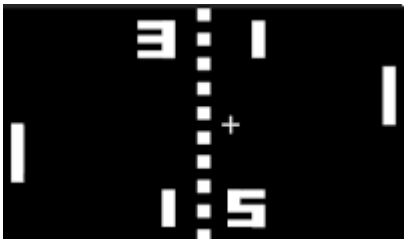
Jogo gráfico



O objetivo deste projeto é construir um jogo monousuário com interface gráfica 2D. O aluno deve escolher um destes jogos [1]:

Clássicos:

- Battle City
- Super Mario Bros
- Maziacs
- Boulder Dash
- Donkey Kong

Recentes:

- Ballz
- Okay?
- Geometry Dash
- ZigZag (fazer em 2D)

Você pode propor outros jogos ao professor para análise.

Requisitos

Requisitos do jogo:

- Executar no Linux, em uma janela gráfica
- Interagir com o usuário através do teclado e/ou mouse
- Usar imagens coloridas (*sprites*), não somente formas geométricas (pode usar figuras prontas obtidas na Internet)
- Tocar sons nos principais eventos (pode usar sons prontos da Internet)
- Suportar ao menos **três níveis** com mapas, ações e/ou objetos diferentes
- Calcular a pontuação do jogador e manter um *score* persistente (salvo em disco) das melhores pontuações obtidas, apresentado ao usuário ao finalizar o jogo
- As teclas h ou F1 abrem uma tela de ajuda com as instruções do jogo, nome do autor e outras informações
- Ter um *Easter Egg* ou um *cheat code* (como o Código Konami)
- Executar a 60 FPS (*frames per second*)

Requisitos do código:

- Ser desenvolvido em C padrão C99 (`-std=c99`, `-std=gnu99` ou `-std=gnu11`)
- Usar a biblioteca gráfica **Allegro 5** [2]
- Separar o código em vários arquivos `.c` e `.h` separados, respeitando as regras de organização de código
- Usar alocação de memória dinâmica
- Usar estruturas

- Compilar com o flag `-Wall` sem erros nem avisos
- Usar o sistema Make para compilação, com ao menos os seguintes alvos:
  - `all`: compila e gera o executável
  - `clean`: remove os arquivos temporários (mantém o executável)
  - `purge`: remove tudo, deixando somente os fontes
- Todos os arquivos que não forem código-fonte (imagens, sons, fontes, mapas, ...) devem ser colocados em um subdiretório `resources/` dentro do diretório do código.

Critérios de avaliação

- cumprimento dos requisitos acima
- fidelidade ao jogo original
- qualidade do código (estrutura, clareza, comentários, endentação)

A biblioteca Allegro

A biblioteca gráfica Allegro permite a manipulação de gráficos simples e áudio, sendo bem adaptada para a construção de jogos 2D. É uma biblioteca mais simples e limitada que a famosa biblioteca SDL, sendo mais adequada para iniciantes e para projetos menores.

Algumas de suas características:

- multiplataforma: Linux, Windows, MacOS, Android, iOS
- pode ser usada em C e outras linguagens
- usa aceleração gráfica (através de OpenGL ou DirectX)
- manipulação de áudio e vídeo
- leitura de mouse, teclado e joystick

Instalação da biblioteca Allegro 5 em Linux (Ubuntu, Mint, Debian):

```
sudo apt-get install liballegro5-dev
```

Uso:

- Página principal
- Referência das funções
- Tutorial de jogo em Allegro
- Galeria de jogos

Bibliotecas gráficas como a Allegro e SDL operam usando um esquema interno de **buffer duplo**, ou seja, dois *buffers* gráficos distintos que são usados em conjunto:

- buffer$_1$: *buffer* onde as operações gráficas são aplicadas
- buffer$_2$: *buffer* cujo conteúdo é mostrado na tela

Esses *buffers* devem ser periodicamente alternados pelo programador (pseudocódigo):

```
fill_color (black) ;  // operações feitas no buffer 1, não são visíveis
draw (...) ;          // na tela neste momento.
draw (...) ;
...
flip_buffers () ;     // troca buffer 1 com 2, mostrando o novo conteúdo
```

As técnicas des buffer duplo e page flipping são implementadas nativamente na biblioteca Allegro, fornecendo

imagens mais estáveis e melhor desempenho gráfico.

