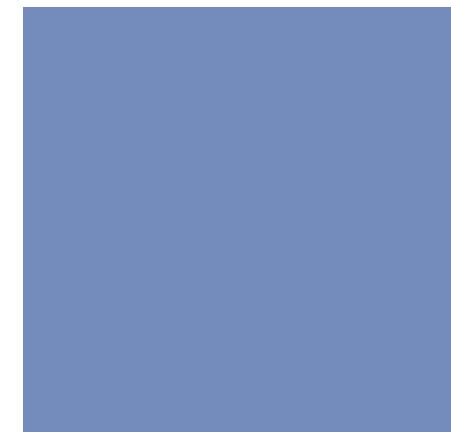
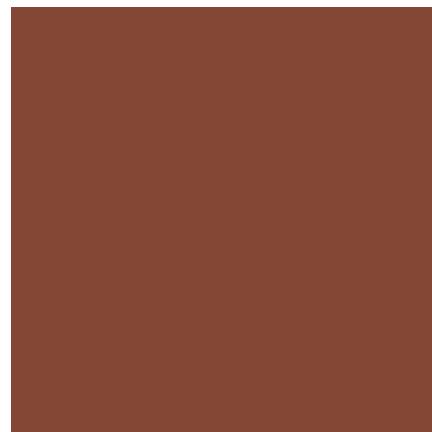
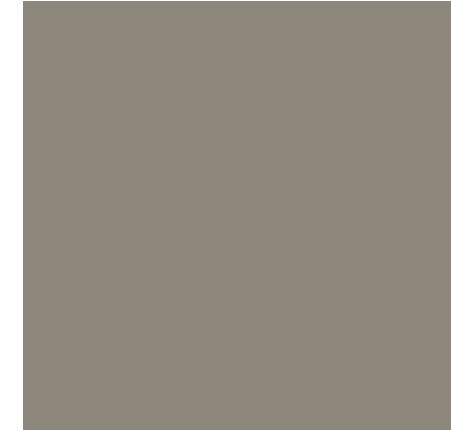
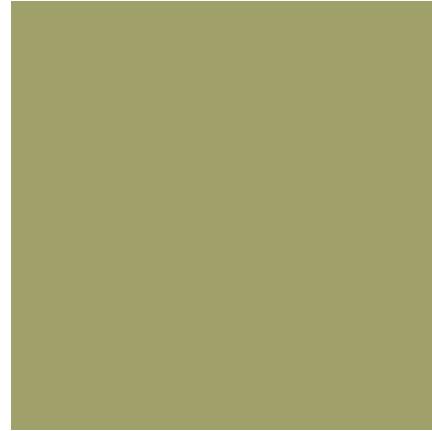




SFML

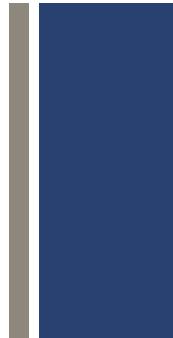
... para o TP3



INF110 – Programação I

Prof. André Gustavo
DPI/UFV – 2025/1

UFV



- Simple and Fast Multimedia Library
- Biblioteca para desenvolvimento de jogos 2D
- Tutorial baseado em:
 - <https://www.sfml-dev.org/tutorials/2.6>
- Os programas usados no tutorial estão disponíveis no PVANet



Instalação

■ Baixe a versão compatível

The screenshot shows the SFML website's header. It features a green navigation bar with the SFML logo on the left. The 'Download' link is highlighted with a red circle. Other menu items include 'Learn', 'Community', and 'Development'. Below the bar, there are language links for 'Français' and 'English', and a 'Donate' button.

Download SFML 2.6.1

On Windows, choosing 32 or 64-bit libraries should be based on which platform you want to compile for, not which OS you have. Indeed, you can perfectly compile and run a 32-bit program on a 64-bit Windows. So you'll most likely want to target 32-bit platforms, to have the largest possible audience. Choose 64-bit packages only if you have good reasons.

Unless you are using a newer version of Visual Studio, the compiler versions have to match 100%!

Here are links to the specific MinGW compiler versions used to build the provided packages:

WinLibs MSVCRT 13.1.0 (32-bit), WinLibs MSVCRT 13.1.0 (64-bit)

| | | | |
|---------------------------------|------------------------------------|---------------------------------|------------------------------------|
| Visual C++ 17 (2022) - 32-bit | Download 20.3 MB | Visual C++ 17 (2022) - 64-bit | Download 21.8 MB |
| Visual C++ 16 (2019) - 32-bit | Download 19.3 MB | Visual C++ 16 (2019) - 64-bit | Download 20.7 MB |
| Visual C++ 15 (2017) - 32-bit | Download 17.6 MB | Visual C++ 15 (2017) - 64-bit | Download 19.4 MB |
| GCC 13.1.0 MinGW (DW2) - 32-bit | Download 17.9 MB | GCC 13.1.0 MinGW (SEH) - 64-bit | Download 18.9 MB |

On Linux, if you have a 64-bit OS then you have the 64-bit toolchain installed by default. Compiling for 32-bit is possible, but you have to install specific packages and/or use specific compiler options to do so. So downloading the 64-bit libraries is the easiest solution if you're on a 64-bit Linux. If you require a 32-bit build of SFML you'll have to [build it yourself](#).

It's recommended to use the SFML version from your package manager (if recent enough) or build from source to prevent incompatibilities.

| | | |
|-------|--------------|------------------------------------|
| Linux | GCC - 64-bit | Download 8.72 MB |
|-------|--------------|------------------------------------|

| | | |
|-------|--|------------------------------------|
| macOS | Clang - 64-bit (macOS 10.15+, compatible with C++11 and libc++) | Download 5.07 MB |
| | Clang - ARM64 (macOS 11+) | Download 5.02 MB |
| | macOS libraries are only compatible with 64-bit and ARM64 (M1 / M2) systems. | |



Instalação

- Instale conforme instruções

The screenshot shows the SFML website's main navigation bar with a green background. The 'Learn' menu item is highlighted with a red circle. Below the navigation bar, the word 'Learn' is centered in a large, bold, black font. There are four main buttons in light green boxes: 'Tutorials' (with a school icon), 'API Documentation' (with a book icon), 'FAQ' (with a lightbulb icon), and 'License' (with a person icon). The 'Tutorials' button is circled in red.

