DESENVOLVIMENTO DE JOGOS. RECRIANDO O SOKOBAN NO STM32.

Desenvolvimento do jogo Japonês Sokoban usando STM32F103C8 (bluepill) com LCD 240x240 e HAL

PROJETO

O jogo Sokoban é um tipo de jogo de transporte e movimentação de cubos ou engradados em um armazém. O objetivo é pegar e estocar o engradado em determinada posições. O jogo é geralmente apresentado como vídeo game. Sokoban foi criado em 1981 por Hiroyuki Imabayashi, e publicado em 1982 por Thinking Rabbit, uma empresa de software localizada em Takarazuka.

Regras.

O personagem empurra caixas em torno de um labirinto e tenta colocá-los em locais designados  . Pressione a tecla de direção das casas adjacentes para empurrá-los. Pressione pressione “**up**" para mover para cima, pressione “**down**" para mover para baixo, pressione “**left**“ para mover para a esquerda, pressione “**right**" para mover para a direita.

Obs.: No Sokoban existe o “retroceder movimentação”, porém não foi implementado.

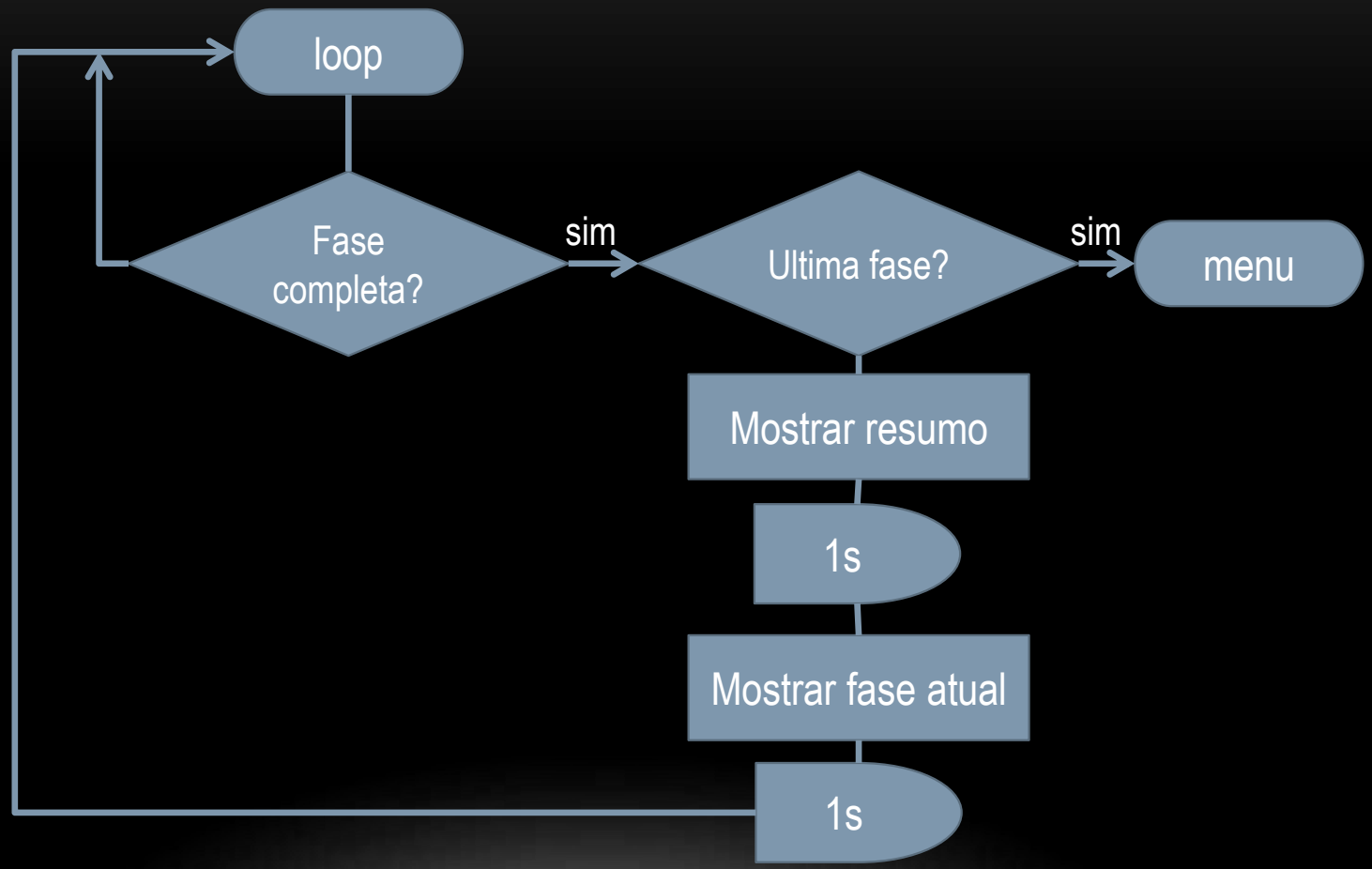
MENU E REINICIO

Pressione “**left**” ou “**right**” para selecionar a fase, em seguida pressione “**enter**” para abrir.

Durante o jogo, você pode voltar ao menu apertando “**enter**” e, para reiniciar a fase, pressione “**restart**”.

Ao final de cada fase, ele será transferido automaticamente para a próxima fase, e na última fase, será direcionado ao menu.

FLUXOGRAMA DE INICIO E FIM DE FASE



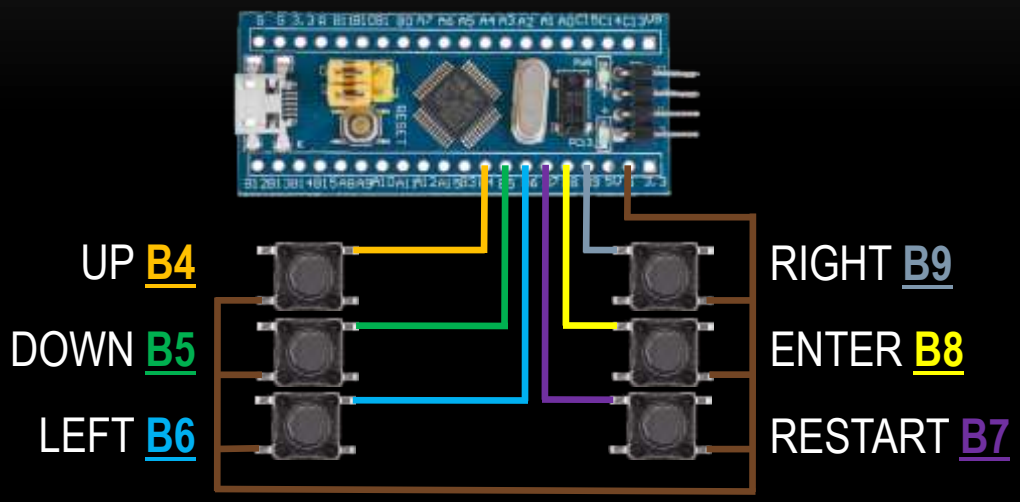
MATERIAIS NECESSÁRIOS

STM32F103C8 (bluepill)

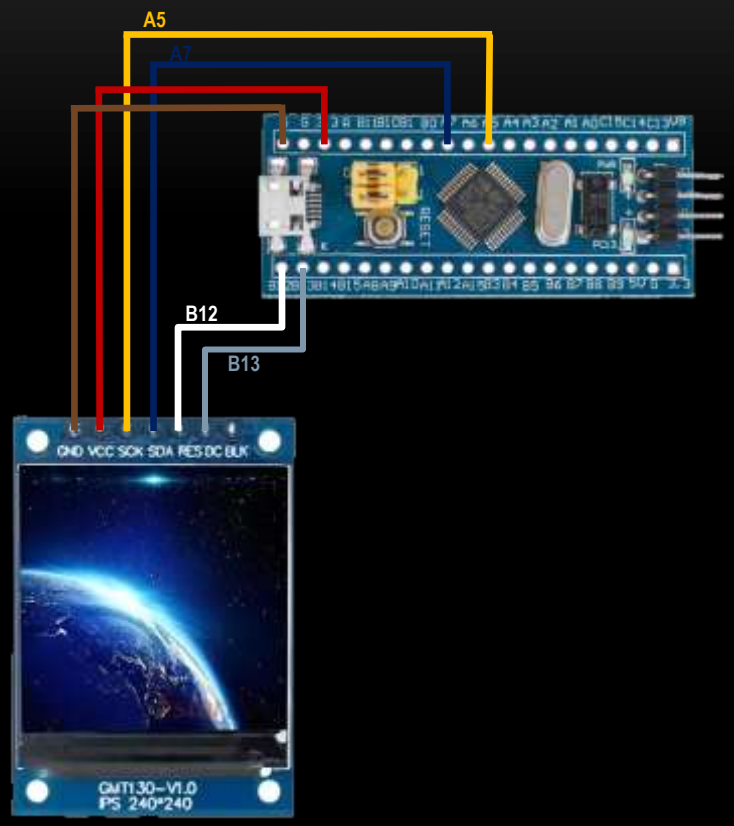
LCD 240x240 SPI

6x Push Buttons

CONEXÃO - BOTÕES



CONEXÃO - LCD



SPRITES

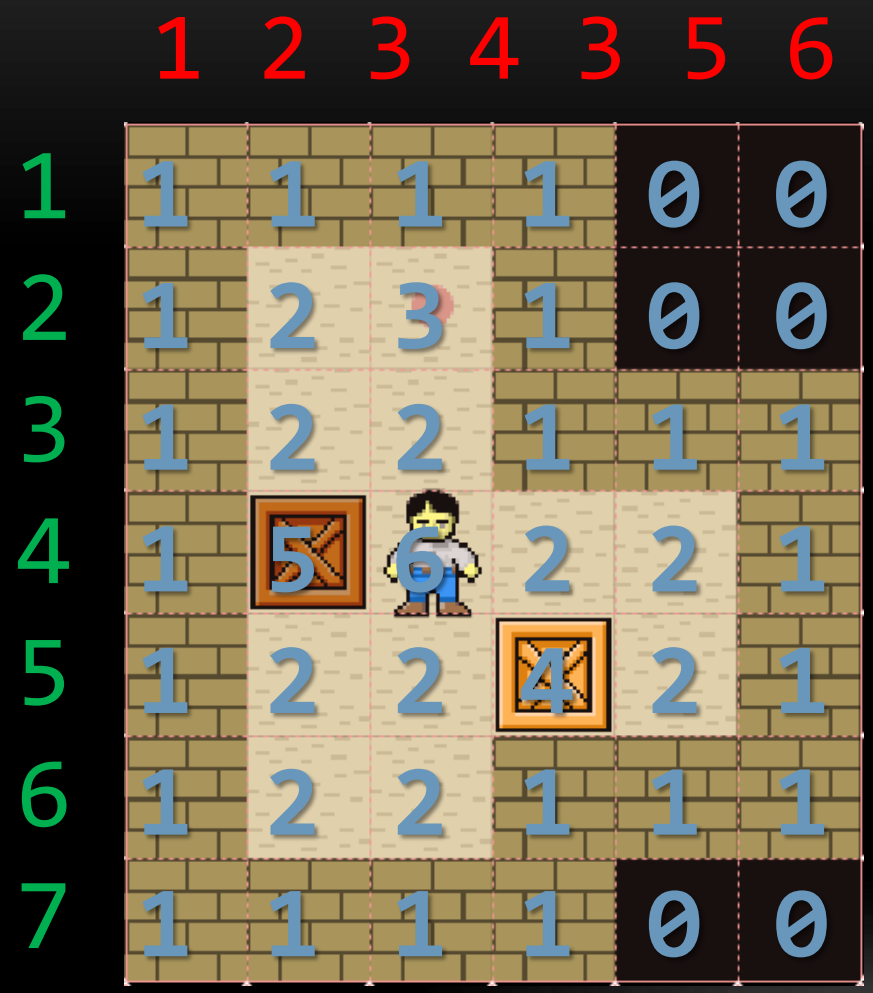
Cada sprite é indexado pela matriz do mapa
cada sprite tem um tamanho de 20x20

0 1 2 3 4 5 6 → **sprite{n}**

0 1 2 3 4 5 → **spriteIndex{n}**

Obs.: O valor **0** representa um espaço vazio

CRIANDO MAPA (PATTERN): FASE 1



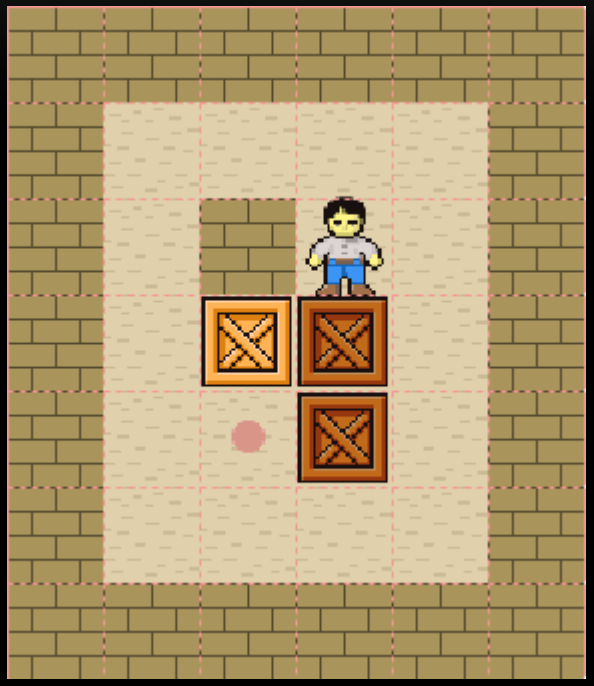
Quantidade de sprites no Y

Quantidade de sprites no X

```
uint8_t level0[7][6] = {
  {1,1,1,1,0,0},
  {1,2,3,1,0,0},
  {1,2,2,1,1,1},
  {1,5,6,2,2,1},
  {1,2,2,4,2,1},
  {1,2,2,1,1,1},
  {1,1,1,1,0,0}
};
```

