *img_health = NULL;
ALLEGRO_BITMAP *img_coin_sheet = NULL;

Drop health;

void drops_load(void);

void drops_reset(void);

int drops_create(DropType type, int x, int y);

void drops_update(void);

void drops_health_effect(void);

void drops_coin_effect(void);
```

### Armazena:

- Imagens da vida e do coin.

### Funções:

- void drops\_load(void); carrega as imagens e reinicia os drops.
- void drops\_reset(void); reinicia os valores drops.
- int drops\_create(DropType type, int x, int y); cria o drop específico nas coordenadas específicas e atribui seus valores de acordo com o tipo.
- void drops\_update(void); atualiza todos os drops, testa colisões e aplica efeitos.
- void drops\_health\_effect(void); efeitos de pick-up da vida;
- void drops\_coin\_effect(void); efeitos de pick-up do coin;

## 2.5 ENTITIES

Responsável pelas entidades/inimigos.

```
#include "game.h"

ALLEGRO_TIMER *entities_timer = NULL;

Entity entities[NUM_ENTITIES];

Entity enemies[NUM_ENEMIES];

ALLEGRO_BITMAP *img_red_guy = NULL;
ALLEGRO_BITMAP *img_explosion_normal = NULL;

void entities_load(void);

void entities_reset(void);

int entities_create(Entity entity, int x, int y);

void entities_create_random_enemy(void);

void entities_update(void);

Entity *entities_get_entities(void);
```

### Armazena:

- Timer próprio.
- Array de entidades.
- Array de inimigos.

### Funções:

- void entities\_load(void); carrega as entidades, os inimigos e as imagens.

- void entities\_reset(void); reinicia todas as entidades.
- int entities\_create(Entity entity, int x, int y); cria uma entidade específica nas coordenadas específicas.
- void entities\_create\_random\_enemy(void); cria um inimigo aleatório.
- void entities\_update(void); atualiza todos os inimigos, testa colisões.
- Entity \*entities\_get\_entities(void); retorna o array de entidades.

## 2.6 MOTION

Responsável por atualizar os objetos do jogo e testar colisões.

```
#include "game.h"

ALLEGRO_TIMER *motion_timer = NULL;

ALLEGRO_BITMAP *explosion = NULL;

void motion_update(Movable_Object *object);

ObType motion_collision(Movable_Object *object);
```

**Armazena:**

- Timer próprio
- Imagem de explosão.

**Funções:**

- void motion\_update(Movable\_Object \*object); recebe um objeto e atualiza suas coordenadas, bounding box e animações, por fim, o desenha.
- ObType motion\_collision(Movable\_Object \*object); testa por possíveis colisões com o objeto recebido.

## 2.7 PLANETS

Responsável pelos planetas.



```
#include "game.h"

ALLEGRO_TIMER *planets_timer = NULL;

Planet planets[NUM_PLANETS];

ALLEGRO_BITMAP *images[NUM_IMAGES];

void planets_load(void);

void planets_reset(void);

void planets_create_random_planet(void);

void planets_update(void);

Planet *planets_get_planets(void);
```

#### Armazena:

- Timer próprio
- Array de planetas
- Array de imagens de planetas.

#### Funções:

- void planets\_load(void); carrega as imagens e inicia os planetas.
- void planets\_reset(void); reinicia os planetas.
- void planets\_create\_random\_planet(void); cria um planeta aleatório em coordenadas aleatórias.
- void planets\_update(void); atualiza todos os planetas e testa por colisões.
- Planet \*planets\_get\_planets(void); retorna a array de planetas.

## 2.8 SHIP

Responsável pela nave do jogo.

```

#include "game.h"

ALLEGRO_TIMER *ship_timer = NULL;

ALLEGRO_BITMAP *img_ship_default = NULL;
ALLEGRO_BITMAP *img_ship_up = NULL;
ALLEGRO_BITMAP *img_ship_down = NULL;
ALLEGRO_BITMAP *img_shot_special = NULL;
ALLEGRO_BITMAP *img_shot_charging_sheet = NULL;
ALLEGRO_BITMAP *img_shot_charged = NULL;
ALLEGRO_BITMAP *img_shot_default = NULL;
ALLEGRO_BITMAP *img_shot_flash = NULL;

ALLEGRO_SAMPLE *shot_sound_default = NULL;

Ship ship;

void ship_load(void);

void ship_reset(void);

void ship_update(void);

void ship_fire_shot(void);