SFML

Home » Learn

Learn Download Community Development

Français Donate

Learn

Tutorials
Learn how to use SFML

API Documentation
Reference

FAQ
Frequently Asked Questions

License
zlib/png license

The screenshot shows a sub-page under the 'Learn' section, specifically '2.6 Tutorials'. The 'Learn' and 'Download' links are visible in the top navigation bar. The page title is 'Tutorials for SFML 2.6'. A red circle highlights the 'Getting started' heading, which is followed by a list of compilation methods:

SFML

Home » Learn » 2.6 Tutorials

Learn Download

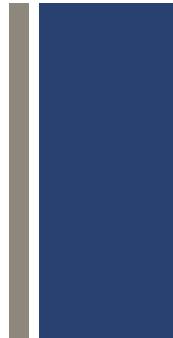
Tutorials for SFML 2.6

Getting started

- SFML with the CMake Project Template
- SFML and Visual Studio
- SFML and Code::Blocks (MinGW)
- SFML and Linux
- SFML and Xcode (macOS)
- Compiling SFML with CMake



Compilação



- SFML contém 5 módulos:
 - system, window, graphics, network e audio
- Existe uma biblioteca para cada um deles
- Para gerar o executável, é preciso linkar as bibliotecas usadas



Compilação

- SFML contém 5 módulos:
 - system, window, graphics, network e audio
- Existe uma biblioteca para cada um deles
- Para gerar o executável, é preciso linkar as bibliotecas usadas
- Para isso, adicione "-lsfml-xxx" na linha de comando, p.ex.

```
g++ prog.cpp -lsfml-graphics -lsfml-window -lsfml-system
```



Compilação

- SFML contém 5 módulos:
 - system, window, graphics, network e audio
- Existe uma biblioteca para cada um deles
- Para gerar o executável, é preciso linkar as bibliotecas usadas
- Para isso, adicione "-lsfml-xxx" na linha de comando, p.ex.

```
g++ prog.cpp -lsfml-graphics -lsfml-window -lsfml-system
```

- Ou configure a IDE conforme instruções do site



Teste de instalação

- Instale a SFML
- Compile o programa **0-teste.cpp**

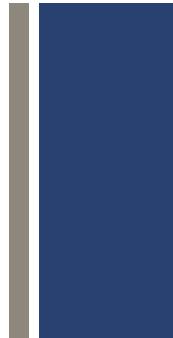
```
g++ 0-teste.cpp -lsfml-graphics -lsfml-window -lsfml-system
```

- Execute o programa e deverá obter uma janela como a abaixo





Window



- Janelas na SFML são definidas pela classe `sf::Window`
- Quando uma janela é criada, ela é aberta:

```
#include <SFML/Window.hpp>

int main()
{
    sf::Window window(sf::VideoMode(800, 600), "My window");

    ...

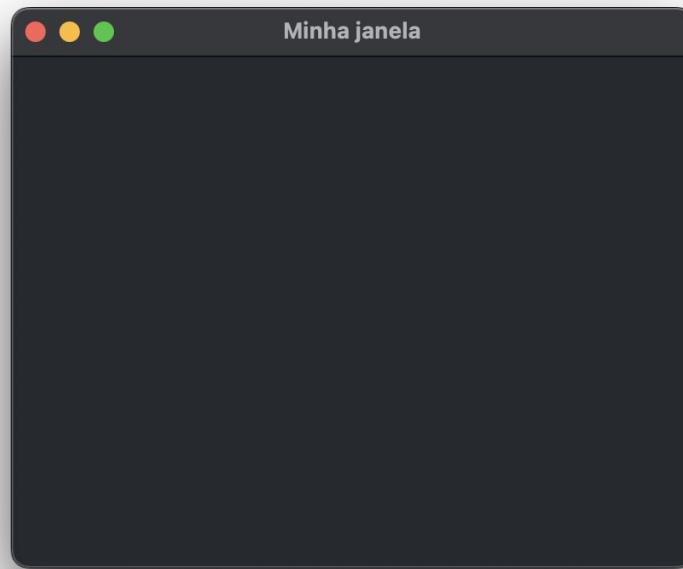
    return 0;
}
```

- Nesse exemplo, 800x600 é o tamanho e "My window", o título



Window – 1-window.cpp

- Compile e execute o programa **1-window.cpp**
- Ele abre uma janela, que fica aberta até ser fechada pelo usuário





Window – 1-window.cpp



```
#include <SFML/Window.hpp>

int main()
{
    sf::Window window(sf::VideoMode(800, 600), "My window");
    // run the program as long as the window is open
    while (window.isOpen())
    {
        // check all the window's events that were triggered since the last iteration of the loop
        sf::Event event;
        while (window.pollEvent(event))
        {
            // "close requested" event: we close the window
            if (event.type == sf::Event::Closed)
                window.close();
        }
    }

    return 0;
}
```

Loop principal (todo programa tem isso):
fica atualizando a aplicação até que a janela seja fechada



Window – 1-window.cpp

```
#include <SFML/Window.hpp>

int main()
{
    sf::Window window(sf::VideoMode(800, 600), "My window");
    // run the program as long as the window is open
    while (window.isOpen())
    {
        // check all the window's events that were triggered since the last iteration of the loop
        sf::Event event;
        while (window.pollEvent(event))
        {
            // "close requested" event: we close the window
            if (event.type == sf::Event::Closed)
                window.close();
        }
    }

    return 0;
}
```

Loop principal (todo programa tem isso):
fica atualizando a aplicação até que a janela seja fechada

Tratamento de eventos: (1^a coisa a se fazer)
verificar eventos pendentes (fechou janela?
pressionou tecla? moveu o mouse? ...) e tratar cada um deles



Window – 1-window.cpp

```
#include <SFML/Window.hpp>

int main()
{
    sf::Window window(sf::VideoMode(800, 600), "My window");
    // run the program as long as the window is open
    while (window.isOpen())
    {
        // check all the window's events that were triggered since the last iteration of the loop
        sf::Event event;
        while (window.pollEvent(event))
        {
            // "close requested" event: we close the window
            if (event.type == sf::Event::Closed)
                window.close();
        }
    }
    return 0;
}
```

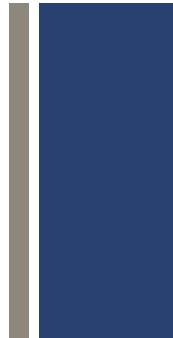
Loop principal (todo programa tem isso): fica atualizando a aplicação até que a janela seja fechada

Tratamento de eventos: (1^a coisa a se fazer) verificar eventos pendentes (fechou janela? pressionou tecla? moveu o mouse? ...) e tratar cada um deles

Usuário "fechou" a janela: no caso, solicitou fechamento; ela só é fechada mesmo no `window.close()` - pode fazer algo antes, p.ex., salvar conteúdo, exibir mensagem...



Window – 1-window.cpp



Tarefa:

- Escreva uma mensagem logo antes de fechar a janela

```
cout << "Fechou...\n";
```

- Note que a mensagem aparecerá no terminal



Event

- O programa fica em loop, redesenhando a tela e tratando eventos à medida que aparecem
- Eventos incluem:
 - ações do teclado (pressionar tecla, soltar tecla, ...)
 - ações do mouse (clicar, mover, entrar/sair na janela, ...)
 - ações do joystick (movimentar, ...)
 - ações na janela (fechar, redimensionar, perder/ganhar foco...)
 - ...



Event – 2-event.cpp

- Compile e execute o programa **2-event.cpp**
- Ele trata mais um evento: `sf::Event::KeyPressed`
- Este é o evento usado para tomar uma ação exatamente quando uma tecla é pressionada, por exemplo, mover um personagem, dar um salto com espaço, sair da tela com esc...