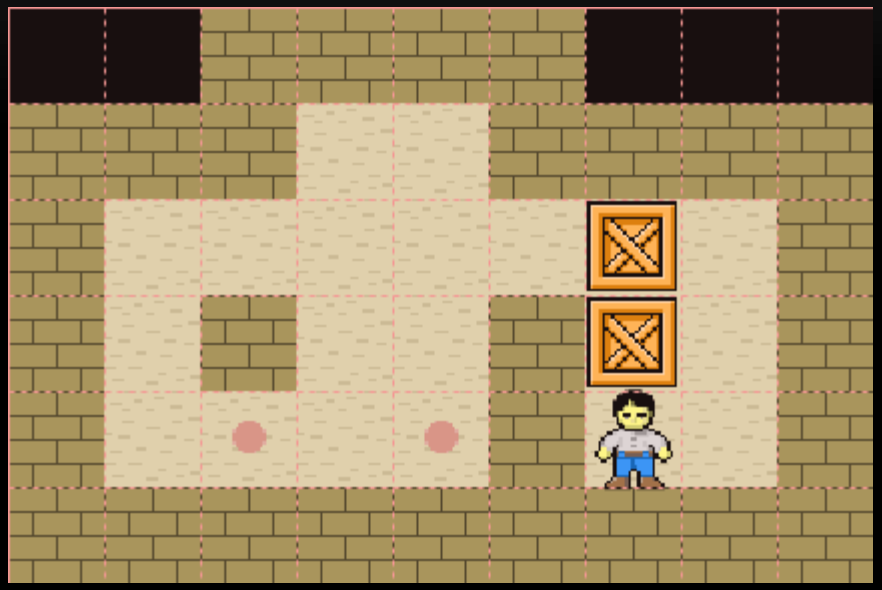
valor do pattern

FASE 2



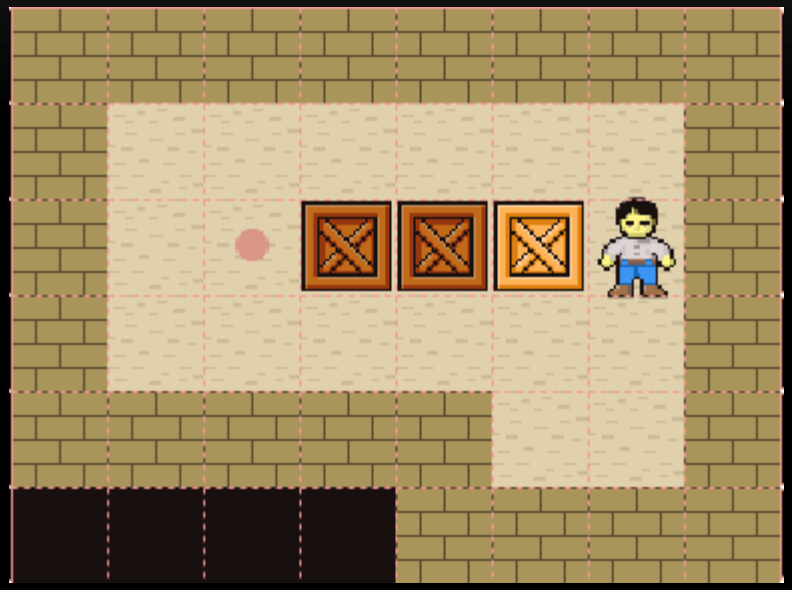
```
uint8_t level1[7][6] = {
  {1,1,1,1,1,1},
  {1,2,2,2,2,1},
  {1,2,1,6,2,1},
  {1,2,4,5,2,1},
  {1,2,3,5,2,1},
  {1,2,2,2,2,1},
  {1,1,1,1,1,1}
};
```

FASE 3



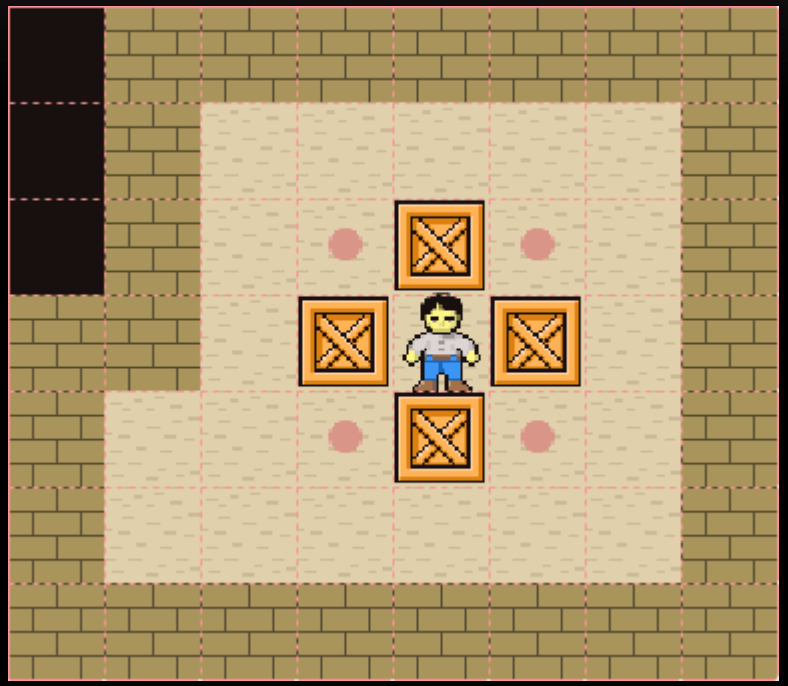
```
uint8_t level2[6][9] = {
  {0,0,1,1,1,1,0,0,0},
  {1,1,1,2,2,1,1,1,1},
  {1,2,2,2,2,2,4,2,1},
  {1,2,1,2,2,1,4,2,1},
  {1,2,3,2,3,1,6,2,1},
  {1,1,1,1,1,1,1,1,1}
};
```

FASE 4



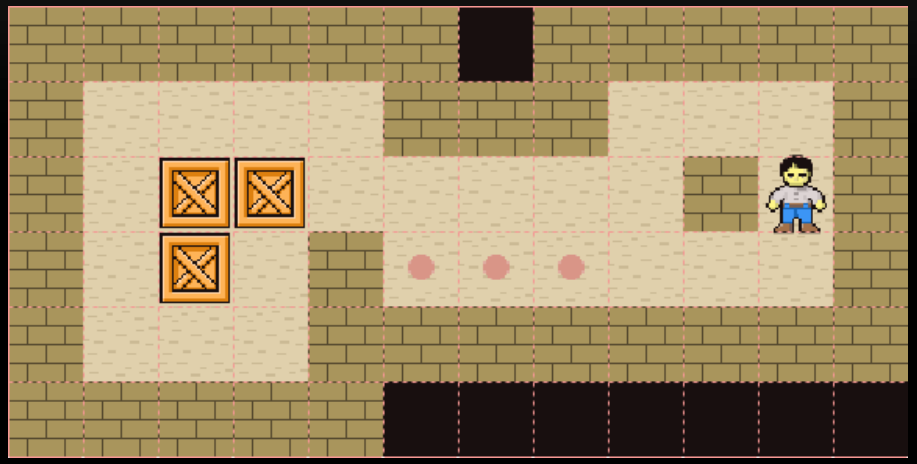
```
uint8_t level3[6][8] = {
  {1,1,1,1,1,1,1,1},
  {1,2,2,2,2,2,2,1},
  {1,2,3,5,5,4,6,1},
  {1,2,2,2,2,2,2,1},
  {1,1,1,1,1,2,2,1},
  {0,0,0,0,1,1,1,1}
};
```

FASE 5



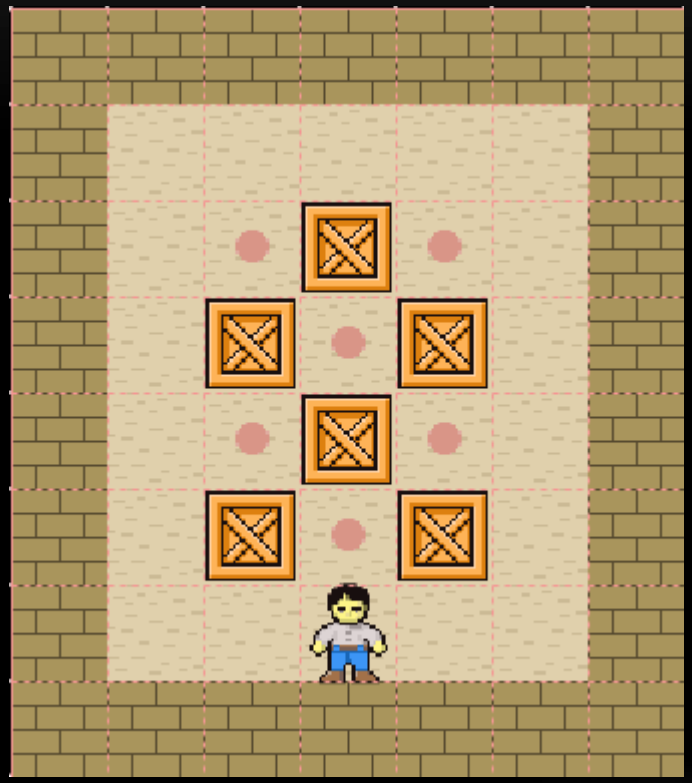
```
uint8_t level4[7][8] = {
  {0,1,1,1,1,1,1,1},
  {0,1,2,2,2,2,2,1},
  {0,1,2,3,4,3,2,1},
  {1,1,2,4,6,4,2,1},
  {1,2,2,3,4,3,2,1},
  {1,2,2,2,2,2,2,1},
  {1,1,1,1,1,1,1,1}
};
```

FASE 6



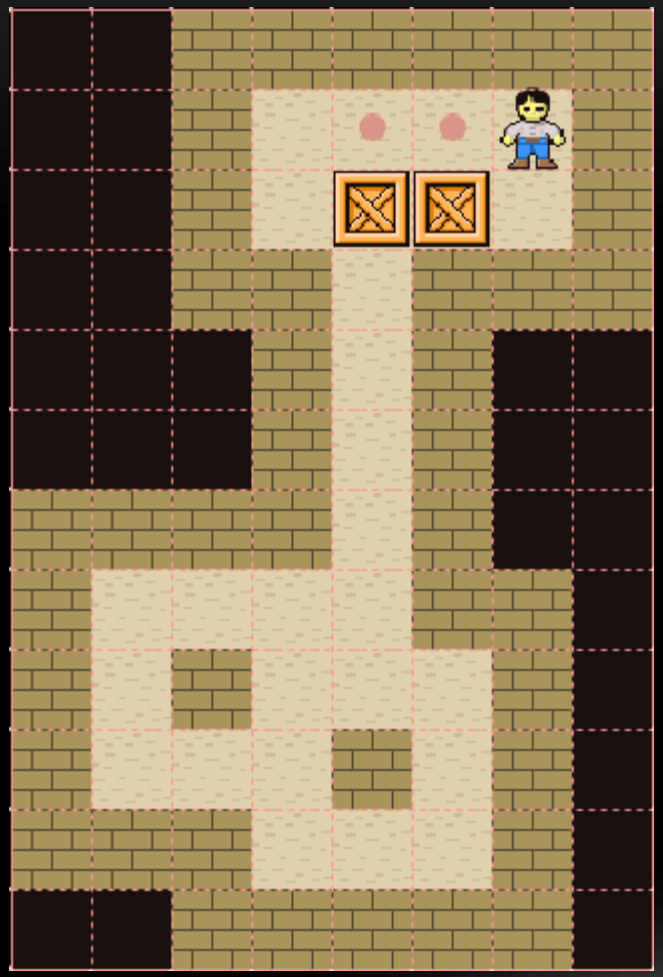
```
uint8_t level5[6][12] = {
  {1,1,1,1,1,1,0,1,1,1,1,1},
  {1,2,2,2,2,1,1,1,2,2,2,1},
  {1,2,4,4,2,2,2,2,2,1,6,1},
  {1,2,4,2,1,3,3,3,2,2,2,1},
  {1,2,2,2,1,1,1,1,1,1,1,1},
  {1,1,1,1,1,0,0,0,0,0,0,0}
};
```

FASE 7



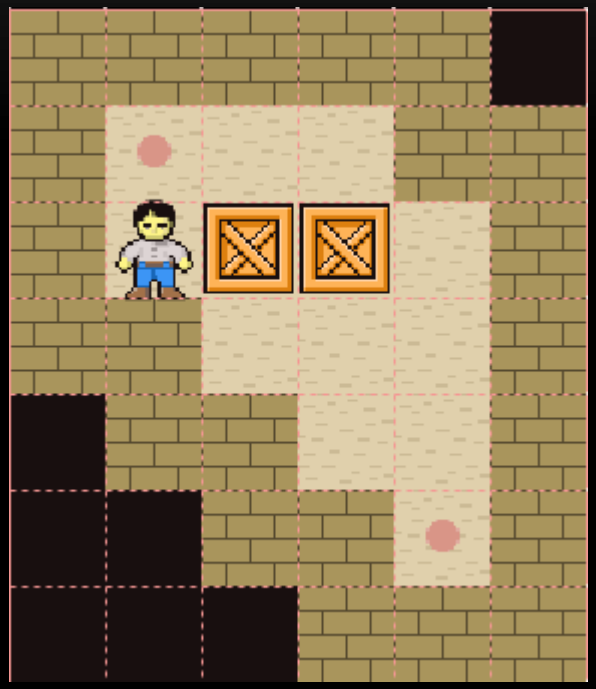
```
uint8_t level6[8][7] = {
  {1,1,1,1,1,1,1},
  {1,2,2,2,2,2,1},
  {1,2,3,4,3,2,1},
  {1,2,4,3,4,2,1},
  {1,2,3,4,3,2,1},
  {1,2,4,3,4,2,1},
  {1,2,2,6,2,2,1},
  {1,1,1,1,1,1,1},
};
```