Estrutura básica de um jogo

A maioria dos jogos segue uma estrutura de código similar. O funcionamento do jogo segue uma "máquina de estados" que define o momento atual do jogo e um laço principal que implementa o jogo em si. Ambos são explicados brevemente nesta seção.

Estado das entidades

Cada entidade do jogo (jogador, bola, etc) tem um conjunto de atributos que a representam, que normalmente é implementado em um `struct`. Por exemplo, no jogo Pong a bola poderia ser representada pela seguinte estrutura:

```
// estrutura de uma bola
typedef struct
{
  short x, y ;     // posição atual
  short sx, sy ;   // velocidade atual
  short w, h ;     // dimensões
  color_t color ;  // cor atual
}
ball_t ;

// cria uma bola
ball_t ball ;
```

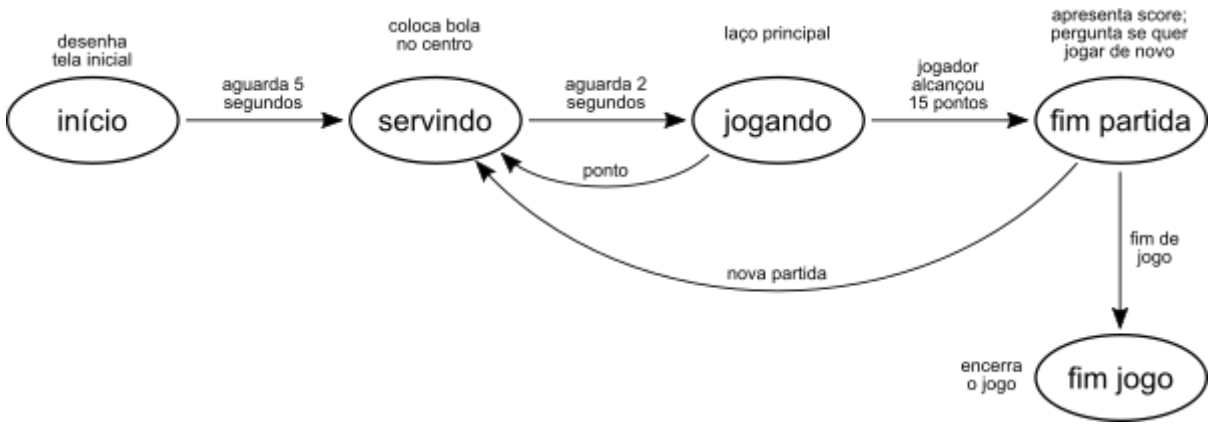
Estruturas como essa devem ser construídas para cada entidade ativa do jogo.

Máquina de estados

O funcionamento do jogo é gerenciado através de uma *máquina de estados*. Cada estado do jogo define o que deve ser feito naquele momento, as transições possíveis entre estados e os eventos que levam de um estado a outro. Por exemplo, a máquina de estados do jogo Pong é composta dos seguintes estados:

- **inicio**: desenha a tela inicial do jogo e aguarda 5 segundos.
- **servindo**: põe a bola no centro e aguarda 2 segundos.
- **jogando**: implementa o jogo propriamente dito (laço principal).
- **fim partida**: um dos jogadores atingiu a pontuação máxima.
- **fim jogo**: o jogo encerra.