Event – 2-event.cpp

- Compile e execute o programa **2-event.cpp**
- Ele trata mais um evento: `sf::Event::KeyPressed`
- Este é o evento usado para tomar uma ação exatamente quando uma tecla é pressionada, por exemplo, mover um personagem, dar um salto com espaço, sair da tela com esc...
- Existe também o evento `sf::Event::KeyReleased`, que é acionado quando se solta uma tecla
- O código da tecla pode ser recuperado em `event.key.code`



Event – 2b-event.cpp

- Compile e execute o programa **2b-event.cpp**
- Ele mostra como recuperar o estado das teclas modificadoras (shift, alt, control, ...) quando uma tecla é pressionada

Tarefa:

- Modifique o programa para fechar a janela quando se pressiona Shift + Esc



Draw window – 3-draw.cpp

- Compile e execute o programa **3-draw.cpp**
- Ele cria uma janela de fundo preto





Draw window – 3-draw.cpp

- Parece o mesmo que o 1-window.cpp, mas é diferente
- A janela é `sf::RenderWindow`, não `sf::Window` de antes
- Ela tem tudo o que uma `sf::Window` tem e mais...
- Ela tem várias funcionalidades para desenhar coisas
- Nesse exemplo, temos a `clear`, que limpa a janela na cor escolhida, e a `display`, que de fato mostra a janela



Draw window – 3-draw.cpp

- O ciclo `clear/draw/display` é repetido continuamente
 - `clear`: limpa janela (apaga tudo)
 - `draw`: desenha as coisas (fundo, personagens, textos, etc)
 - `display`: mostra o que foi desenhado
- `clear` e `draw` são feitos em background, num buffer escondido; depois de tudo pronto, `display` copia do buffer para a janela



Draw window – 3-draw.cpp

- O ciclo `clear/draw/display` é a melhor forma de desenhar
- É assim que a tela de um jogo é atualizada!
 - Para mover um personagem, ele é apagado e redesenrado no novo local
 - Coisas que não se movem são apagadas e redesenhadas no mesmo local



Draw window – 3-draw.cpp

- O ciclo `clear/draw/display` é a melhor forma de desenhar
- É assim que a tela de um jogo é atualizada!
 - Para mover um personagem, ele é apagado e redesenrado no novo local
 - Coisas que não se movem são apagadas e redesenhadas no mesmo local
- **Não** tente outras estratégias, como manter coisas do frame anterior, apagar algumas coisas, desenhar uma vez e display múltiplas vezes...



Draw window – 3-draw.cpp

- O ciclo `clear/draw/display` é a melhor forma de desenhar
- É assim que a tela de um jogo é atualizada!
 - Para mover um personagem, ele é apagado e redesenrado no novo local
 - Coisas que não se movem são apagadas e redesenhadas no mesmo local
- **Não** tente outras estratégias, como manter coisas do frame anterior, apagar algumas coisas, desenhar uma vez e display múltiplas vezes...
- Não tenha medo de desenhar 1000 coisas 60 vezes por segundo; isso é pouco comparado com milhões de triângulos que o computador suporta
- Placas gráficas são *projetadas* para esse ciclo: apagar tudo e desenhar tudo novamente em cada iteração do loop principal



Draw shape – 4-shape.cpp

- Compile e execute o programa **4-shape.cpp**
- Ele cria uma janela de fundo preto com um círculo verde





Draw shape – 4-shape.cpp

```
// cria um círculo de raio 50
sf::CircleShape circ(50);

// define a posição absoluta do círculo
circ.setPosition(10, 50);

// define a cor do círculo (verde)
circ.setFillColor(sf::Color(100, 250, 50));

// executa o programa enquanto a janela está aberta
while (window.isOpen())
{
    ...

    // limpa a janela com a cor preta
    window.clear(sf::Color::Black);

    // desenhar tudo aqui...
    // desenha o círculo na janela
    window.draw(circ);

    // termina e desenha o frame corrente
    window.display();
}
```

- shape círculo criado uma vez
- mas apagado e redesenhado em toda iteração

clear: limpa tudo

draw: desenha coisas

display: mostra tudo

loop principal



Draw shape – 4-shape.cpp

■ Tarefas:

- adicionar um círculo amarelo exatamente no centro da janela
- adicionar um retângulo 100 x 50 em cada canto

```
// cria um retângulo de tamanho 50 x 50 (um quadrado)
sf::RectangleShape caixa(sf::Vector2f(50, 50));
```

exemplo do RectangleShape



Draw shape – 4-shape.cpp

■ Tarefas:

- adicionar um círculo amarelo exatamente no centro da janela
- adicionar um retângulo 100 x 50 em cada canto

```
// cria um retângulo de tamanho 50 x 50 (um quadrado)
sf::RectangleShape caixa(sf::Vector2f(50, 50));
```

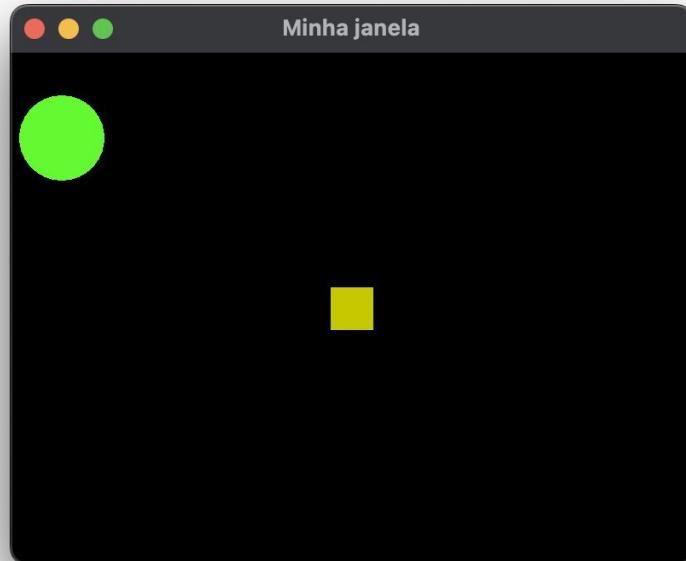
exemplo do RectangleShape

- mudar a cor do círculo aleatoriamente (a cada frame)



Draw shape – 4b-shape.cpp

- Compile e execute o programa **4b-shape.cpp**
- Ele adiciona um quadrado amarelo no centro
- O quadrado pode ser movimentado com as setas esquerda/direita!





Draw shape – 4b-shape.cpp

```
// cria um quadrado de tamanho 50
sf::RectangleShape quad(sf::Vector2f(50, 50));
quad.setFillColor(sf::Color(200, 200, 00));
```

cria um shape quadrado

```
float posx = 375, posy = 275; //posicao do quadrado
```

```
// executa o programa enquanto a janela está aberta
while (window.isOpen())
{
    // verifica todos os eventos que foram acionados na janela desde a última
    sf::Event event;
    while (window.pollEvent(event)) {
        // tecla pressionada
        if (event.type == sf::Event::KeyPressed) {
            if (event.key.code == sf::Keyboard::Left)
                posx -= 10; // left key: move o quadrado para esquerda
            else if (event.key.code == sf::Keyboard::Right)
                posx += 10; // right key: move o quadrado para direita
        }
    }

    // limpa a janela com a cor preta
    window.clear(sf::Color::Black);
```

altera o valor da variável de posição

clear: limpa tudo

```
// reposiciona o quadrado
quad.setPosition(posx, posy);
// desenha o quadrado na janela
window.draw(quad);
```

draw: desenha o quadrado (em nova posição)

```
// termina e desenha o frame corrente
window.display();
```

display: mostra tudo

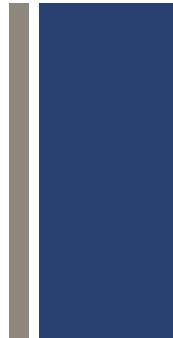


Draw shape – 4b-shape.cpp

- Note que "movimentar" o quadrado consiste em redesenhá-lo em nova posição no ciclo clear/draw/display
- Quando uma seta é pressionada, imediatamente muda-se o valor da variável que indica onde o quadrado é desenhado
- Mas a "movimentação" em si só é feita quando ele é redesenhado (*atraso imperceptível, a janela é redesenhada várias vezes por seg.*)