FASE 8



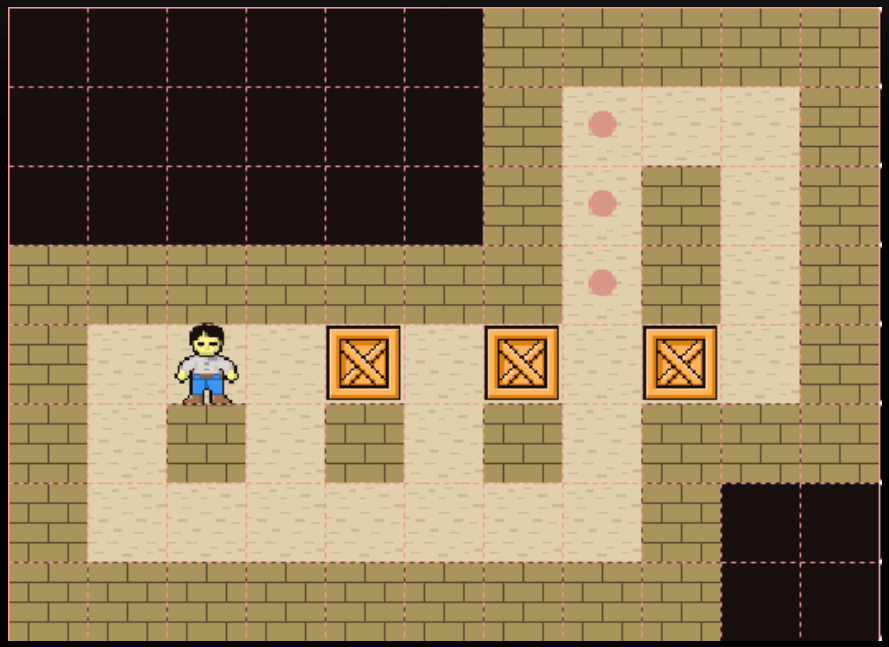
```
uint8_t level7[12][8] = {
  {0,0,1,1,1,1,1,1},
  {0,0,1,2,3,3,6,1},
  {0,0,1,2,4,4,2,1},
  {0,0,1,1,2,1,1,1},
  {0,0,0,1,2,1,0,0},
  {0,0,0,1,2,1,0,0},
  {1,1,1,1,2,1,0,0},
  {1,2,2,2,2,1,1,0},
  {1,2,1,2,2,2,1,0},
  {1,2,2,2,1,2,1,0},
  {1,1,1,2,2,2,1,0},
  {0,0,1,1,1,1,1,0}
};
```

FASE 9



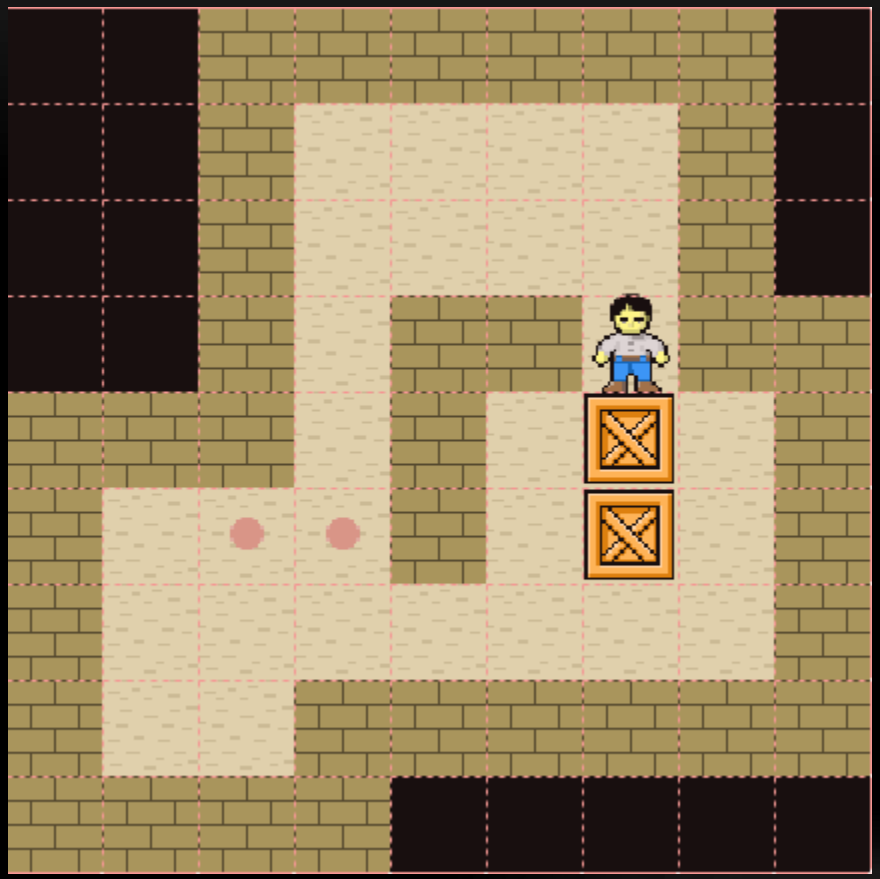
```
uint8_t level8[7][6] = {
  {1,1,1,1,1,0},
  {1,3,2,2,1,1},
  {1,6,4,4,2,1},
  {1,1,2,2,2,1},
  {0,1,1,2,2,1},
  {0,0,1,1,3,1},
  {0,0,0,1,1,1}
};
```

FASE 10



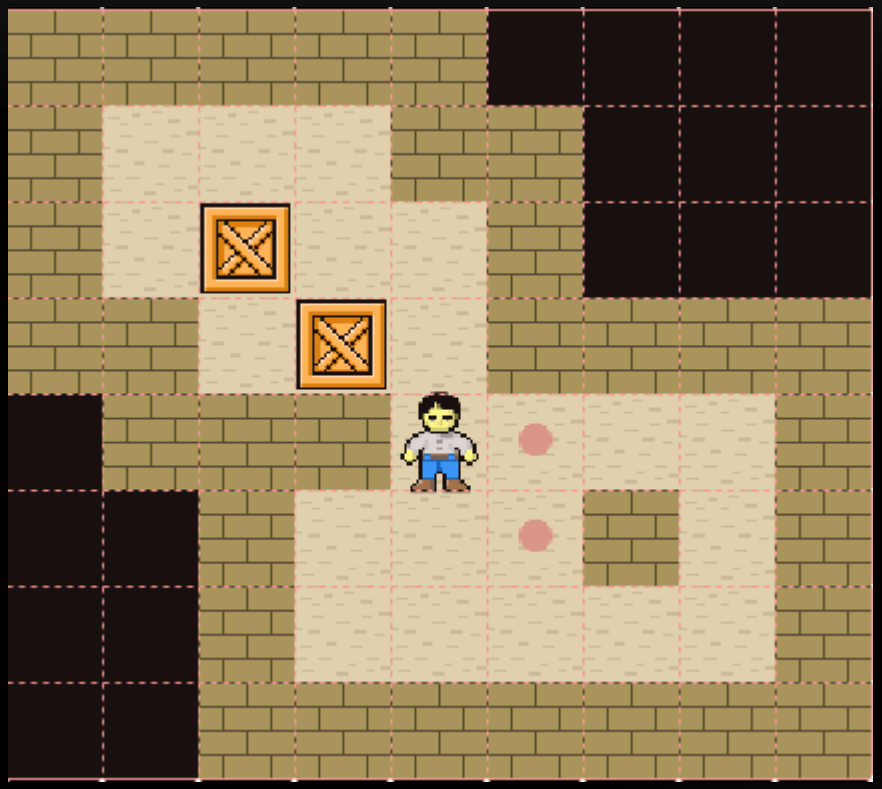
```
uint8_t level9[8][11] = {
  {0,0,0,0,0,0,1,1,1,1,1},
  {0,0,0,0,0,0,1,3,2,2,1},
  {0,0,0,0,0,0,1,3,1,2,1},
  {1,1,1,1,1,1,1,3,1,2,1},
  {1,2,6,2,4,2,4,2,4,2,1},
  {1,2,1,2,1,2,1,2,1,1,1},
  {1,2,2,2,2,2,2,2,1,0,0},
  {1,1,1,1,1,1,1,1,1,0,0}
};
```

FASE 11



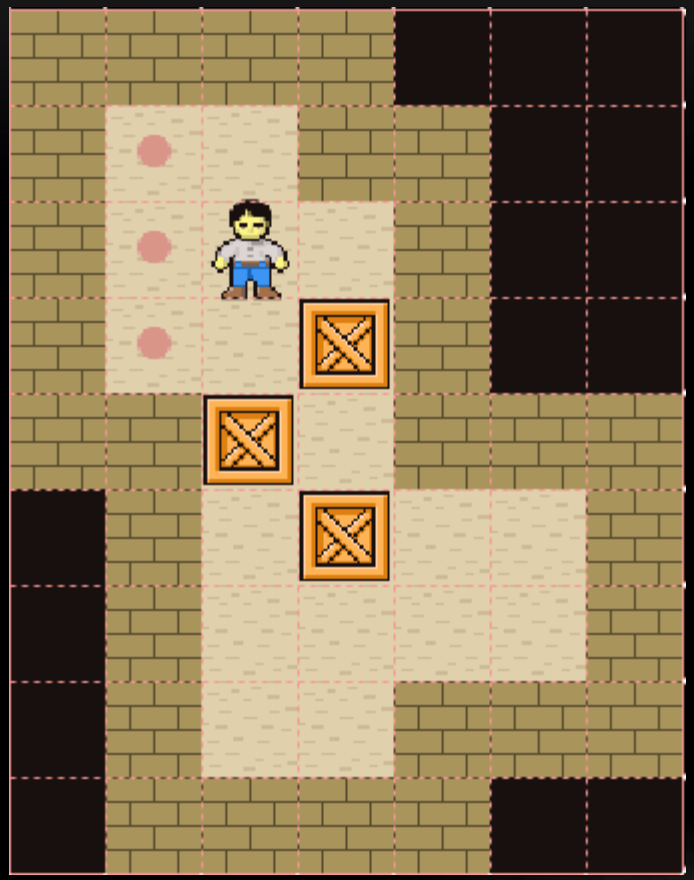
```
uint8_t level10[9][9] = {
  {0,0,1,1,1,1,1,1,0},
  {0,0,1,2,2,2,2,1,0},
  {0,0,1,2,2,2,2,1,0},
  {0,0,1,2,1,1,6,1,1},
  {1,1,1,2,1,2,4,2,1},
  {1,2,3,3,1,2,4,2,1},
  {1,2,2,2,2,2,2,2,1},
  {1,2,2,1,1,1,1,1,1},
  {1,1,1,1,0,0,0,0,0}
};
```

FASE 12



```
uint8_t level11[8][9] = {
  {1,1,1,1,1,0,0,0,0},
  {1,2,2,2,1,1,0,0,0},
  {1,2,4,2,2,1,0,0,0},
  {1,1,2,4,2,1,1,1,1},
  {0,1,1,1,6,3,2,2,1},
  {0,0,1,2,2,3,1,2,1},
  {0,0,1,2,2,2,2,2,1},
  {0,0,1,1,1,1,1,1,1}
};
```

FASE 13

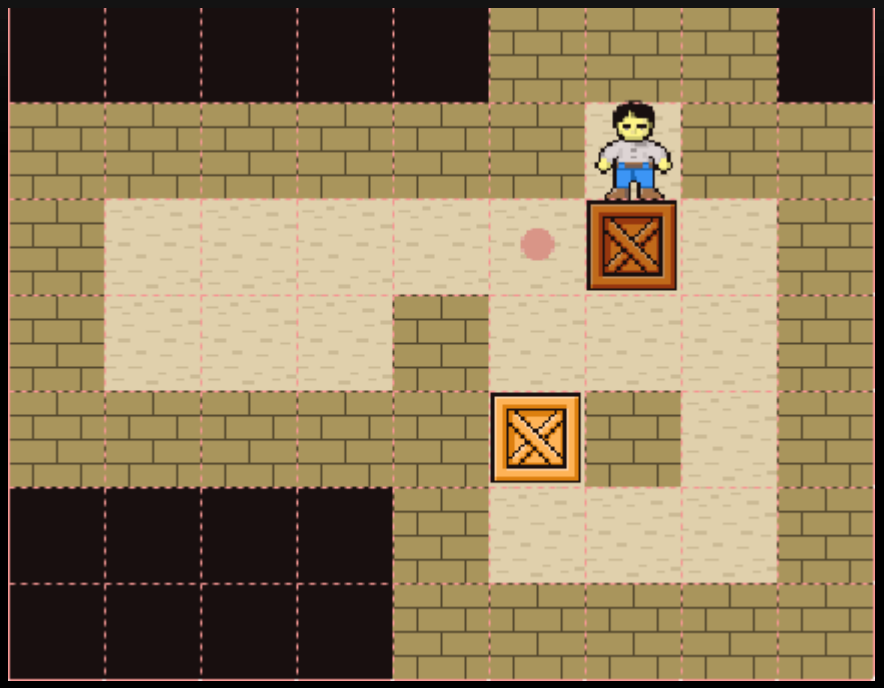


```
uint8_t level12[9][7] = {
  {1,1,1,1,0,0,0},
  {1,3,2,1,1,0,0},
  {1,3,6,2,1,0,0},
  {1,3,2,4,1,0,0},
  {1,1,4,2,1,1,1},
  {0,1,2,4,2,2,1},
  {0,1,2,2,2,2,1},
  {0,1,2,2,1,1,1},
  {0,1,1,1,1,0,0}
};
```