Ship *ship_get_ship(void);

```

#### Armazena:

- Timer próprio
- Imagens da nave, dos tiros e dos efeitos especiais dos tiros.

#### Funções:

- void ship\_load(void); carrega a nave, as imagens e reinicia o timer.
- void ship\_reset(void); reinicia a nave e o timer.
- void ship\_update(void); atualiza a nave, testa por colisões e renderiza efeitos de carregamento de tiro especial.
- void ship\_fire\_shot(void); dispara um tiro que pode ou não ser carregado e especial.
- Ship \*ship\_get\_ship(void); retorna um ponteiro para a nave.

## 2.9 SHOTS

Responsável pelos tiros da nave.

```
#include "game.h"

ALLEGRO_TIMER *shot_timer = NULL;

/* Images */
ALLEGRO_BITMAP *img_shot_hit_sheet = NULL;
ALLEGRO_BITMAP *img_shot_hit_charged_sheet = NULL;

/* Sounds */
Shot shots[NUM_SHOTS];

void shots_load(void);

void shots_reset(void);

void shots_update(void);

Shot *shots_get_shots(void);
```

**Armazena:**

- Timer próprio
- Imagens de tiro normal e carregado.
- Array de tiros.

**Funções:**

- void shots\_load(void); carrega as imagens e os tiros.
- void shots\_reset(void); reinicia os tiros.
- void shots\_update(void); atualiza os tiros, testa por colisões e os renderiza.
- Shot \*shots\_get\_shots(void); retorna o array de tiros.

## 2.10 STATS

Responsável por renderizar os status do jogo, tal como pontuação atual e máxima, moedas, barra de energia e pontos de vida.

```
#include "game.h"

/* Score and coins */
int score = 0;
int record = 0;
int coins = 0;

/* Stats */
ALLEGRO_BITMAP *img_heart_full = NULL;
ALLEGRO_BITMAP *img_heart_empty = NULL;
ALLEGRO_BITMAP *img_flipping_coin_sheet = NULL;
ALLEGRO_BITMAP *img_bar_charging_sheet = NULL;
ALLEGRO_BITMAP *img_coin = NULL;

void stats_load(void);

void stats_update(void);
```

#### Armazena:

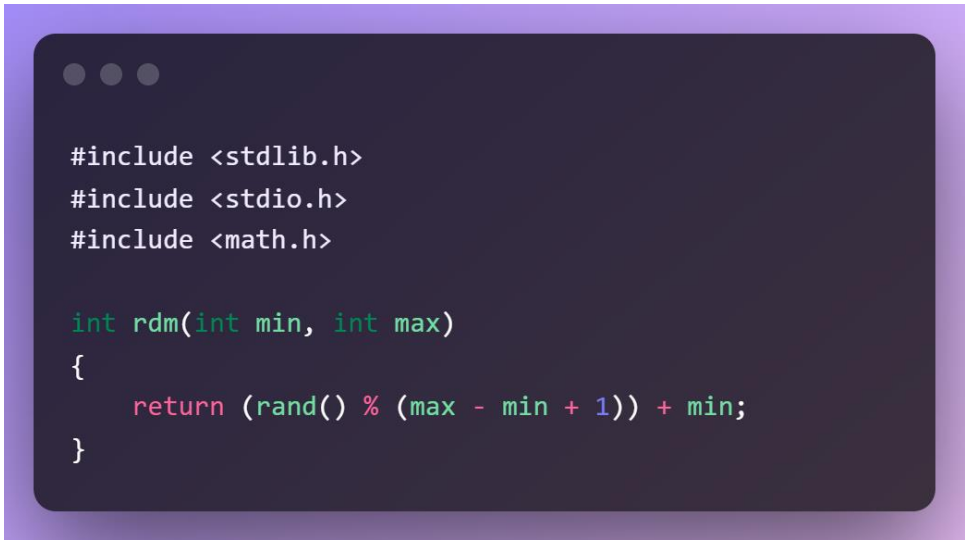
- Variável de score.
- Variável de record.
- Variável de coins.

#### Funções:

- void stats\_load(void); carrega as imagens e o valor do record.
- void stats\_update(void); atualiza e renderiza os stats na tela.

## 2.11 UTILS

Funções de utilidade geral que auxiliam no código.



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

int rdm(int min, int max)
{
    return (rand() % (max - min + 1)) + min;
}
```

### Funções:

- `int rdm(int min, int max);` retorna um número entre o range especificado.