# Relatório Projeto Agar.io

Anderson Caporale Anan

2019/2

Este projeto é baseado no jogo já existente “Agar.io”. Em que o objetivo do jogo é se manter vivo pela maior quantidade de tempo que conseguir e eliminar inimigos menores, evitando os explosivos, para aumentar a pontuação. Sempre que colidir com um inimigo menor, o jogador aumenta seu tamanho. Mas ao colidir com inimigos maiores, o jogador morre e o jogo acaba.

Há, ao total, 300 inimigos em campo que são divididos em 3 tipos: Os normais, que não aplicam nenhum efeito ao contato; os Envenenados, que em vez de aumentar o tamanho do jogador, diminui e não é possível comer outros inimigos por 5 segundos; e os Explosivos, que independente do tamanho, eliminam outros inimigos e se explodem ao colidirem.

Apenas o menu é controlado através do Mouse, com as opções “Novo Jogo”, “Carregar Jogo” e “Sair”. O jogo é controlado pelos direcionais do teclado. Além disso, há outros botões especiais, como: Esc para voltar ao Menu, P para pausar o jogo e S para salvar o momento atual do jogo (precisa estar pausado para salvar).

## **Estruturas utilizadas**:

Estrutura JOGADOR possui campos para a posição x, posição y, o raio, a velocidade, o estado (0 = normal e 1 = envenenado), os pontos e uma estrutura COR para representar a cor do jogador. É utilizada especificamente para a peça do jogador.

```
typedef struct jogador {
   int x;
   int y;
   int raio;
   int velocidade;
   int estado;
   int pontos;
   COR cor;
} JOGADOR;
```

Estrutura COR possui 3 inteiros, baseado na classificação RGB. Cada inteiro é para representar uma cor.

```
typedef struct cor {
   int red;
   int green;
   int blue;
} COR;
```

Estrutura LINHA é utilizada para as linhas de fundo do jogo, possui a informação do primeiro ponto (x1 e y1) e do segundo ponto (x2 e y2).

```
typedef struct linhas_background {
   int x1;
   int y1;
   int x2;
   int y2;
} LINHA;
```

Estrutura PONTOS é usada para representar os pontos. Possui uma estrutura time_t para o inicio e time_t para o fim, além de um inteiro para representar se está parado ou andando.

```
typedef struct tempo {
   time_t inicio;
   time_t fim;
   int andando;
} TEMPO;
```

Estrutura FUNDO serve para armazenar a posição x e a posição y da imagem de fundo. Possui tipos float devido à tela se movimentar em números pequenos.

```
typedef struct fundo {
   float x;
   float y;
} FUNDO;
```

Estrutura INIMIGOS é utilizada para os inimigos, possui uma posição x, uma posição y, um raio, um inteiro que representa a direção dos inimigos direcionados (0 = Norte; 1=Sul; 2=Leste; 3=Oeste; 4=Nordeste; 5=Sudeste; 6=Sudoeste; 7=Noroeste), um inteiro que representa o estado (0 = morto e 1 = vivo), um tipo (0 = estáticas, 1 = envenenados e 2 = explosivas) e uma estrutura COR para representar a cor (de acordo com o tipo).

```
typedef struct inimigos {
   int x;
   int y;
   int raio;
   int direcao;
   int estado;
   int tipo;
   COR cor;
} INIMIGOS;
```

## Funções Implementadas:

**void iniciarJogador (JOGADOR*); -** Recebe um ponteiro do tipo Jogador. Serve para iniciar os dados do jogador.

**void iniciarInimigos (INIMIGOS[]); -** Recebe um vetor do tipo INIMIGOS. Serve para iniciar os dados dos inimigos.

**void iniciarBackground (LINHA[], LINHA []); -** Recebe dois vetores do tipo LINHA. Serve para iniciar as linhas do fundo.

**void iniciarFundo (FUNDO*); -** Recebe um ponteiro do tipo FUNDO. Serve para iniciar a posição da imagem de fundo.

**void desenharInimigos (INIMIGOS[]); -** Recebe um vetor do tipo INIMIGOS. Serve para desenhar os inimigos.

**void desenharFundo(LINHA[], LINHA []);** - Recebe dois vetores do tipo LINHA. Serve para desenhar as linhas do fundo.

**void moverBaixo (JOGADOR , INIMIGOS [], bool[], LINHA[], FUNDO *);** - Recebe um JOGADOR, um vetor do tipo INIMIGOS, um vetor bool, um vetor LINHA e um ponteiro FUNDO. Serve para mover o jogador para baixo. (Movimenta o resto para cima)

**void moverCima (JOGADOR , INIMIGOS [], bool[], LINHA[], FUNDO *); -** Recebe um JOGADOR, um vetor do tipo INIMIGOS, um vetor bool, um vetor LINHA e um ponteiro FUNDO. Serve para mover o jogador para cima. (Movimenta o resto para baixo)

**void moverDir(JOGADOR , INIMIGOS [], bool[], LINHA[], FUNDO *);** - Recebe um JOGADOR, um vetor do tipo INIMIGOS, um vetor bool, um vetor LINHA e um ponteiro FUNDO. Serve para mover o jogador para direita. (Movimenta o resto para esquerda)

**void moverEsq (JOGADOR , INIMIGOS [], bool[], LINHA[], FUNDO *);** - Recebe um JOGADOR, um vetor do tipo INIMIGOS, um vetor bool, um vetor LINHA e um ponteiro FUNDO. Serve para mover o jogador para esquerda. (Movimenta o resto para direita)

**void inimigosLinear (INIMIGOS []); -** Recebe um vetor do tipo INIMIGOS. Serve para movimentar os inimigos Direcionados.