FASE 14

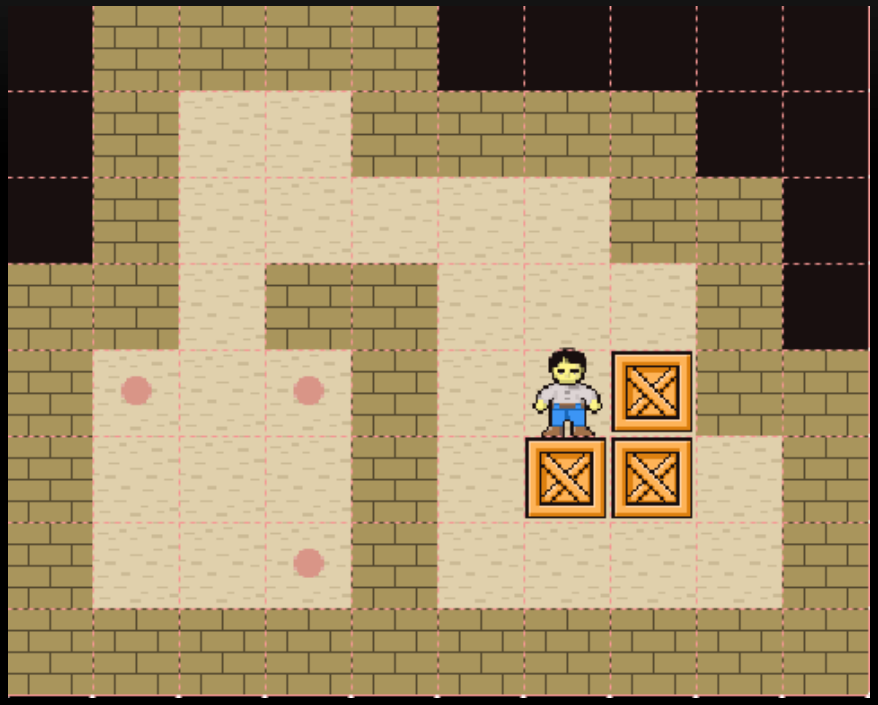
```
uint8_t level13[6][7] = {
  {1,1,1,1,1,1,1},
  {1,2,2,2,2,2,1},
  {1,2,1,2,1,2,1},
  {1,3,2,4,5,6,1},
  {1,2,2,2,1,1,1},
  {1,1,1,1,1,0,0}
};
```

FASE 15



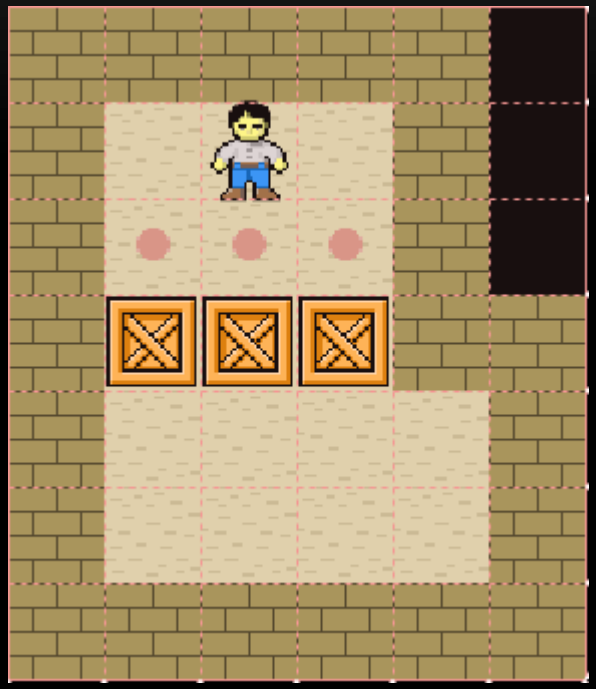
```
uint8_t level14[7][9] = {
  {0,0,0,0,0,1,1,1,0},
  {1,1,1,1,1,1,6,1,1},
  {1,2,2,2,2,3,5,2,1},
  {1,2,2,2,1,2,2,2,1},
  {1,1,1,1,1,4,1,2,1},
  {0,0,0,0,1,2,2,2,1},
  {0,0,0,0,1,1,1,1,1}
};
```

FASE 16



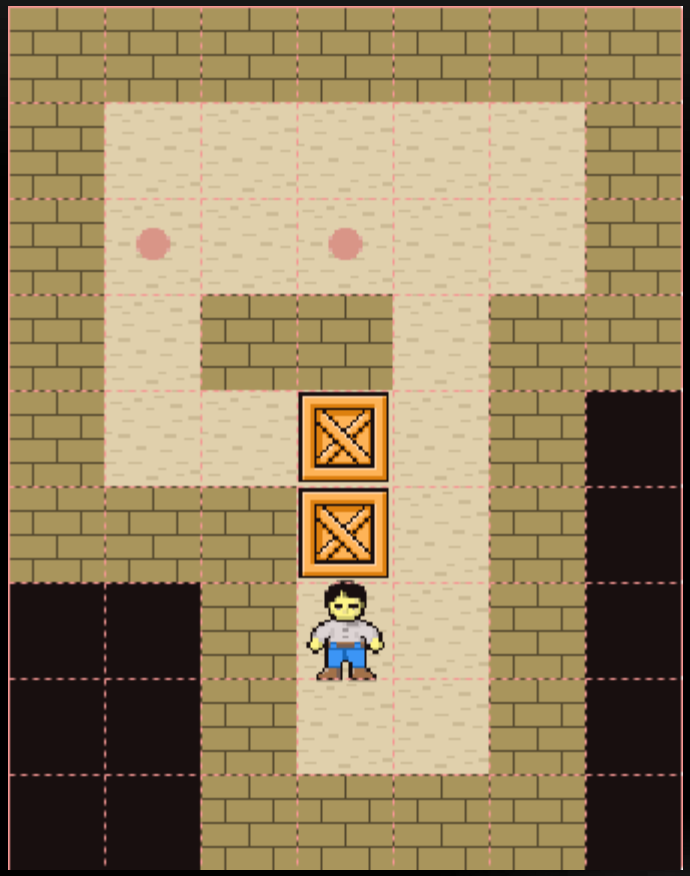
```
uint8_t level15[8][10] = {
  {0,1,1,1,1,0,0,0,0,0},
  {0,1,2,2,1,1,1,1,0,0},
  {0,1,2,2,2,2,2,1,1,0},
  {1,1,2,1,1,2,2,2,1,0},
  {1,3,2,3,1,2,6,4,1,1},
  {1,2,2,2,1,2,4,4,2,1},
  {1,2,2,3,1,2,2,2,2,1},
  {1,1,1,1,1,1,1,1,1,1}
};
```

FASE 17



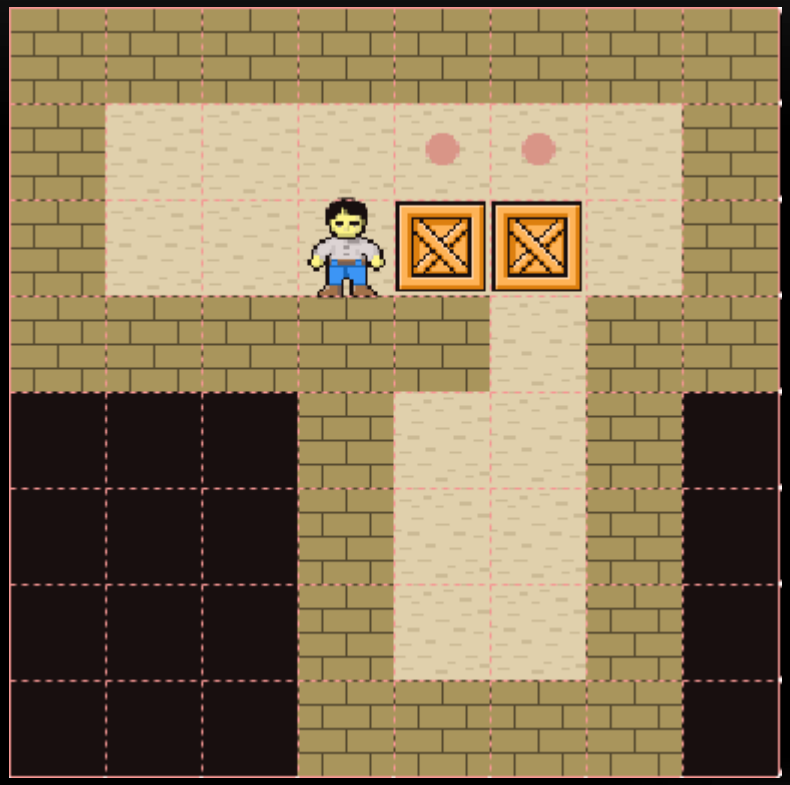
```
uint8_t level16[7][6] = {
  {1,1,1,1,1,0},
  {1,2,6,2,1,0},
  {1,3,3,3,1,0},
  {1,4,4,4,1,1},
  {1,2,2,2,2,1},
  {1,2,2,2,2,1},
  {1,1,1,1,1,1}
};
```

FASE 18



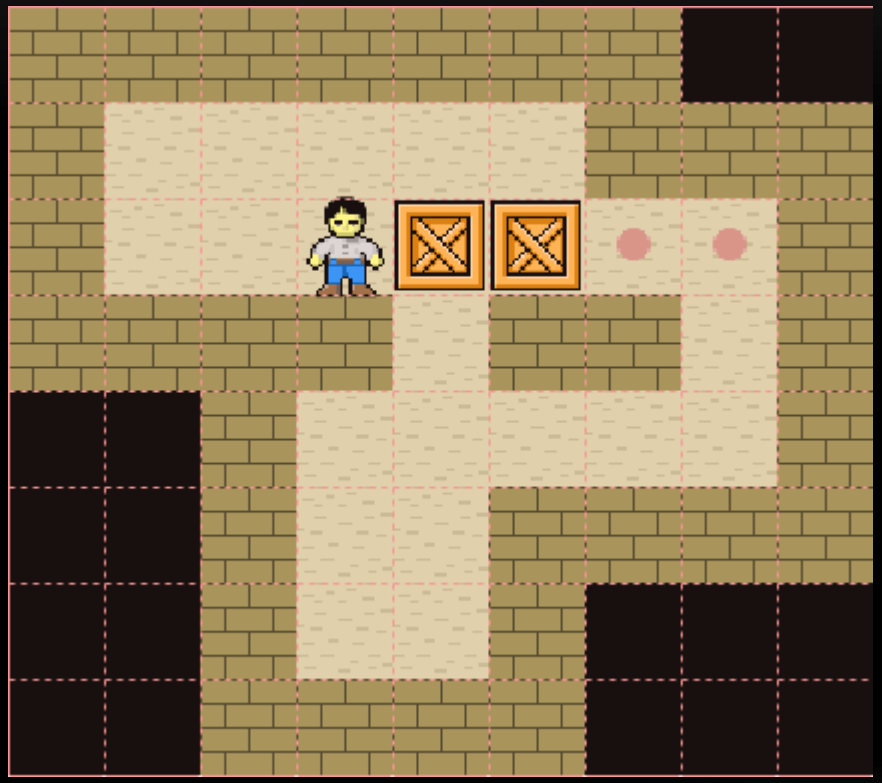
```
uint8_t level17[9][7] = {
  {1,1,1,1,1,1,1},
  {1,2,2,2,2,2,1},
  {1,3,2,3,2,2,1},
  {1,2,1,1,2,1,1},
  {1,2,2,4,2,1,0},
  {1,1,1,4,2,1,0},
  {0,0,1,6,2,1,0},
  {0,0,1,2,2,1,0},
  {0,0,1,1,1,1,0}
};
```

FASE 19



```
uint8_t level18[8][8] = {
  {1,1,1,1,1,1,1,1},
  {1,2,2,2,3,3,2,1},
  {1,2,2,6,4,4,2,1},
  {1,1,1,1,1,2,1,1},
  {0,0,0,1,2,2,1,0},
  {0,0,0,1,2,2,1,0},
  {0,0,0,1,2,2,1,0},
  {0,0,0,1,1,1,1,0}
};
```

FASE 20



```
uint8_t level19[8][9] = {
  {1,1,1,1,1,1,1,0,0},
  {1,2,2,2,2,2,1,1,1},
  {1,2,2,6,4,4,3,3,1},
  {1,1,1,1,2,1,1,2,1},
  {0,0,1,2,2,2,2,2,1},
  {0,0,1,2,2,1,1,1,1},
  {0,0,1,2,2,1,0,0,0},
  {0,0,1,1,1,1,0,0,0}
};
```

INTENCIONALMENTE EM BRANCO

VARIÁVEIS:
MATRIZ DAS FASES

```
uint8_t level0[7][6];
uint8_t level1[7][6];
uint8_t level2[6][9];
uint8_t level3[6][8];
uint8_t level4[7][8];
uint8_t level5[6][12];
uint8_t level6[8][7];
uint8_t level7[12][8];
uint8_t level8[7][6];
uint8_t level9[8][11];
```

```
uint8_t level10[9][9];
uint8_t level11[8][9];
uint8_t level12[9][7];
uint8_t level13[6][7];
uint8_t level14[7][9];
uint8_t level15[8][10];
uint8_t level16[7][6];
uint8_t level17[9][7];
uint8_t level18[8][8];
uint8_t level19[8][9];
```

VARIÁVEIS (CONTINUAÇÃO): MATRIZ DOS SPRITES

```
uint16_t spriteList[6][400]
```

16 bits (2 bytes) por Pixel

$$Size = \frac{Width * Height * 16}{8} * Qtd$$

$$Size = \frac{20 * 20 * 16}{8} * 6$$

$$Size = 4800\ bytes$$

VARIÁVEIS (CONTINUAÇÃO):

```
typedef struct{
  uint8_t x;
  uint8_t y;
  uint8_t active;
} Box;
```


```
Box boxList[5];
uint8_t gameBoxAmount;

uint8_t currentLevelIndex;
uint8_t currentLevelWidth;
uint8_t currentLevelHeight;
uint16_t currentMovementCount;
```

```
uint8_t offsetMap[2] = {0, 0};

uint8_t playerX;

uint8_t playerY;

uint8_t movementDirection = 0;
uint8_t keyPressed = 0;
uint8_t keyLocked = 0;
uint8_t isMenuMode = 1;
```

O `struct` **`Box`** é a representação das caixas com:

Posição **x**;

Posição **y**;

Se está **ativo**, ficando mais escura sprite(5).