Draw shape – 4b-shape.cpp

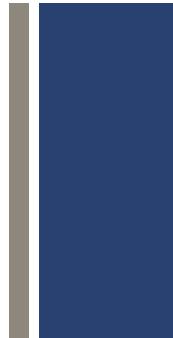


■ Tarefas:

- movimentar o quadrado também para cima e para baixo

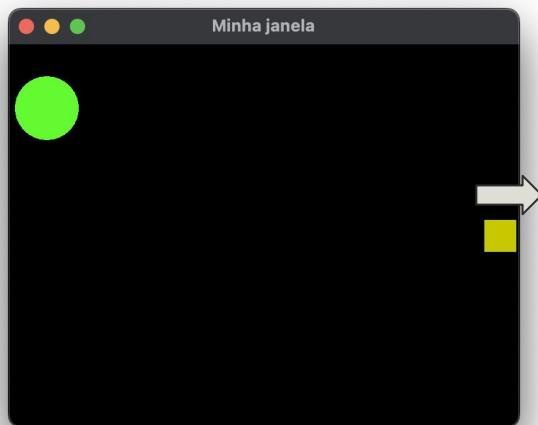


Draw shape – 4b-shape.cpp



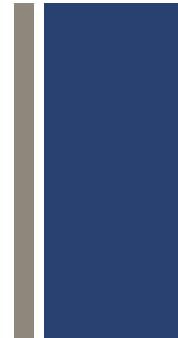
■ Tarefas:

- movimentar o quadrado também para cima e para baixo
- impedir que o quadrado saia da janela



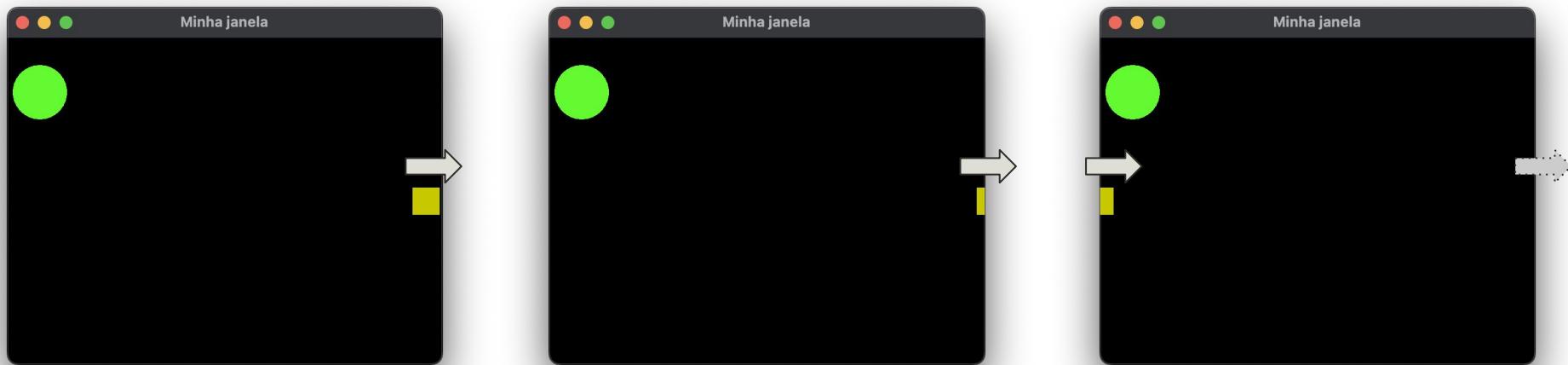


Draw shape – 4b-shape.cpp



■ Tarefas:

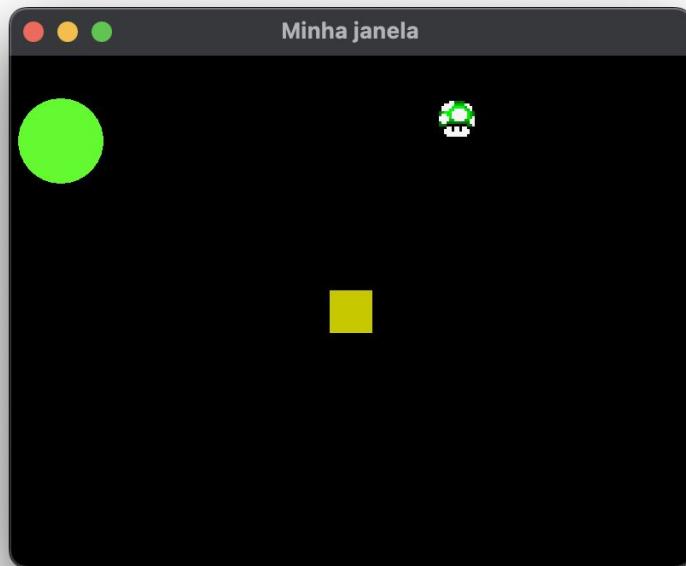
- movimentar o quadrado também para cima e para baixo
- impedir que o quadrado saia da janela
- em vez de impedir que saia da janela, fazê-lo aparecer do outro lado





Draw texture/sprite – 5-sprite.cpp

- Compile e execute o programa **5-sprite.cpp**
- Ele adiciona uma "vida" na janela





Draw texture/sprite – 5-sprite.cpp

```
// cria uma "textura" (uma figura)
sf::Texture texture;
if (!texture.loadFromFile("1up.png"))
{
    std::cout << "Erro lendo imagem 1up.png\n";
    return 0;
}
// cria um sprite (um objeto gráfico) com a figura
sf::Sprite sprite;
sprite.setTexture(texture);
sprite.setPosition(500, 50);
```

cria uma "texture"
com uma imagem

cria um sprite com a
texture (imagem)

- Sprite é uma caixa com uma textura (uma figura)



Rectangular entity

Texture

Sprite!