O relacionamento entre esses estados pode ser representado graficamente:



Essa máquina de estados pode ser facilmente implementada em C:

```
// estados possíveis do jogo Pong
enum {INICIO, SERVINDO, JOGANDO, FIMPART, FIMJOGO} state ;

// programa principal do jogo
int main ()
{
  state = INICIO ;
  for (;;)
    switch (state)
    {
      case INICIO  : state_init () ;  break ;
      case SERVINDO: state_serve () ; break ;
      case JOGANDO : state_play () ;  break ;
      case FIMPART : state_over () ;  break ;
      case FIMJOGO : state_close () ; break ;
      default: break ;
    }
}
```

Dessa forma, as funções `state_init()` e demais são usadas para implementar o comportamento de cada etapa do jogo. A variável `state` pode ser modificada dentro dessas funções para trocar de estado conforme necessário.

O laço principal

Durante o jogo propriamente dito, o programa deve continuamente ler as entradas de dados (teclado, mouse, ...), atualizar o estado dos jogadores e demais entidades do jogo de acordo com essas entradas e desenhar a tela. Esse comportamento é chamado o **laço principal** do jogo.

O pseudocódigo a seguir dá um exemplo do laço principal do jogo Pong (simplificado):

```
// define estado inicial das entidades
inicia_jogadores ()
inicia_bola ()

// laço principal
faça
  ler_entrada (teclado, mouse, ...)
  atualiza_estado_jogadores ()
  atualiza_estado_bola ()
```

```
  desenha_tela ()
até fim da partida
```

Esse ciclo deve ser repetido rapidamente (dezenas de vezes por segundo) para dar a impressão de fluidez do jogo.

O tutorial de jogo da biblioteca Allegro explica com detalhes o funcionamento do laço principal.

Controle da taxa de quadros

A tela do jogo deve ser atualizada com velocidade suficiente para dar fluidez ao jogo. De preferência, a taxa de atualização da tela (taxa de quadros por segundo ou *frame rate*) deve ser constante, para que a animação seja consistente.

Em um jogo simples, a quantidade de cálculo a efetuar antes de cada atualização de estado e de tela é geralmente pequena, se comparada à capacidade de processamento do computador. Por isso, é interessante **pausar a execução** por alguns milissegundos a cada ciclo, para garantir uma taxa de quadros constante (e também diminuir o consumo de CPU).

Uma forma de controlar a taxa de quadros é através de pausas (pseudocódigo):

```
// taxa de quadros por segundo
frame_rate = 60

// duração de cada quadro, em ms
t_quadro = 1000 / frame_rate

// laço principal do jogo
repita

   t_inicio = relogio_ms ()

  // processamento do jogo
  ...

  // espera um tempo para completar o quadro
  t_final = relogio_ms ()
  t_pausa = t_quadro - (t_final - t_inicio)
  pausa_ms (t_pausa)

até o fim da partida
```

Outra forma, que é usada frequentemente na Allegro 5, é a **programação por eventos**. Neste caso específico, programa-se um temporizador (*timer*) para gerar um evento a cada quadro; quando o evento ocorre, o processamento do jogo é feito:

```
// taxa de quadros por segundo
frame_rate = 60

// duração de cada quadro, em ms
t_quadro = 1000 / frame_rate

// define timer (um evento a cada X ms)
define_timer (t_quadro)
```

```
// laço principal do jogo
repita

   // espera o próximo evento do timer
   espera_evento (timer)

  // processamento do jogo
  ...

até o fim da partida
```

Links

Desenvolvimento de jogos:

- CS50's Introduction to Game Development - Harvard (assistir pelo menos as duas primeiras aulas)
- Tutorial de jogo da biblioteca Allegro
- Beginner's Guide to Roguelikes (ler ao menos a parte 1)
- Game Loop
- Canal Jogos & Programação no Youtube

Sprites e sons:

- Open Game Art
- JSFXR sound generator

[1)]
Diversos outros jogos foram desconsiderados porque existem muitas implementações prontas usando C e Allegro.
[2)]
Não é a mais poderosa, mas tem um bom compromisso entre funcionalidades e facilidade de uso.