O **`BoxList[5]`** é o array que armazena até 5 caixas por mapa.

`gameBoxAmount` é a quantidade de caixas por mapa.

`currentLevelIndex` é o nível atual (a partir do 0)

`currentLevelWidth` é a quantidade da largura de sprites no grid.
`currentLevelHeight` é a quantidade da altura de sprites no grid.

Ex.: um mapa de 6x4 teremos:

```
currentLevelWidth = 6;
currentLevelHeight = 4;
```

`currentMovementCount` é a quantidade de movimentos feitos pelo jogador.

keyPressed é o debouce do teclado, assim permitindo um clique por vez.
keyLocked é a "trava" o teclado via software
isMenuMode é a variável que fala se esta no menu ou no jogo.

movementDirection diz a direção clicada:
1 = UP
2 = DOWN
3 = LEFT
4 = RIGHT

offsetMap[2] deslocamento do mapa para ser gerarado no centro da tela.

offsetMap[0] representa o **X**
offsetMap[1] representa o **y**

Formulas dos offsets:

$X = \frac{Dw - Sw * Qx}{2}$	$Y = \frac{Dh - Sh * Qy}{2}$

Dw: Largura da tela
Sw: Largura do sprite
Qx: Quantida de sprites no **X**

Dh: Altura da tela
Sh: Altura do sprite
Qy: Quantidade de sprites no **Y**

INTENCIONALMENTE EM BRANCO

FUNÇÕES

```
uint8_t *getAddress(uint8_t item)
void generateLevel(uint8_t w, uint8_t h, uint8_t arr[h][w])
void generateLevelByIndex(uint8_t levelIndex)
int8_t hasBoxAt(uint8_t x, uint8_t y)
void updateBox(uint8_t levelIndex)
void updateLevel(uint8_t direction)

uint8_t gameWin()
void showSummary()
void showCurrentLevel()
void showMenu()
```

Funções usada pelo LCD:

```
void tft_init()
void tft_fillRect(int16_t x, uint16_t y, uint16_t w, uint16_t h, uint16_t color)
void renderBitmap(int16_t x, uint16_t y, uint16_t w, uint16_t h, uint16_t *bitmap)
```

```
uint8_t *getAddress(uint8_t item)
```

`getAddress()` retorna o endereço de memoria do mapa selecionado em **`item`**.

```
uint8_t *getAddress(uint8_t item) {
  if (item == 0) return (void*)level0;
  if (item == 1) return (void*)level1;
  ...
  ...
  ...
  if (item == 19) return (void*)level19;
  return NULL;
}
```

```
int8_t hasBoxAt(uint8_t x, uint8_t y)
```

`hasBoxAt()` retorna o index da caixa na posição **x** e **y**.
Retorna -1 caso não existir.

```
int8_t hasBoxAt(uint8_t x, uint8_t y){
  for (uint8_t boxIndex = 0; boxIndex < gameBoxAmount; boxIndex++){
    if ( boxList[boxIndex].x == x && boxList[boxIndex].y == y ){
      return boxIndex;
    }
  }
  return -1;
}
```

uint8_t gameWin()

gameWin() retorna 1 caso o jogador tenha colocador todas as caixas no lugar

```
uint8_t gameWin(){

  uint8_t (*mapa)[currentLevelHeight][currentLevelWidth] = (uint8_t (*)[currentLevelHeight][currentLevelWidth]) getAddress(currentLevelIndex);

  for (uint8_t boxIndex = 0; boxIndex < gameBoxAmount; boxIndex++){
    uint8_t box_x = boxList[boxIndex].x;
    uint8_t box_y = boxList[boxIndex].y;

    if ( (*mapa)[box_y][box_x] != 3 && (*mapa)[box_y][box_x] != 5 ){
      return 0;
    }
  }

  return 1;
}
```

Coleta o endereço de memoria da fase e atribui no:
uint8_t (*mapa)[h][w];

É verificado se o pattern do mapa e a posição da caixa correspondem ao **spriteIndex{3}** ou **spriteIndex {5}** em negação.
spriteIndex{5} corresponde à caixa escura e não é o slot da caixa, porém se o pattern diz 5 então pressume um **spriteIndex {3}** por trás da caixa.

```
void updateBox(uint8_t levelIndex)
```

updateBox() é chamado a cada movimento do jogador para atualizar as caixas caso entrar/sair da dona de conexão sprite(3).

```
void updateBox(uint8_t levelIndex){
  for (uint8_t boxIndex = 0; boxIndex < gameBoxAmount; boxIndex++){
    uint8_t (*mapa)[currentLevelHeight][currentLevelWidth] = (uint8_t (*)[currentLevelHeight][currentLevelWidth]) getAddress(currentLevelIndex);
    boxList[boxIndex].active = ((*mapa)[boxList[boxIndex].y][boxList[boxIndex].x] == 3 || (*mapa)[boxList[boxIndex].y][boxList[boxIndex].x] == 5);
    renderBitmap(20*boxList[boxIndex].x+offsetMap[0], 20*boxList[boxIndex].y+offsetMap[1], 20, 20, spriteList[3 + boxList[boxIndex].active]);
  }
}
```

Tamanhdo do sprite: 20x20

É iterado o array das caixas de 0 até gameAmountBox
É verificado se o pattern do mapa e a posição da caixa corresponde ao **spriteIndex{3}** ou **spriteIndex{5}**
Caso verdadeiro: boxList[boxIndex].active = 1;
É renderizado com o **spriteIndex{3 + boxList[boxIndex].active}**
3 para caixa clara **sprite{4}**
4 para caixa escura **sprite{5}**

```
void showSummary()
```

`showSummary()` mostra a quantidade de movimentos no final de casa fase.

```
void showSummary(){
  char text[10];
  sprintf(text, "Moves: %i", currentMovementCount);
  tft_fillRect(0, 0, 240, 240, 0x0000);
  tft_drawText(30, 108, text, 0xFFFF, 0x0000, 3);
}
```

```
void showCurrentLevel()
```

`showCurrentLevel()` mostra a fase atual antes de cada fase.

```
void showCurrentLevel(){
  char text[9];
  sprintf(text, "Level: %i", currentLevelIndex+1);
  tft_fillRect(0, 0, 240, 240, 0x0000);
  tft_drawText(39, 108, text, 0xFFFF, 0x0000, 3);
}
```

```
void showMenu()
```

`showMenu()` Mostra o menu inicial ao ligar, quanto aperta “Enter” ou no final da ultima fase.

```
void showMenu(){
  char text[6];
  sprintf(text, "< %2i >", currentLevelIndex+1);
  tft_drawText(30, 100, text, 0xFFFF, 0x0000, 5);
}
```

```
void generateLevel(uint8_t w, uint8_t h, uint8_t arr[h][w])
```

generateLevel() gera um mapa a partir do pattern **levelN[h][w]**

```
void generateLevel(uint8_t w, uint8_t h, uint8_t arr[h][w]){
  currentLevelWidth = w;
  currentLevelHeight = h;
  gameBoxAmount = 0;
  currentMovementCount = 0;

  offsetMap[0] = (240 - 20*w) / 2; // offsetX
  offsetMap[1] = (240 - 20*h) / 2; // offsetY

  ...
  ...
  ...
```

```
for (int y = 0; y < h; y++){
  for (int x = 0; x < w; x++){
    uint8_t value = arr[y][x];

    if ( arr[y][x] == 6 ){
      playerX = x;
      playerY = y;
    }
```

Se o valor do pattern for o player: atribuir a posição do player nas variáveis.

```
    if ( gameBoxAmount < 5 && (value == 4 || value == 5) ){
      boxList[gameBoxAmount].active = value == 5;
      boxList[gameBoxAmount].x = x;
      boxList[gameBoxAmount].y = y;
      gameBoxAmount++;
    }
```

Caso for caixa:

1. Atribuir “ative” caso for caixa escura
2. Atribuir a posição
3. Incrementar quantidade

```
    if ( value != 0 ){
      renderBitmap(20*x+offsetMap[0], 20*y+offsetMap[1], 20, 20, spriteList[value-1]);
    }

  }
}
```

Caso **não** for espaço vazio: renderize o sprite

```
void generateLevelByIndex(uint8_t levelIndex)
```

generateLevelByIndex() gera o mapa pelo index da fase

```
void generateLevelByIndex(uint8_t levelIndex){
  if ( levelIndex == 0 ){
    generateLevel(6, 7, level0);
    return;
  }
  ...
  ...
  ...
  if ( levelIndex == 19 ){
    generateLevel(9, 8, level19);
    return;
  }
}
```

void updateLevel(uint8_t direction)

updateLevel() atualiza pedaços do mapa quando o jogador clicar nas teclas direcionais

```
void updateLevel(uint8_t direction){


  updateBox(currentLevelIndex);


  uint8_t (*mapa)[currentLevelHeight][currentLevelWidth] = (uint8_t (*)[currentLevelHeight][currentLevelWidth]) getAddress(currentLevelIndex);

  if ( (*mapa)[playerY][playerX] == 3 || (*mapa)[playerY][playerX] == 5 ){
    renderBitmap(20*playerX+offsetMap[0], 20*playerY+offsetMap[1], 20, 20, spriteList[2]);
  }else{
    renderBitmap(20*playerX+offsetMap[0], 20*playerY+offsetMap[1], 20, 20, spriteList[1]);
  }
  ...

  ...

  ...
```

```
// UP
if ( direction == 1 ){
  int8_t boxIndex = hasBoxAt(playerX, playerY-1);

  if ( playerY > 0 ){
    if ( boxIndex == -1 ){
      if ( (*mapa)[playerY-1][playerX] > 1 ){
        playerY--;
        currentMovementCount++;
      }
    }else if ( playerY >= 3 ){
      if ( boxIndex >= 0 && hasBoxAt(playerX, playerY-2) == -1 && (*mapa)[playerY-2][playerX] > 1 ){
        boxList[boxIndex].y--;
        playerY--;
        currentMovementCount++;
      }
    }
  }
}

// DOWN
else if ( direction == 2 ){
  int8_t boxIndex = hasBoxAt(playerX, playerY+1);

  if ( playerY < currentLevelHeight-1 ){
    if ( boxIndex == -1 ){
      if ( (*mapa)[playerY+1][playerX] > 1 ){
        playerY++;
        currentMovementCount++;
      }
    }else if ( playerY < currentLevelHeight-2-1 ){
      if ( boxIndex >= 0 && hasBoxAt(playerX, playerY+2) == -1 && (*mapa)[playerY+2][playerX] > 1 ){
        boxList[boxIndex].y++;
        playerY++;
        currentMovementCount++;
      }
    }
  }
}

...
...
...
```

Caso tiver um espaço vazio na próxima posição:
Mover jogador.

Caso tiver uma caixa na próxima posição e um espaço vazio depois da caixa:
Mover caixa.

```
  // LEFT
  else if ( direction == 3 ){
    int8_t boxIndex = hasBoxAt(playerX-1, playerY);

    if ( playerX > 0 ){
      if ( boxIndex == -1 ){
        if ( (*mapa)[playerY][playerX-1] > 1 ){
          playerX--;
          currentMovementCount++;
        }
      }else if ( playerX >= 3 ){
        if ( boxIndex >= 0 && hasBoxAt(playerX-2, playerY) == -1 && (*mapa)[playerY][playerX-2] > 1 ){
          boxList[boxIndex].x--;
          playerX--;
          currentMovementCount++;
        }
      }
    }
  }

  // RIGHT
  else if ( direction == 4 ){
    int8_t boxIndex = hasBoxAt(playerX+1, playerY);

    if ( playerX < currentLevelWidth-1 ){
      if ( boxIndex == -1 ){
        if ( (*mapa)[playerY][playerX+1] > 1 ){
          playerX++;
          currentMovementCount++;
        }
      }else if ( playerX < currentLevelWidth-2-1 ){
        if ( boxIndex >= 0 && hasBoxAt(playerX+2, playerY) == -1 && (*mapa)[playerY][playerX+2] > 1 ){
          boxList[boxIndex].x++;
          playerX++;
          currentMovementCount++;
        }
      }
    }
  }


  // UPDATE
  updateBox(currentLevelIndex);
  renderBitmap(20*playerX+offsetMap[0], 20*playerY+offsetMap[1], 20, 20, spriteList[5]);

}
```

1. Atualizar posição das caixas (trocar sprite caso nessesário)
2. Atualizar posição do jogador

```
void tft_init()
```

`tft_init()` Inicializa o LCD

```
void tft_fillRect(int16_t x, int16_t y, int16_t w, int16_t h, uint16_t color)
```

`tft_fillRect()` Cria um quadrado na posição x,y com tamanho w,h e cor.

```
void renderBitmap(uint16_t x, uint16_t y, uint16_t w, uint16_t h, uint16_t *bitmap)
```

`renderBitmap()` Renderiza um bitmap na posição x,y com tamanho do bitmap fixo em w,y e um ponteiro do array

Obs.: O w,h não faz o bitmap ficar com o tamanho diferente do estipulado no array.