Draw texture/sprite – 5-sprite.cpp

```
// executa o programa enquanto a janela está aberta
while (window.isOpen())
{
    // limpa a janela com a cor preta
    window.clear(sf::Color::Black);

    // desenhar tudo aqui...

    //reposiciona o quadrado
    quad.setPosition(posx, posy);

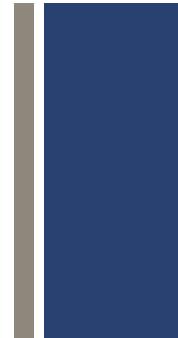
    // desenha o círculo na janela
    window.draw(circ);
    // desenha o quadrado na janela
    window.draw(quad);
    // desenha sprite de vida na janela
    window.draw(sprite);

    // termina e desenha o frame corrente
    window.display();
}
```

sprites são desenhados da mesma forma que shapes



Draw texture/sprite – 5-sprite.cpp



- **Tarefa:**

- acrescentar outro sprite (um personagem qualquer à sua escolha)
- acrescentar uma imagem de fundo



Draw texture/sprite – 5-sprite.cpp

■ Tarefa:

- acrescentar outro sprite (um personagem qualquer à sua escolha)
- acrescentar uma imagem de fundo
- criar um sprite com apenas parte de uma imagem
 - ao ler uma imagem pode-se criar uma textura com parte dela
 - ex.: textura 32x32 píxel que começa na posição (10,10)

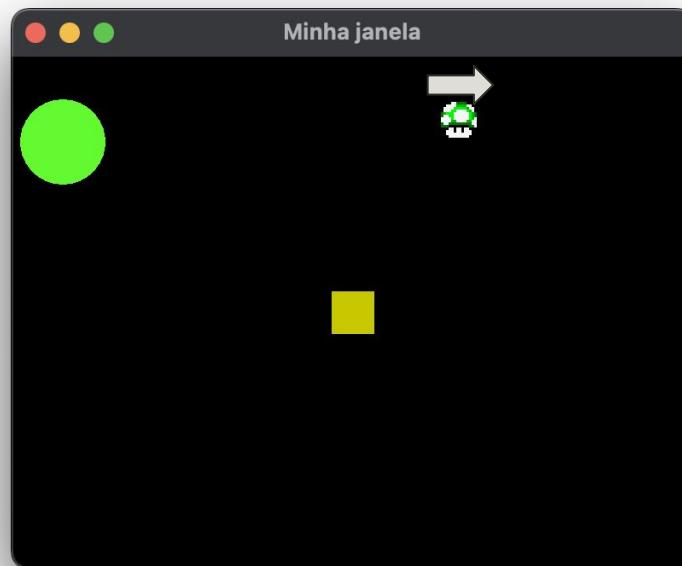
```
// load a 32x32 rectangle that starts at (10, 10)
if (!texture.loadFromFile("image.png", sf::IntRect(10, 10, 32, 32)))
{
    // error...
}
```

obs.: bastante útil pois há imagens disponíveis na web contendo vários personagens, em diferentes posições



Time – 6-time.cpp

- Compile e execute o programa **6-time.cpp**
- Ele movimenta o sprite vida para a direita automaticamente





Time – 6-time.cpp

- O tempo decorrido pode ser medido com a classe `sf::Clock`
- Ela tem apenas duas funções
 - `getElapsedTime`: tempo decorrido desde que o relógio começou
 - `restart`: reinicia o relógio
- No programa, é usado para movimentar o sprite vida a cada 0.1s

```
// cria um relógio para medir o tempo  
sf::Clock clock;
```

```
// Muda a posição do sprite vida a cada 0.1 segundos  
if (clock.getElapsedTime() > sf::seconds(0.1)) {  
    clock.restart();  
    posxup += 10;  
}
```

passou mais de 0,1 s?

muda posição e reinicia a contagem de tempo



Time – 6-time.cpp

- Para marcar intervalos de tempo, usa-se a classe `sf::Time`
- Os valores representam uma certa quantidade de tempo, mas não em unidades de tempo propriamente, como segundos, minutos, etc
- Existem funções para converter unidades de tempo específicas
 - `sf::seconds(0.1)` no if do programa significa 0,1 segundos



Time – 6-time.cpp

- Para marcar intervalos de tempo, usa-se a classe `sf::Time`
- Os valores representam uma certa quantidade de tempo, mas não em unidades de tempo propriamente, como segundos, minutos, etc
- Existem funções para converter unidades de tempo específicas
 - `sf::seconds(0.1)` no if do programa significa 0,1 segundos
 - O mesmo valor pode ser obtido por
`sf::milliseconds(100)` ou `sf::microseconds(100000)`



Time – 6-time.cpp

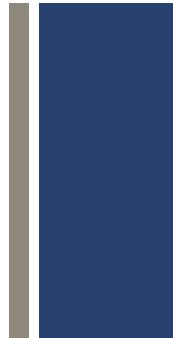
- Para marcar intervalos de tempo, usa-se a classe `sf::Time`
- Os valores representam uma certa quantidade de tempo, mas não em unidades de tempo propriamente, como segundos, minutos, etc
- Existem funções para converter unidades de tempo específicas
 - `sf::seconds(0.1)` no if do programa significa 0,1 segundos
 - O mesmo valor pode ser obtido por
`sf::milliseconds(100)` ou `sf::microseconds(100000)`
- Pode-se criar uma variável para armazenar o tempo e usá-la depois

```
// tempo de 0.1 segundos na unidade da sf::Time
const sf::Time Tempo = sf::seconds(0.1);
```

```
// Muda a posição do sprite vida a cada 0.1 segundos
if (clock.getElapsedTime() > Tempo) {
    clock.restart();
    posxup += 10;
}
```



Time – 6-time.cpp



- **Tarefa:**
 - mover o sprite vida na mesma velocidade, mas de forma mais "suave"



Time – 6-time.cpp

■ Tarefa:

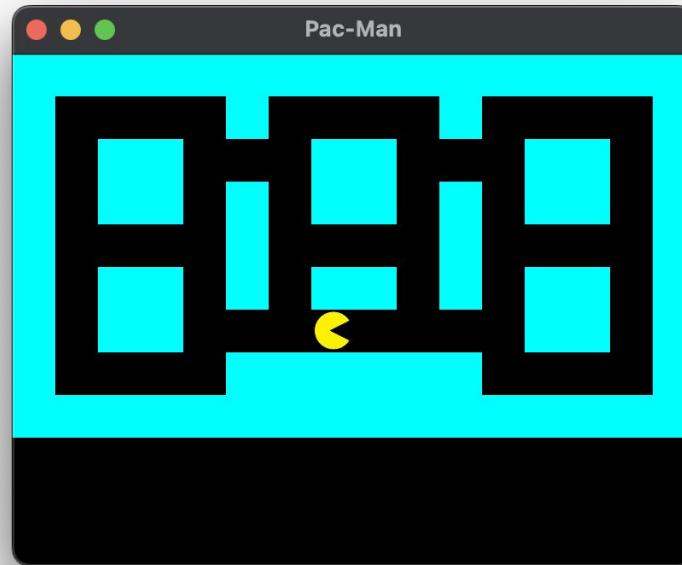
- mover o sprite vida na mesma velocidade, mas de forma mais "suave"
- não deixar a vida escapar da tela!
quando atingir o limite da direita, passar a movimentá-la para a esquerda





Juntando tudo! – 7-pacman.cpp

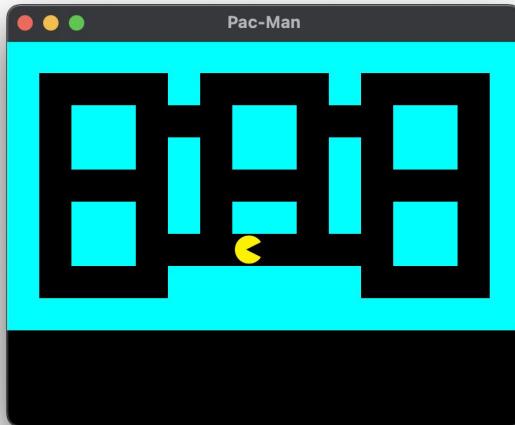
- Compile e execute o programa **7-pacman.cpp**
- Ele abre um cenário tosco (e incompleto) de Pac-Man...