**void inimigosPerseguidores (INIMIGOS [], JOGADOR);** - Recebe um vetor do tipo INIMIGOS e uma variável do tipo JOGADOR. Serve para movimentar os inimigos Perseguidores. A posição do inimigo é baseada na posição do jogador.

**void inimigosAleatorios(INIMIGOS []); -** Recebe um vetor do tipo INIMIGOS. Serve para movimentar os inimigos que se movem em direções completamente aleatórias.

**void respawnInimigos (INIMIGOS [], JOGADOR , LINHA [], LINHA []);** - Recebe um vetor do tipo INIMIGOS, uma variável tipo JOGADOR, e 2 vetores do tipo LINHA. Serve para reviver os inimigos que foram mortos. Eles só reaparecem caso não estejam no campo de visão do jogador. E apenas dentro dos limites do mapa.

**int colisaoJogador (JOGADOR *, INIMIGOS [], clock_t [], ALLEGRO_SAMPLE * , ALLEGRO_SAMPLE *);** – Recebe um ponteiro do tipo JOGADOR, um vetor do tipo INIMIGOS, um vetor do tipo clock_t, e 2 ponteiros do tipo ALLEGRO_SAMPLE (sons). Serve para detectar colisão do jogador com os inimigos. Retorna -1 se houve colisão que mata o jogador.

**void colisaoInimgos (INIMIGOS []); -** Recebe um vetor do tipo INIMIGOS. Serve para detectar colisões entre os inimigos.

**void limitesInimigo (INIMIGOS [], LINHA [], LINHA []); -** Recebe um vetor do tipo INIMIGOS e 2 vetores do tipo LINHA. Serve para não deixar os inimigos passarem das bordas do mapa.

**void limitesJogador (JOGADOR, LINHA [], LINHA []); -** Recebe uma variável do tipo JOGADOR e 2 vetores do tipo LINHA. Serve para não deixar o jogador passar das bordas do mapa.

**void inicio (TEMPO *);** - Recebe um ponteiro do tipo TEMPO. Serve para marcar o inicio do tempo dos pontos.

**void para (TEMPO *); -** Recebe um ponteiro do tipo TEMPO. Serve para marcar quando se para o tempo dos pontos.

**void continua (TEMPO *); -** Recebe um ponteiro do tipo TEMPO. Serve para marcar quando continua a contar o tempo dos pontos.

**int tempo (TEMPO); -** Recebe uma variável do tipo TEMPO. Serve retornar quanto tempo passou desde o inicio do jogo. A cada 1 segundo, o jogador ganha mais 5 pontos.

**void salvarJogo(JOGADOR , INIMIGOS [], LINHA [], LINHA [], int, TEMPO, FUNDO , int );** - Recebe uma variável do tipo JOGADOR, um vetor do tipo INIMIGOS, 2 vetores do tipo LINHA, uma variável do tipo int, uma variável do tipo PONTOS, uma variável do tipo TEMPO e um inteiro. Serve para salvar as informações dos parâmetros recebidos em um arquivo binário. Para poder continuar o jogo de onde parou se salvar antes de fechar.

**void carregarJogo(JOGADOR *, INIMIGOS [], LINHA [], LINHA [], int *, TEMPO*, FUNDO *, int*); -** Recebe um ponteiro do tipo JOGADOR, um vetor do tipo INIMIGOS, 2 vetores do tipo LINHA, um ponteiro do tipo int, um ponteiro do tipo TEMPO, um ponteiro do tipo FUNDO e um ponteiro inteiro. Serve para ler um arquivo binário e salvar as informações nos parâmetros da função. Para continuar o jogo de onde parou.

**void opcoesMenu (int*, int*, int*, int*, int*, int, int, ALLEGRO_EVENT_QUEUE*, ALLEGRO_EVENT, ALLEGRO_SAMPLE *, ALLEGRO_FONT *); -** Recebe 5 ponteiros e 2 variáveis do tipo int, uma variável do tipo ALLEGRO_EVENT e 3 ponteiros dos tipos ALLEGRO_EVENT_QUEUE, ALLEGRO_SAMPLE e ALLEGRO_FONT. Serve para controlar os cliques e opções do menu.

**void opcoesMorte (int*, int*, int*, int*, int, int, ALLEGRO_EVENT_QUEUE *, ALLEGRO_EVENT, ALLEGRO_SAMPLE *, ALLEGRO_FONT *); -** Recebe 4 ponteiros e 2 variáveis do tipo int, uma variável do tipo ALLEGRO_EVENT e 3 ponteiros dos tipos ALLEGRO_EVENT_QUEUE, ALLEGRO_SAMPLE e ALLEGRO_FONT. Serve para controlar os cliques e opções da tela de morte.