Ex.: Se o bitmap for 16x8 com 16bit será: `uint16_t image[128];`
w <u>não</u> pode ser diferente de `16`
h <u>não</u> pode ser diferente de `8`

```
renderBitmap(0, 0, 16, 8, image);
```

```
void main(void)
```

`main()` Função principal do programa.

```
void main(void){
  HAL_Init();

  SystemClock_Config();

  MX_GPIO_Init();
  MX_SPI1_Init();
  MX_USART2_UART_Init();

  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_RESET); // SCK LOW STATE
  HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_SET);
  tft_init();
  tft_fillRect(0, 0, 240, 240, 0x0000);
  showMenu();

  ...
  ...
  ...
```

Caso necessário debug via serial

```
while (1){
  if ( !keyLocked ){
    if ( !HAL_GPIO_ReadPin(KEY_UP_GPIO_Port, KEY_UP_Pin) ){
      if ( !keyPressed ){
        keyPressed = 1;
        movementDirection = 1;
        HAL_Delay(5);
      }
    }else if ( !HAL_GPIO_ReadPin(KEY_DOWN_GPIO_Port, KEY_DOWN_Pin) ){
        if ( !keyPressed ){
          keyPressed = 1;
          movementDirection = 2;
          HAL_Delay(5);
        }
      }else if ( !HAL_GPIO_ReadPin(KEY_LEFT_GPIO_Port, KEY_LEFT_Pin) ){
        if ( !keyPressed ){
          keyPressed = 1;
          movementDirection = 3;
          HAL_Delay(5);
        }
      }else if ( !HAL_GPIO_ReadPin(KEY_RIGHT_GPIO_Port, KEY_RIGHT_Pin) ){
        if ( !keyPressed ){
          keyPressed = 1;
          movementDirection = 4;
          HAL_Delay(5);
        }
      }
...
...
...
```

Verifica se a tecla está pressionada e faz o debounde usando **keyPressed.**

```
else if ( !HAL_GPIO_ReadPin(KEY_ENTER_GPIO_Port, KEY_ENTER_Pin) ){
  if ( !keyPressed ){
    keyPressed = 1;

    if ( isMenuMode ){
      tft_fillRect(0, 0, 240, 240, 0x0000);
      generateLevelByIndex(currentLevelIndex);
    }else{
      tft_fillRect(0, 0, 240, 240, 0x0000);
      showMenu();
    }
    isMenuMode = !isMenuMode;
    HAL_Delay(5);
  }
}

...
...
...
```

Caso apertar “Enter” no modo menu: entrar na fase **n**

Caso contrário: Abrir o menu

Toggle do estado do menu

```
else if (!HAL_GPIO_ReadPin(KEY_RESTART_GPIO_Port, KEY_RESTART_Pin)){
  if ( !keyPressed ){
    keyPressed = 1;
    if ( !isMenuMode ){
      tft_fillRect(0, 0, 240, 240, 0x0000);
      generateLevelByIndex(currentLevelIndex);
    }
    HAL_Delay(5);
  }
}else{
  movementDirection = 0;
  keyPressed = 0;
}
}
```

Caso apertar “Restart” e estiver no modo jogo: Reiniciar fase

Caso soltar a tecla:

1. Movement é 0 (nenhuma)
2. keyPressed é “false”

...
...
...

```
if ( movementDirection != 0 ){
  if ( isMenuMode ){
    if ( movementDirection == 3 ){
      if (currentLevelIndex > 0){
        currentLevelIndex--;
        showMenu();
      }else{
        currentLevelIndex = 20 - 1;
      }
    }else if ( movementDirection == 4 ){
      currentLevelIndex = (currentLevelIndex + 1) % 20;
      showMenu();
    }
    movementDirection = 0;
    continue;
  }
```

...
...
...

Caso clicar nas teclas direcionais e estiver no modo menu:

Apertar Left: decrementa fase
Apertar Right: incrementa fase
E pula o loop com **continue**

```
updateLevel(movementDirection);

if ( gameWin() ){
  keyLocked = 1;
  keyPressed = 0;
  showSummary();
  HAL_Delay(1000);
  currentLevelIndex = (currentLevelIndex + 1) % 20;
  if ( currentLevelIndex == 0 ){
    tft_drawText(30, 100, "END :)", 0xFFFF, 0x0000, 5);
    HAL_Delay(1000);
    isMenuMode = 1;
    tft_fillRect(0, 0, 240, 240, 0x0000);
    showMenu();
  }else{
    showCurrentLevel();
    HAL_Delay(1000);
    tft_fillRect(0, 0, 240, 240, 0x0000);
    generateLevelByIndex(currentLevelIndex);
  }

    keyLocked = 0;
}

movementDirection = 0;
}
```

Caso finalizar a fase

Mostra o Resumo após finalizar.

Caso acabar as fases: ir para o menu

Caso Contrário: Mostrar nível ir para próxima fase

SPRITE 1

```
{
  0x5265, 0x6B27, 0x6B27, 0x6B27, 0x6B27, 0x6B27, 0x6B27, 0x6B27, 0x6B27, 0x6306, 0x5285, 0x6B27, 0x6B27, 0x6B27, 0x6B27, 0x6B27, // 0x0010 (16) pixels
  0x6B27, 0x6B27, 0x6B27, 0x6B27, 0x6B27, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0x9449, 0x7347, 0xA4AA, // 0x0020 (32) pixels
  0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0x6B27, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, // 0x0030 (48) pixels
  0xA4AA, 0x9449, 0x7347, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0x6B27, 0xA4AA, 0xA4AA, 0xA4AA, // 0x0040 (64) pixels
  0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0x9449, 0x7347, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, // 0x0050 (80) pixels
  0x62E6, 0x9449, 0x9449, 0x9449, 0x9449, 0x946A, 0x9449, 0x9449, 0x9449, 0x8C09, 0x6B07, 0x9449, 0x9449, 0x9449, 0x9449, 0x946A, // 0x0060 (96) pixels
  0x9449, 0x9449, 0x9449, 0x9449, 0x6B27, 0x7347, 0x7347, 0x7347, 0x6B27, 0x5AC6, 0x7347, 0x7347, 0x7347, 0x7347, 0x6B27, 0x7347, // 0x0070 (112) pixels
  0x7347, 0x7347, 0x6B27, 0x5AC6, 0x7347, 0x7347, 0x7347, 0x7347, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0x9449, 0x7347, 0xA4AA, 0xA4AA, // 0x0080 (128) pixels
  0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0x9449, 0x7347, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, // 0x0090 (144) pixels
  0x9449, 0x7347, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0x9449, 0x7347, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, // 0x00A0 (160) pixels
  0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0x9449, 0x7347, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0x9449, 0x7347, // 0x00B0 (176) pixels
  0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0x946A, 0x9449, 0x9449, 0x9449, 0x8C09, 0x6B07, 0x9449, 0x9449, 0x9449, 0x9449, 0x946A, 0x9449, // 0x00C0 (192) pixels
  0x9449, 0x9449, 0x8C09, 0x6B07, 0x9449, 0x9449, 0x9449, 0x9449, 0x5AA6, 0x7347, 0x7347, 0x7347, 0x7347, 0x6B27, 0x7347, 0x7347, // 0x00D0 (208) pixels
  0x7347, 0x6B27, 0x5AC6, 0x7347, 0x7347, 0x7347, 0x7347, 0x6B27, 0x7347, 0x7347, 0x7347, 0x7347, 0x6B27, 0xA4AA, 0xA4AA, 0xA4AA, // 0x00E0 (224) pixels
  0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0x9449, 0x7347, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, // 0x00F0 (240) pixels
  0x6B27, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0x9449, 0x7347, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, // 0x0100 (256) pixels
  0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0x6B27, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0x9449, 0x7347, 0xA4AA, // 0x0110 (272) pixels
  0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0x62E6, 0x9449, 0x9449, 0x9449, 0x9449, 0x946A, 0x9449, 0x9449, // 0x0120 (288) pixels
  0x9449, 0x8C09, 0x6B07, 0x9449, 0x9449, 0x9449, 0x9449, 0x946A, 0x9449, 0x9449, 0x9449, 0x9449, 0x6B27, 0x7347, 0x7347, 0x7347, // 0x0130 (304) pixels
  0x6B27, 0x5AC6, 0x7347, 0x7347, 0x7347, 0x7347, 0x6B27, 0x7347, 0x7347, 0x7347, 0x6B27, 0x5AC6, 0x7347, 0x7347, 0x7347, 0x7347, // 0x0140 (320) pixels
  0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0x9449, 0x7347, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0x9449, 0x7347, // 0x0150 (336) pixels
  0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0x9449, 0x7347, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, // 0x0160 (352) pixels
  0xA4AA, 0xA4AA, 0x9449, 0x7347, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0x9449, 0x7347, 0xA4AA, 0xA4AA, // 0x0170 (368) pixels
  0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0x9449, 0x7347, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, // 0x0180 (384) pixels
  0x9449, 0x7347, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, 0x9449, 0x7347, 0xA4AA, 0xA4AA, 0xA4AA, 0xA4AA, // 0x0190 (400) pixels
},
```

SPRITE 2

```
{
  0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xD674, 0xD653, 0xD654, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xD674, 0xD653, // 0x0010 (16) pixels
  0xD654, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDE95, 0xDE95, 0xDEB5, 0xDEB5, 0xDE95, 0xD654, // 0x0020 (32) pixels
  0xD674, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xD674, 0xD654, 0xD674, 0xDEB5, 0xDEB5, 0xD674, 0xD654, // 0x0030 (48) pixels
  0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDE95, 0xDE95, 0xDE95, // 0x0040 (64) pixels
  0xD674, 0xDE95, 0xDEB5, 0xDEB5, 0xDEB5, 0xD674, 0xDE95, 0xDEB5, 0xDEB5, 0xDE95, 0xD654, 0xD654, 0xDE95, 0xDEB5, 0xDEB5, 0xDEB5, // 0x0050 (80) pixels
  0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xD674, 0xDE95, 0xDEB5, 0xDEB5, 0xDEB5, 0xD674, 0xDE95, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, // 0x0060 (96) pixels
  0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, // 0x0070 (112) pixels
  0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDE95, 0xD674, 0xD674, 0xDEB5, 0xDEB5, 0xDEB5, 0xDE95, 0xD654, 0xD654, 0xDE95, 0xDEB5, // 0x0080 (128) pixels
  0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xD674, 0xD654, 0xDE95, 0xDE95, 0xD674, 0xD674, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, // 0x0090 (144) pixels
  0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDE95, 0xD654, 0xD654, 0xD674, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, // 0x00A0 (160) pixels
  0xD674, 0xDE95, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDE95, 0xDE95, 0xDE95, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, // 0x00B0 (176) pixels
  0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xD674, 0xD674, 0xDEB5, 0xD674, 0xD654, 0xD674, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, // 0x00C0 (192) pixels
  0xDE95, 0xD654, 0xD654, 0xDEB5, 0xDEB5, 0xDE95, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDE95, 0xDEB5, 0xDEB5, 0xDEB5, // 0x00D0 (208) pixels
  0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDE95, 0xDE95, 0xDEB5, 0xDEB5, 0xD654, 0xD674, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, // 0x00E0 (224) pixels
  0xDEB5, 0xDEB5, 0xDEB5, 0xDE95, 0xDE95, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, // 0x00F0 (240) pixels
  0xDEB5, 0xD674, 0xD654, 0xD674, 0xDEB5, 0xDEB5, 0xDEB5, 0xD654, 0xD674, 0xDEB5, 0xDEB5, 0xDEB5, 0xD654, 0xD654, 0xDE95, 0xDEB5, // 0x0100 (256) pixels
  0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDE95, 0xDE95, 0xDE95, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, // 0x0110 (272) pixels
  0xDE95, 0xDE95, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDE95, 0xDE95, 0xDEB5, 0xDEB5, 0xDEB5, 0xDE95, 0xDE95, 0xDE95, // 0x0120 (288) pixels
  0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDE95, 0xDE95, 0xDEB5, 0xDEB5, 0xDEB5, 0xD654, 0xD654, 0xDE95, 0xDEB5, // 0x0130 (304) pixels
  0xDEB5, 0xD654, 0xD654, 0xD654, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xD654, 0xD654, 0xDE95, 0xDEB5, // 0x0140 (320) pixels
  0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xD674, 0xD654, 0xDEB5, 0xDEB5, 0xDEB5, // 0x0150 (336) pixels
  0xDEB5, 0xDEB5, 0xDE95, 0xDE95, 0xDEB5, 0xDEB5, 0xD654, 0xD654, 0xDE95, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, // 0x0160 (352) pixels
  0xDEB5, 0xDEB5, 0xDEB5, 0xD674, 0xD674, 0xDEB5, 0xD674, 0xD654, 0xDEB5, 0xDEB5, 0xDE95, 0xDE95, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, // 0x0170 (368) pixels
  0xDE95, 0xD654, 0xD654, 0xDE95, 0xDEB5, 0xDEB5, 0xDEB5, 0xDE95, 0xDE95, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, // 0x0180 (384) pixels
  0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, // 0x0190 (400) pixels
},
```