Juntando tudo! – 7-pacman.cpp

- Note que o mapa não é uma figura de fundo...
- Ele é desenhado a partir de uma matriz
- 1 = parede, 0 = vazio
- cada "pixel" da matriz é um quadrado 50x50



```
char mapa[9][17] = {  
    "1111111111111111",  
    "1000010000100001",  
    "1011000110001101",  
    "1011010110101101",  
    "1000010000100001",  
    "1011010110101101",  
    "1011000000001101",  
    "1000011111100001",  
    "1111111111111111"  
};
```

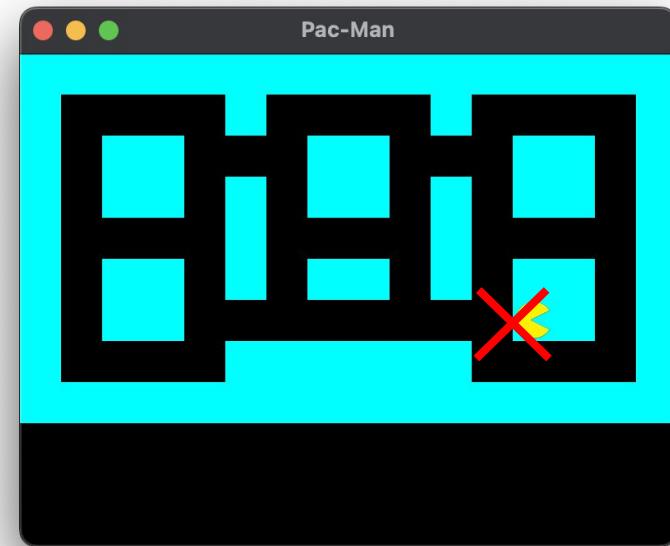
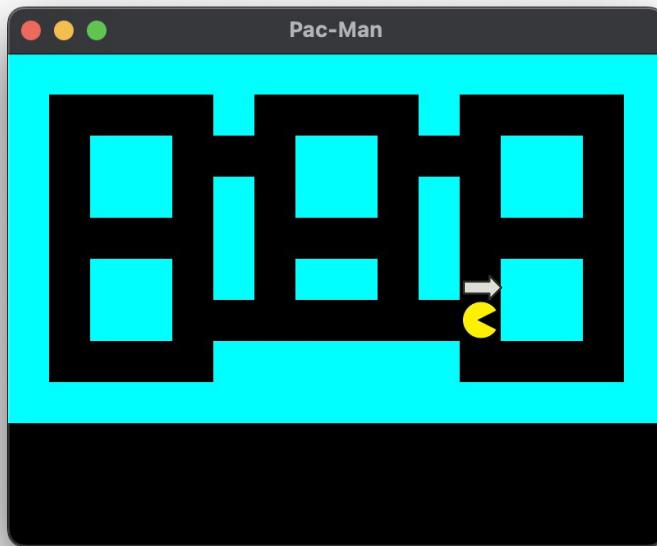
```
// parede  
sf::RectangleShape rectangle(sf::Vector2f(50, 50));  
rectangle.setFillColor(sf::Color(0, 255, 255));
```

```
//desenha paredes  
for(int i=0;i<9;i++)  
    for(int j=0;j<17;j++)  
        if (mapa[i][j]=='1') {  
            rectangle.setPosition(j*50, i*50);  
            window.draw(rectangle);  
        }
```



Juntando tudo! – 7-pacman.cpp

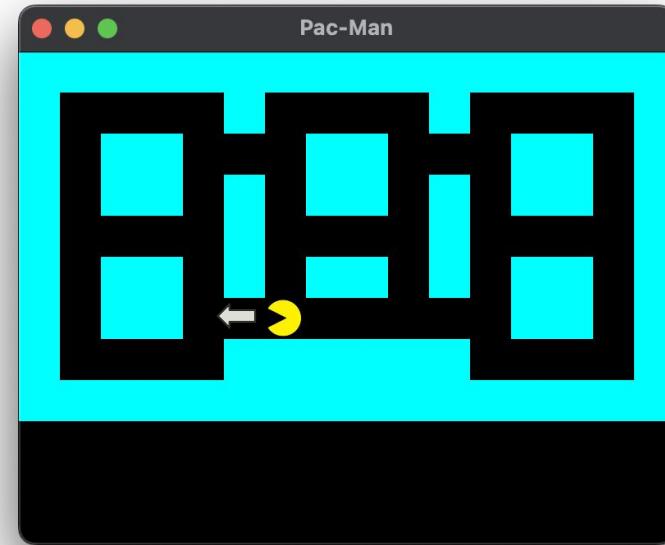
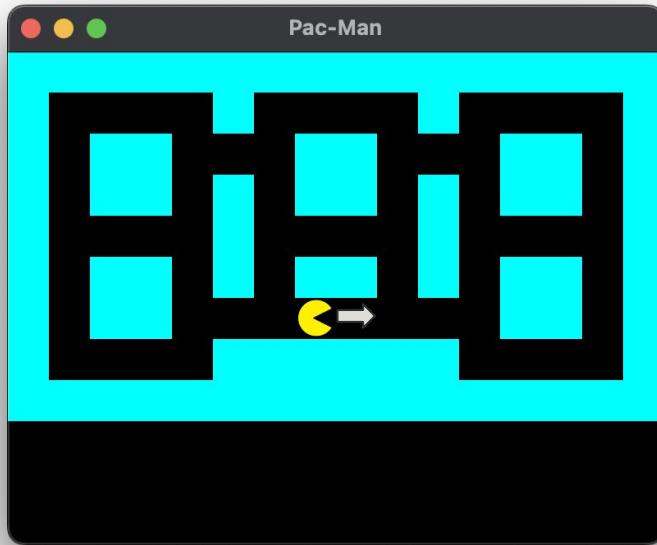
- Tarefa:
 - não deixar o Pac-Man atravessar as paredes!





Juntando tudo! – 7-pacman.cpp

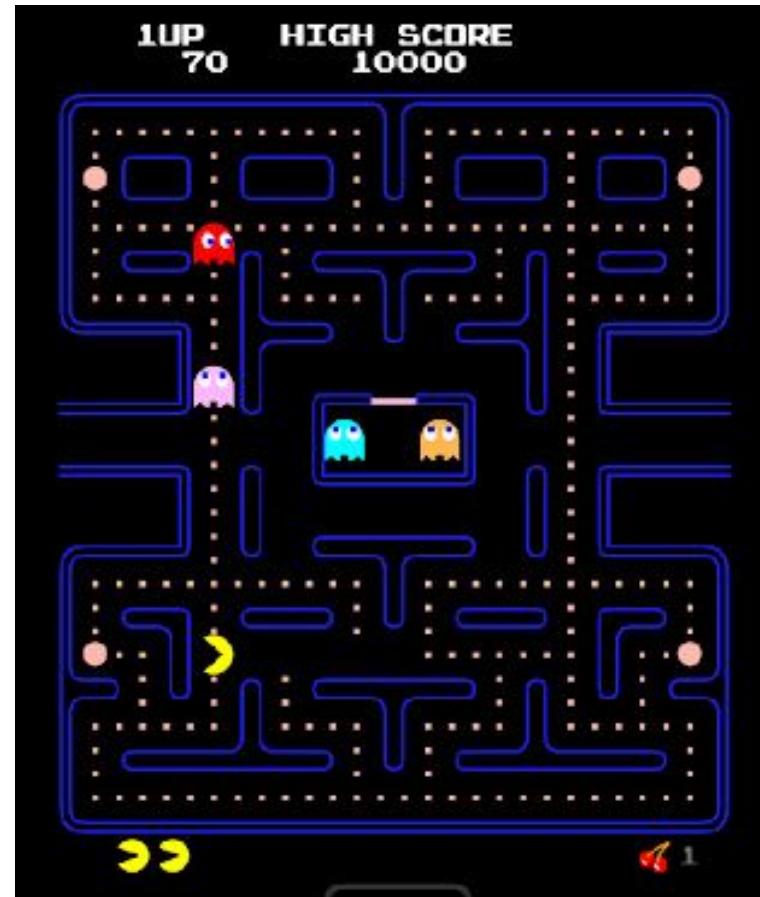
- Tarefa:
 - mudar a figura do PacMan quando ele se move para a esquerda





Pac-man – Trabalho final

- PacMan controlado pelo teclado
 - Vence ao comer todas as pílulas
 - Morre se encostar em um fantasma

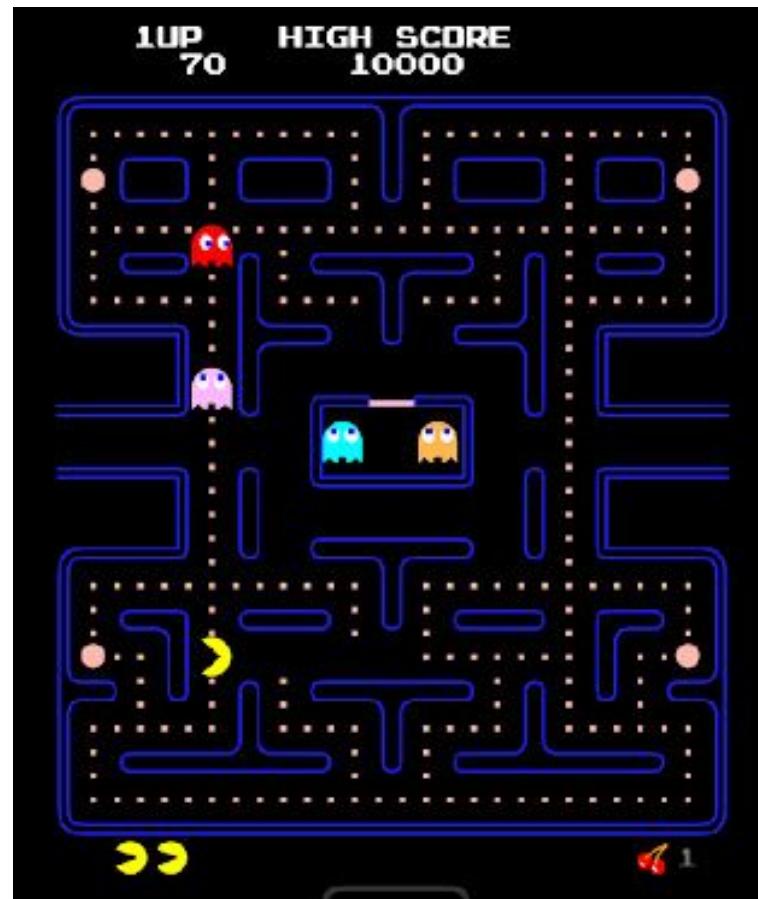




Pac-man – Trabalho final

- PacMan controlado pelo teclado
 - Vence ao comer todas as pílulas
 - Morre se encostar em um fantasma

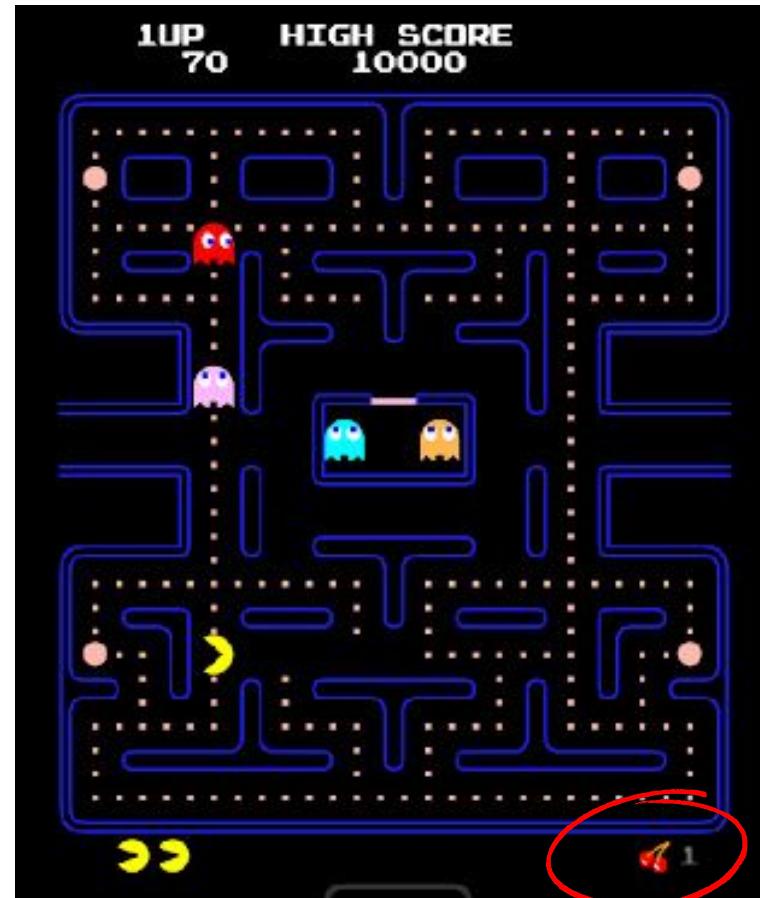
- Fantasmas se movem sozinhos
 - Mudam de direção numa encruzilhada
 - Alguns aleatoriamente
 - Outros "perseguem" o PacMan