SPRITE 3

```
{
  0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xD674, 0xD653, 0xD654, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xD674, 0xD653, // 0x0010 (16) pixels
  0xD654, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDE95, 0xDE95, 0xDEB5, 0xDEB5, 0xDE95, 0xD654, // 0x0020 (32) pixels
  0xD674, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xD674, 0xD654, 0xD674, 0xDEB5, 0xDEB5, 0xD674, 0xD654, // 0x0030 (48) pixels
  0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDE95, 0xDE95, 0xDE95, // 0x0040 (64) pixels
  0xD674, 0xDE95, 0xDEB5, 0xDEB5, 0xDEB5, 0xD674, 0xDE95, 0xDEB5, 0xDEB5, 0xDE95, 0xD654, 0xD654, 0xDE95, 0xDEB5, 0xDEB5, 0xDEB5, // 0x0050 (80) pixels
  0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xD674, 0xDE95, 0xDEB5, 0xDEB5, 0xDEB5, 0xD674, 0xDE95, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, // 0x0060 (96) pixels
  0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, // 0x0070 (112) pixels
  0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDE95, 0xD674, 0xD674, 0xDEB5, 0xDEB5, 0xDE95, 0xD654, 0xD654, 0xDE95, 0xDEB5, // 0x0080 (128) pixels
  0xDE34, 0xDD72, 0xDD32, 0xDDB3, 0xDE95, 0xDEB5, 0xD674, 0xD654, 0xDE95, 0xDE95, 0xD674, 0xD674, 0xDEB5, 0xDEB5, 0xDEB5, // 0x0090 (144) pixels
  0xDEB5, 0xDEB5, 0xDEB5, 0xDE34, 0xD4F1, 0xD4B0, 0xD4B0, 0xD4B0, 0xDD52, 0xDE95, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, // 0x00A0 (160) pixels
  0xD674, 0xDE95, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDE75, 0xDD12, 0xD4B0, 0xD4B0, 0xD4B0, 0xD4B0, 0xD4B0, 0xDDB3, 0xDEB5, 0xDEB5, // 0x00B0 (176) pixels
  0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xD674, 0xD674, 0xDEB5, 0xD674, 0xD654, 0xD674, 0xDE54, 0xD4B0, 0xD4B0, 0xD4B0, 0xD4B0, 0xD4B0, // 0x00C0 (192) pixels
  0xD4B0, 0xD511, 0xD654, 0xDEB5, 0xDEB5, 0xDE95, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDE95, 0xDEB5, 0xDE75, 0xDCF1, // 0x00D0 (208) pixels
  0xD4B0, 0xD4B0, 0xD4B0, 0xD4B0, 0xD4B0, 0xDD52, 0xDE95, 0xDEB5, 0xDEB5, 0xD654, 0xD674, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, // 0x00E0 (224) pixels
  0xDEB5, 0xDEB5, 0xDEB5, 0xD592, 0xD4B0, 0xD4B0, 0xD4B0, 0xD4B0, 0xD4F1, 0xDE34, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, // 0x00F0 (240) pixels
  0xDEB5, 0xD674, 0xD654, 0xD674, 0xDEB5, 0xDEB5, 0xDEB5, 0xD654, 0xDDB3, 0xDCF1, 0xD4B0, 0xD511, 0xD5D3, 0xD654, 0xDE95, 0xDEB5, // 0x0100 (256) pixels
  0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDE95, 0xDE95, 0xDE95, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDE75, 0xDE54, 0xDE75, // 0x0110 (272) pixels
  0xDE95, 0xDE95, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDE95, 0xDE95, 0xDEB5, 0xDEB5, 0xDEB5, 0xDE95, 0xDE95, 0xDE95, // 0x0120 (288) pixels
  0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDE95, 0xDE95, 0xDEB5, 0xDEB5, 0xDEB5, 0xD654, 0xD654, 0xDE95, 0xDEB5, // 0x0130 (304) pixels
  0xDEB5, 0xD654, 0xD654, 0xD654, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xD654, 0xD654, 0xDE95, 0xDEB5, 0xDEB5, // 0x0140 (320) pixels
  0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xD674, 0xD654, 0xDEB5, 0xDEB5, 0xDEB5, // 0x0150 (336) pixels
  0xDEB5, 0xDEB5, 0xDE95, 0xDE95, 0xDEB5, 0xDEB5, 0xD654, 0xD654, 0xDE95, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, // 0x0160 (352) pixels
  0xDEB5, 0xDEB5, 0xDEB5, 0xD674, 0xD674, 0xDEB5, 0xD674, 0xD654, 0xDEB5, 0xDEB5, 0xDE95, 0xDE95, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, // 0x0170 (368) pixels
  0xDE95, 0xD654, 0xD654, 0xDE95, 0xDEB5, 0xDEB5, 0xDEB5, 0xDE95, 0xDE95, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, // 0x0180 (384) pixels
  0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, // 0x0190 (400) pixels
},
```

SPRITE 4

```
{
  0xB571, 0x83CC, 0x83EC, 0x83EC, 0x83EC, 0x83EC, 0x83EC, 0x83EC, 0x83EC, 0x83EC, 0x83EC, 0x83EC, 0x83EC, 0x83EC, 0x83EC, 0x83EC, // 0x0010 (16) pixels
  0x83EC, 0x83EC, 0x83EC, 0xB571, 0x83CC, 0x836A, 0xBCCD, 0xB4AD, 0xB4AD, 0xB4AD, 0xB4AD, 0xB4AD, 0xB4AD, 0xB4AD, 0xB4AD, 0xB4AD, // 0x0020 (32) pixels
  0xB4AD, 0xB4AD, 0xB4AD, 0xB4AD, 0xB4AD, 0xBCAD, 0x7AC5, 0x83CC, 0x83EC, 0xBCCD, 0xFE0B, 0xFDCA, 0xFDEA, 0xFDEA, 0xFDEA, 0xFDEA, // 0x0030 (48) pixels
  0xFDEA, 0xFDEA, 0xFDEA, 0xFDEA, 0xFDEA, 0xFDEA, 0xFDEA, 0xFDEA, 0xFDCA, 0xFDEA, 0xA322, 0x83CC, 0x83EC, 0xB4AD, 0xFDCA, 0xD423, // 0x0040 (64) pixels
  0xA301, 0xA301, 0xA321, 0xA301, 0xA301, 0xA301, 0xA301, 0xA301, 0xA301, 0xA321, 0xA301, 0xA321, 0xE529, 0xFDCA, 0xA301, 0x83CC, // 0x0050 (80) pixels
  0x83EC, 0xB4AD, 0xFDEA, 0xA301, 0x5180, 0x61E1, 0x59A0, 0x8A60, 0x8260, 0x8260, 0x8260, 0x8260, 0x8A60, 0x59A0, 0x61E1, 0x5980, // 0x0060 (96) pixels
  0xB4AD, 0xFDEA, 0xA301, 0x83CC, 0x83EC, 0xB4AD, 0xFDEA, 0xA301, 0x8AA0, 0xFD8A, 0x9BA8, 0xB320, 0xEC83, 0xECA3, 0xECA3, 0xF4C3, // 0x0070 (112) pixels
  0xBB82, 0x9BA8, 0xFD8A, 0x8AA1, 0xB4AD, 0xFDEA, 0xA301, 0x83CC, 0x83EC, 0xB4AD, 0xFDEA, 0xA301, 0x7220, 0xFD68, 0xF5CC, 0x9388, // 0x0080 (128) pixels
  0xBBE4, 0xFDCA, 0xFDCA, 0xCC88, 0x9BEA, 0xF5CC, 0xCC24, 0x4140, 0xB4AD, 0xFDEA, 0xA301, 0x83CC, 0x83EC, 0xB4AD, 0xFDEA, 0xA321, // 0x0090 (144) pixels
  0x4140, 0xA322, 0xFD68, 0xF5CC, 0x9BC9, 0xCC88, 0xCC88, 0x9BCA, 0xFDCC, 0xCC24, 0x8260, 0x7220, 0xB4AD, 0xFDEA, 0xA301, 0x83CC, // 0x00A0 (160) pixels
  0x83EC, 0xB4AD, 0xFDEA, 0xA301, 0x8260, 0x9AC0, 0x9B02, 0xFD68, 0xF5CC, 0x8B89, 0x6AA6, 0xF5CC, 0xCC24, 0x8260, 0xE4A5, 0x8AA1, // 0x00B0 (176) pixels
  0xB4AD, 0xFDEA, 0xA301, 0x83CC, 0x83EC, 0xB4AD, 0xFDEA, 0xA301, 0x8260, 0xEC62, 0x9B02, 0xA322, 0xFD68, 0xF5CC, 0x9BCA, 0xA343, // 0x00C0 (192) pixels
  0x8260, 0xE4C6, 0xFDA9, 0x8AA1, 0xB4AD, 0xFDEA, 0xA301, 0x83CC, 0x83EC, 0xB4AD, 0xFDEA, 0xA301, 0x8260, 0xECA3, 0xFDA9, 0x7AA4, // 0x00D0 (208) pixels
  0x8AA1, 0xFD68, 0xF5CC, 0x8B68, 0xB3A4, 0xFDCA, 0xFD89, 0x8AA1, 0xB4AD, 0xFDEA, 0xA301, 0x83CC, 0x83EC, 0xB4AD, 0xFDEA, 0xA301, // 0x00E0 (224) pixels
  0x8260, 0xF4C3, 0xCC88, 0x9BC9, 0x9303, 0xA302, 0xFD68, 0xF5CC, 0x9BEA, 0xCC88, 0xFDA9, 0x8AA1, 0xB4AD, 0xFDEA, 0xA301, 0x83CC, // 0x00F0 (240) pixels
  0x83EC, 0xB4AD, 0xFDEA, 0xA301, 0x8A60, 0xBB82, 0x9BEA, 0xFDCC, 0xCC04, 0x69E0, 0xA322, 0xFD68, 0xF5CC, 0x9BCA, 0xCC67, 0x8AA1, // 0x0100 (256) pixels
  0xB4AD, 0xFDEA, 0xA301, 0x83CC, 0x83EC, 0xB4AD, 0xFDEA, 0xA321, 0x59A0, 0x9BA8, 0xF5CC, 0xCC24, 0x8260, 0xDCA6, 0xABC6, 0xA322, // 0x0110 (272) pixels
  0xFD68, 0xF5CC, 0x9BA7, 0x59A0, 0xB4AD, 0xFDEA, 0xA301, 0x83CC, 0x83EC, 0xB4AD, 0xFDEA, 0xA301, 0x61E1, 0xFD8A, 0xCC24, 0x8260, // 0x0120 (288) pixels
  0xE4A5, 0xFDA9, 0xFDA9, 0xABA5, 0xA322, 0xFD68, 0xFD48, 0x61E1, 0xB4AD, 0xFDEA, 0xA301, 0x83CC, 0x83EC, 0xB4AD, 0xFDEA, 0xA321, // 0x0130 (304) pixels
  0x5980, 0x8AA1, 0x4140, 0x7220, 0x8AA1, 0x8AA1, 0x8AA1, 0x8AA1, 0x4940, 0x7221, 0x92A1, 0x59A0, 0xBCAD, 0xFDEA, 0xA301, 0x83CC, // 0x0140 (320) pixels
  0x83EC, 0xB4AD, 0xFDCA, 0xE529, 0xB4AD, 0xB4AD, 0xB4AD, 0xB4AD, 0xB4AD, 0xB4AD, 0xB4AD, 0xB4AD, 0xBCAD, 0xB4AD, 0xB4AD, 0xBCAD, // 0x0150 (336) pixels
  0xEDF0, 0xFDCA, 0xA301, 0x83CC, 0x83EC, 0xBCAD, 0xFDEA, 0xFDCA, 0xFDEA, 0xFDEA, 0xFDEA, 0xFDEA, 0xFDEA, 0xFDEA, 0xFDEA, 0xFDEA, // 0x0160 (352) pixels
  0xFDEA, 0xFDEA, 0xFDEA, 0xFDEA, 0xFDCA, 0xFDA9, 0xA321, 0x83CC, 0x83EC, 0x7AC5, 0xA322, 0xA301, 0xA301, 0xA301, 0xA301, 0xA301, // 0x0170 (368) pixels
  0xA301, 0xA301, 0xA301, 0xA301, 0xA301, 0xA301, 0xA301, 0xA301, 0xA301, 0xA321, 0x7200, 0x83CC, 0xB571, 0x83CC, 0x83CC, 0x83CC, // 0x0180 (384) pixels
  0x83CC, 0x83CC, 0x83CC, 0x83CC, 0x83CC, 0x83CC, 0x83CC, 0x83CC, 0x83CC, 0x83CC, 0x83CC, 0x83CC, 0x83CC, 0x83CC, 0x83CC, 0xB571, // 0x0190 (400) pixels
},
```