Pac-man – Trabalho final

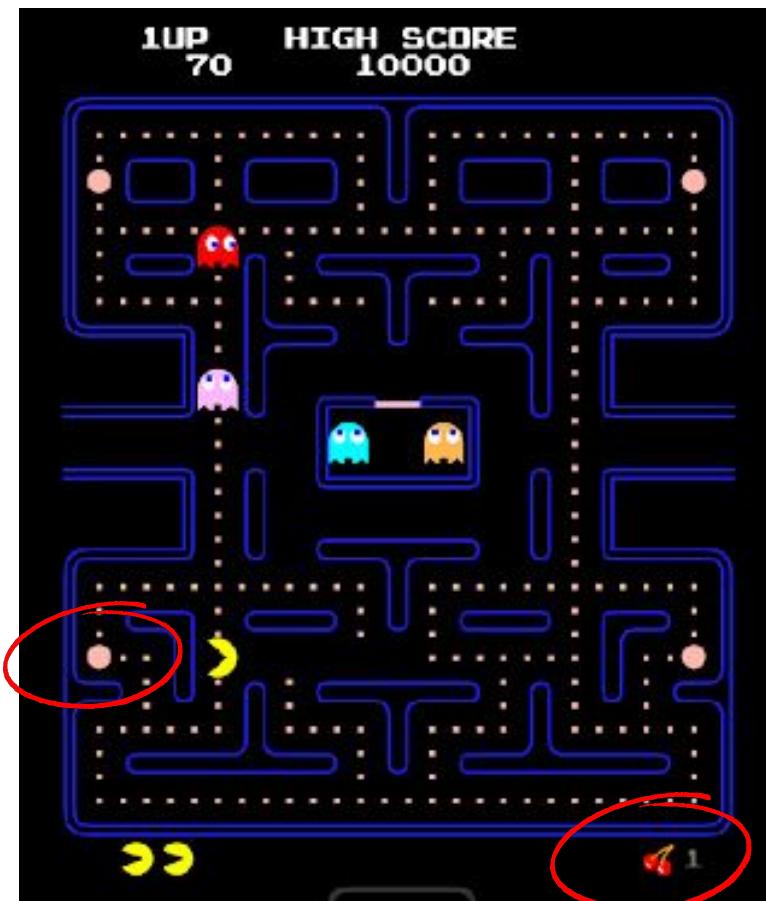
- Não precisa ter as frutinhas





Pac-man – Trabalho final

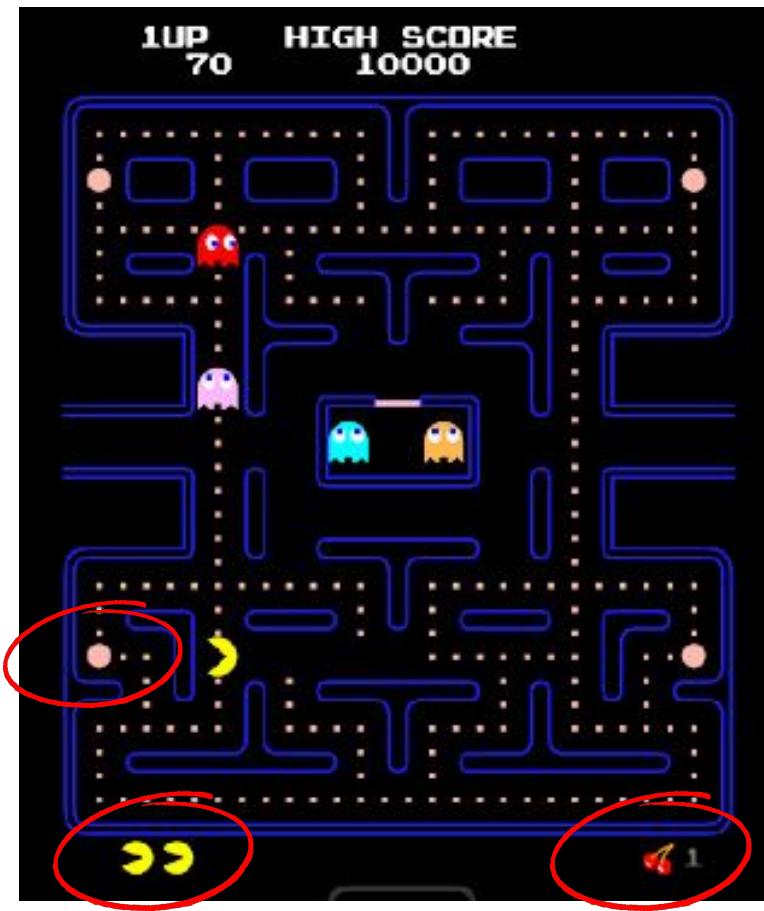
- Não precisa ter as frutinhas
- Não precisa ter as pílulas de força





Pac-man – Trabalho final

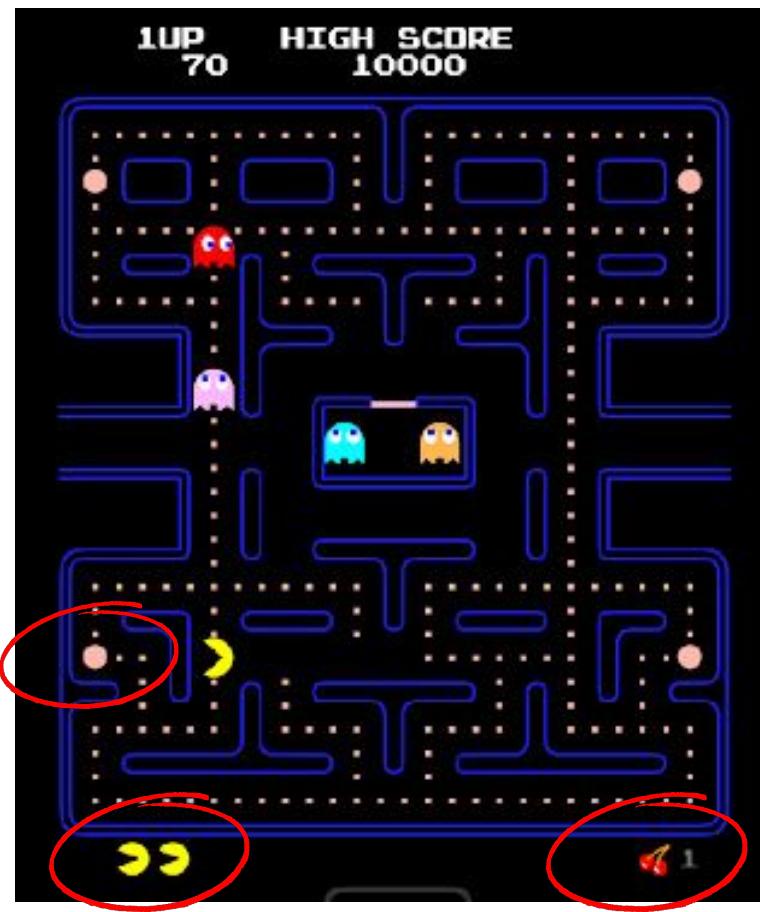
- Não precisa ter as frutinhas
- Não precisa ter as pílulas de força
- Não precisa ter vidas





Pac-man – Trabalho final

- Não precisa ter as frutinhas
- Não precisa ter as pílulas de força
- Não precisa ter vidas
- Não precisa ser o mapa original
- Nem os personagens originais
(pode criar os seus! ou um "crossover")





Mais...

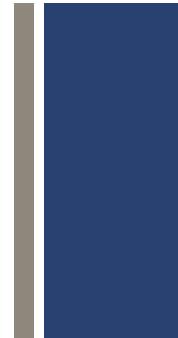
■ Outras funcionalidades

Graphics module

- Drawing 2D stuff
- Sprites and textures
- Text and fonts
- Shapes

Audio module

- Playing sounds and music
- Recording audio
- Custom audio streams
- Spatialization: Sounds in 3D





Movimento contínuo – 7b-pacman.cpp

- Sugestão para um movimento mais suave, contínuo, sem necessidade de pressionar uma tecla repetidas vezes:
 - usar booleanos, ativados quando a tecla é pressionada (e desativados quando é liberada)
 - movimentar pelo status dos booleanos
- Ver **7b-pacman.cpp**
 - nesse código, o usuário não precisa pressionar a seta o tempo todo
 - o PacMan continua se movendo na última direção informada
- **Tarefa:**
 - adicionar opção de reiniciar: tecla 'R' reposiciona na posição inicial