SPRITE 5

```
{
  0xB571, 0x83CC, 0x83EC, 0x83EC, 0x83EC, 0x83EC, 0x83EC, 0x83EC, 0x83EC, 0x83EC, 0x83EC, 0x83EC, 0x83EC, 0x83EC, 0x83EC, 0x83EC, // 0x0010 (16) pixels
  0x83EC, 0x83EC, 0x83CC, 0xB571, 0x83CC, 0x5A24, 0x82E6, 0x82E5, 0x82E5, 0x82E5, 0x82E5, 0x82E5, 0x82E5, 0x82E5, 0x82E5, 0x82E5, // 0x0020 (32) pixels
  0x82E5, 0x82E5, 0x82E5, 0x82E5, 0x82E5, 0x82E5, 0x5981, 0x7BCC, 0x83EC, 0x82E6, 0xC362, 0xBB42, 0xBB42, 0xBB42, 0xBB42, 0xBB42, // 0x0030 (48) pixels
  0xBB42, 0xBB42, 0xBB42, 0xBB42, 0xBB42, 0xBB42, 0xBB42, 0xBB42, 0xBB42, 0xBB42, 0x7140, 0x83CC, 0x83EC, 0x82E5, 0xBB42, 0x91C0, // 0x0040 (64) pixels
  0x6940, 0x6940, 0x6940, 0x6940, 0x6940, 0x6940, 0x6940, 0x6940, 0x6940, 0x6940, 0x6940, 0x6940, 0xA2C2, 0xBB21, 0x6940, 0x83CC, // 0x0050 (80) pixels
  0x83EC, 0x82E5, 0xBB42, 0x6940, 0x3880, 0x40C0, 0x3880, 0x58C0, 0x50C0, 0x50C0, 0x50C0, 0x50C0, 0x58C0, 0x3880, 0x40C0, 0x3880, // 0x0060 (96) pixels
  0x82E6, 0xBB42, 0x6940, 0x83CC, 0x83EC, 0x82E5, 0xBB42, 0x6940, 0x5900, 0xB303, 0x6A24, 0x7120, 0xA1E0, 0xA200, 0xA200, 0xA220, // 0x0070 (112) pixels
  0x7960, 0x6A24, 0xB303, 0x5900, 0x82E6, 0xBB42, 0x6940, 0x83CC, 0x83EC, 0x82E5, 0xBB42, 0x6940, 0x48C0, 0xB2C1, 0xB363, 0x6A04, // 0x0080 (128) pixels
  0x81E0, 0xBB21, 0xBB21, 0x9281, 0x7264, 0xB343, 0x8A00, 0x2880, 0x82E6, 0xBB42, 0x6940, 0x83CC, 0x83EC, 0x82E5, 0xBB42, 0x6940, // 0x0090 (144) pixels
  0x2860, 0x6940, 0xB2E1, 0xB363, 0x6A44, 0x9281, 0x9281, 0x7264, 0xB363, 0x89E0, 0x50C0, 0x48C0, 0x82E6, 0xBB42, 0x6940, 0x83CC, // 0x00A0 (160) pixels
  0x83EC, 0x82E5, 0xBB42, 0x6940, 0x50C0, 0x60E0, 0x6940, 0xB2E1, 0xB363, 0x6224, 0x51A2, 0xB343, 0x89E0, 0x50E0, 0x9A40, 0x5900, // 0x00B0 (176) pixels
  0x82E6, 0xBB42, 0x6940, 0x83CC, 0x83EC, 0x82E5, 0xBB42, 0x6940, 0x50C0, 0x99C0, 0x6960, 0x6960, 0xB2E1, 0xB363, 0x7264, 0x7180, // 0x00C0 (192) pixels
  0x58E0, 0x9A61, 0xBB01, 0x5900, 0x82E6, 0xBB42, 0x6940, 0x83CC, 0x83EC, 0x82E5, 0xBB42, 0x6940, 0x50C0, 0xA200, 0xBB21, 0x5960, // 0x00D0 (208) pixels
  0x5920, 0xB2E1, 0xB363, 0x6203, 0x79C0, 0xBB21, 0xBB01, 0x5900, 0x82E6, 0xBB42, 0x6940, 0x83CC, 0x83EC, 0x82E5, 0xBB42, 0x6940, // 0x00E0 (224) pixels
  0x50C0, 0xA220, 0x9281, 0x6A44, 0x6161, 0x6940, 0xB2E1, 0xB363, 0x7264, 0x9281, 0xBB01, 0x5900, 0x82E6, 0xBB42, 0x6940, 0x83CC, // 0x00F0 (240) pixels
  0x83EC, 0x82E5, 0xBB42, 0x6940, 0x58C0, 0x7960, 0x7264, 0xB363, 0x89E0, 0x40A0, 0x6940, 0xB2E1, 0xB363, 0x7264, 0x9261, 0x5900, // 0x0100 (256) pixels
  0x82E6, 0xBB42, 0x6940, 0x83CC, 0x83EC, 0x82E5, 0xBB42, 0x6940, 0x3880, 0x6A24, 0xB343, 0x89E0, 0x50E0, 0x9A61, 0x7A00, 0x6960, // 0x0110 (272) pixels
  0xB2E1, 0xB363, 0x7202, 0x38A0, 0x82E6, 0xBB42, 0x6940, 0x83CC, 0x83EC, 0x82E5, 0xBB42, 0x6940, 0x40C0, 0xB303, 0x8A00, 0x50C0, // 0x0120 (288) pixels
  0x9A40, 0xBB01, 0xBB01, 0x79E0, 0x6960, 0xB2E1, 0xB2C1, 0x40C0, 0x82E6, 0xBB42, 0x6940, 0x83CC, 0x83EC, 0x82E5, 0xBB42, 0x6940, // 0x0130 (304) pixels
  0x3880, 0x5900, 0x2880, 0x48C0, 0x5900, 0x5900, 0x5900, 0x5900, 0x2860, 0x48E0, 0x5900, 0x38A0, 0x82E6, 0xBB42, 0x6940, 0x83CC, // 0x0140 (320) pixels
  0x83EC, 0x82E5, 0xBB42, 0xA2C2, 0x82E6, 0x82E6, 0x82E6, 0x82E6, 0x82E6, 0x82E6, 0x82E6, 0x82E6, 0x82E6, 0x82E6, 0x82E6, 0x82E6, // 0x0150 (336) pixels
  0xABA6, 0xBB21, 0x6940, 0x83CC, 0x83EC, 0x82E5, 0xBB42, 0xBB21, 0xBB42, 0xBB42, 0xBB42, 0xBB42, 0xBB42, 0xBB42, 0xBB42, 0xBB42, // 0x0160 (352) pixels
  0xBB42, 0xBB42, 0xBB42, 0xBB42, 0xBB21, 0xBB01, 0x6940, 0x83CC, 0x83CC, 0x5981, 0x7140, 0x6940, 0x6940, 0x6940, 0x6940, 0x6940, // 0x0170 (368) pixels
  0x6940, 0x6940, 0x6940, 0x6940, 0x6940, 0x6940, 0x6940, 0x6940, 0x6940, 0x6940, 0x48A0, 0x7BCC, 0xB571, 0x7BCC, 0x83CC, 0x83CC, // 0x0180 (384) pixels
  0x83CC, 0x83CC, 0x83CC, 0x83CC, 0x83CC, 0x83CC, 0x83CC, 0x83CC, 0x83CC, 0x83CC, 0x83CC, 0x83CC, 0x83CC, 0x83CC, 0x7BCC, 0xB571, // 0x0190 (400) pixels
},
```

SPRITE 6

```
{
  0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xD674, 0xD654, 0x83EC, 0x18C2, 0x0000, 0x0000, 0x0000, 0x0000, 0x18C2, 0x8C4D, 0xD674, 0xD653, // 0x0010 (16) pixels
  0xD654, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xE6D6, 0xB571, 0x18C2, 0x0000, 0x2122, 0x39E4, 0x0000, 0x0000, // 0x0020 (32) pixels
  0x0000, 0x18C2, 0xB571, 0xE6D6, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xD674, 0xD654, 0xD674, 0xE6D6, 0x8C2D, 0x0000, 0x0000, // 0x0030 (48) pixels
  0x7BC8, 0xB58B, 0x2962, 0x0840, 0x0020, 0x0000, 0x8C2D, 0xE6D6, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDE95, 0xDE95, 0xDE95, // 0x0040 (64) pixels
  0xDEB5, 0x83EC, 0x18C1, 0x73C8, 0xEF6F, 0xFFD0, 0xE70E, 0xD68E, 0x9CCA, 0x2101, 0x7BCC, 0xD674, 0xDE95, 0xDEB5, 0xDEB5, 0xDEB5, // 0x0050 (80) pixels
  0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDE95, 0x8C2D, 0x6346, 0xBDCC, 0x7387, 0xCE6D, 0xCE4D, 0x7388, 0x9CEB, 0x5B06, 0x944E, 0xE6D6, // 0x0060 (96) pixels
  0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xD654, 0x6328, 0xD6CE, 0xC60F, 0xEF50, 0xDEEE, 0xC5EF, // 0x0070 (112) pixels
  0xC60D, 0x62E7, 0xD654, 0xDEB5, 0xDEB5, 0xDE95, 0xD674, 0xD674, 0xDEB5, 0xDEB5, 0xDEB5, 0xDE95, 0xD654, 0xD654, 0x5AA8, 0x73A7, // 0x0080 (128) pixels
  0xF78F, 0xF7B0, 0xF7B0, 0xEF50, 0x7387, 0x5AC8, 0xD674, 0xD654, 0xDE95, 0xDE95, 0xD674, 0xD674, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, // 0x0090 (144) pixels
  0xDE95, 0xAD30, 0x6B4B, 0x5AEB, 0x6B68, 0x7387, 0x6B87, 0x6B48, 0x4A69, 0x6B2A, 0xAD31, 0xDE95, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, // 0x00A0 (160) pixels
  0xD674, 0xDE95, 0xE6D5, 0xCE34, 0x738C, 0x7BCE, 0xB5B6, 0xD69A, 0xB5B6, 0xA534, 0x8431, 0x8C51, 0xB596, 0xAD55, 0x7BCE, 0x738C, // 0x00B0 (176) pixels
  0xCE34, 0xE6D6, 0xDEB5, 0xDEB5, 0xD674, 0xD674, 0xCE34, 0x7BCD, 0xAD75, 0xDEDB, 0xDEDB, 0xD6BA, 0xDEFB, 0xC638, 0xBDF7, 0xD6BA, // 0x00C0 (192) pixels
  0xD69A, 0xDEFB, 0xDEDB, 0xAD75, 0x7BAD, 0xCE34, 0xDEB5, 0xDEB5, 0xDEB5, 0xE6D6, 0x7BCC, 0x8C51, 0xDEDB, 0xD6BA, 0xA514, 0xCE79, // 0x00D0 (208) pixels
  0xDEFB, 0xC638, 0xC638, 0xDEDB, 0xCE59, 0xAD55, 0xD6BA, 0xD6BA, 0x8C51, 0x738B, 0xDE95, 0xDEB5, 0xDEB5, 0xDE95, 0x7BAB, 0xCE4F, // 0x00E0 (224) pixels
  0xE716, 0x9492, 0x632C, 0xADF9, 0xCE38, 0xC5D6, 0xC5D6, 0xCE38, 0xA5B8, 0x630C, 0x9CD3, 0xE737, 0xCE4F, 0x73AA, 0xDE95, 0xDEB5, // 0x00F0 (240) pixels
  0xE6D6, 0xAD51, 0x8429, 0xFFF0, 0xD6AE, 0x4A47, 0x426A, 0x23FA, 0x6B4D, 0x8349, 0x8329, 0x6B4D, 0x23FA, 0x424A, 0x4A27, 0xEF50, // 0x0100 (256) pixels
  0xFFD0, 0x7387, 0xB572, 0xE6D6, 0xDEB5, 0xD674, 0x946D, 0x9489, 0x842A, 0x6309, 0x1291, 0x13DB, 0x1B97, 0x1335, 0x1B76, 0x1B97, // 0x0110 (272) pixels
  0x13DB, 0x1291, 0x2123, 0x7BC9, 0x9489, 0x946D, 0xDE95, 0xDEB5, 0xDE95, 0xDEB5, 0xDE95, 0x948E, 0xC5D2, 0x8C2D, 0x1B56, 0x2CBF, // 0x0120 (288) pixels
  0x24BF, 0x13DB, 0x247E, 0x24BF, 0x2CBF, 0x1B56, 0x8C2D, 0xBDB2, 0x946E, 0xDE95, 0xDEB5, 0xDEB5, 0xD654, 0xD654, 0xDE95, 0xE6D6, // 0x0130 (304) pixels
  0xE6D6, 0x83EC, 0x1B57, 0x2CDF, 0x2CBF, 0x0AF4, 0x12F5, 0x2CBF, 0x2CDF, 0x1B57, 0x8C0D, 0xDE95, 0xD674, 0xDE95, 0xDEB5, 0xDEB5, // 0x0140 (320) pixels
  0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xE6F6, 0x8C2D, 0x1315, 0x1C9F, 0x1378, 0x6B8D, 0x6B8D, 0x1378, 0x1C9F, 0x1315, 0x8C2D, 0xE6F6, // 0x0150 (336) pixels
  0xDEB5, 0xDEB5, 0xDE95, 0xDE95, 0xDEB5, 0xDEB5, 0xD654, 0xD674, 0xCE33, 0x736B, 0x530D, 0x4B50, 0x2A0B, 0x944D, 0x944E, 0x3ACE, // 0x0160 (352) pixels
  0x5370, 0x42AC, 0x62E9, 0xC5F3, 0xD674, 0xDEB5, 0xD674, 0xD654, 0xDEB5, 0xDEB5, 0xDE95, 0xCE13, 0x734A, 0x8B8A, 0xA388, 0x9B68, // 0x0170 (368) pixels
  0x5206, 0x840C, 0x840D, 0x7B29, 0xA388, 0x9B68, 0x7AC6, 0x7309, 0xCE13, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xDEB5, 0xE6D6, 0xA4CF, // 0x0180 (384) pixels
  0x28E2, 0x4183, 0x4163, 0x4163, 0x20C2, 0x8C4D, 0x8C4D, 0x28E2, 0x4163, 0x4163, 0x4183, 0x28E2, 0xA4CF, 0xE6D6, 0xDEB5, 0xDEB5, // 0x0190 (400) pixels
},
```

LINKS

1. Sokban. https://en.wikipedia.org/wiki/Sokoban
2. STM32103C8. https://www.st.com/en/microcontrollers-microprocessors/stm32f103c8
3. Github. https://github.com/lucas458/sokobanSTM32
4. Conversor para RGB 565. http://www.rinkydinkelectronics.com/t_imageconverter565.php

FIM