```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <avr/pgmspace.h>

//#define SPEAKER_PIN 8 //buzzer
#define VERT_PIN0 A3 //player 0
#define VERT_PIN1 A0 //player 1
#define HORZ_PIN1 A2
#define HORZ_PIN0 A1 //player 0
#define SEL_PIN1  2 //botão player 1
#define SEL_PIN0  4 //botão player 0
// SDA A4, SCL A5
// cima=0,direita=0

#define SCREEN_WIDTH       128 // OLED display width, in pixels
#define SCREEN_HEIGHT       64 // OLED display height, in pixels

uint8_t fruit[64][3];


// main menu
uint8_t cena=0;

////////////////////////////////////////
uint8_t ball_x = 64, ball_y = 32;
uint8_t ball_dir_x = 1, ball_dir_y = 1;
uint8_t count;
```

```
const unsigned long PADDLE_RATE = 40;
const unsigned long BALL_RATE = 20;
uint8_t PADDLE_HEIGHT = 20;

unsigned long ball_update = 0;
unsigned long paddle_update = 0;
const uint8_t raio = 2; // class Ball
bool jogoIniciado = false;
//////////////////////////////////////

unsigned long time;

// navegação do menu
unsigned long lastMoveTime = 0;
uint8_t moveDelay = 200;

// estados do menu
enum MenuState { MAIN_MENU, REGRAS_MENU, RANDOM_MENU,
ANALOGICO_MENU};
MenuState currentMenu = MAIN_MENU;

// menu principal
uint8_t mainIndex = 0;
uint8_t mainLength = 4;

// submenu de regras
uint8_t regrasIndex = 0;
```

```
uint8_t regrasLength = 3;
bool pontoc = false; // fim de jogo 5
bool pontod = false; //fim de jogo 10

// submenu do random
uint8_t regrasRandomIndex = 0;
uint8_t regrasRandomLength = 3;

// Declaration for an SSD1306 display connected to I2C (SDA, SCL pins)
#define OLED_RESET           -1// Reset pin # (or -1 if sharing Arduino reset pin)
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

void setup() {
  Serial.begin(9600);

  pinMode(VERT_PIN0, INPUT);
  pinMode(VERT_PIN1, INPUT);
  pinMode(HORZ_PIN0, INPUT);
  pinMode(HORZ_PIN1, INPUT);
  pinMode(SEL_PIN0, INPUT_PULLUP);
  pinMode(SEL_PIN1, INPUT_PULLUP);


  if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
    Serial.println(F("SSD1306 allocation failed"));
    for(;;);
  }
  // Show initial display buffer contents on the screen --
```

```
  // the library initializes this with an Adafruit splash screen.
  display.display();
  delay(500);
  // Clear the buffer
  //display.clearDisplay();

  randomSeed(analogRead(7)); // semi aleatorio
  setupGame();

  //display.display();

  for(uint8_t i=0; i<64; i++){
    fruit[i][0] = 0; // fruta X
    fruit[i][1] = 0; // fruta Y
    fruit[i][2] = 0; // true/false
  }

}

//////////////////////Teste de RAM//////////////////////////////
void display_freeram() {
  Serial.print(F("- SRAM left: "));
  Serial.println(freeRam());
}

int freeRam() {
  extern int __heap_start,*__brkval;
  int v;
```

```
  return (int)&v - (__brkval == 0
    ? (int)&__heap_start : (int) __brkval);
}
////////////////////////////////////////////////////////////

void setupGame() {
  display_freeram();
  display.clearDisplay();
  drawMenu();
}



void drawInicio() // cena 0 menu
{
  bool selPressed0 = digitalRead(SEL_PIN0) == LOW;
  bool selPressed1 = digitalRead(SEL_PIN1) == LOW;
  int vert0 = analogRead(VERT_PIN0);
  int vert1 = analogRead(VERT_PIN1);
  int horz0 = analogRead(HORZ_PIN0);
  int horz1 = analogRead(HORZ_PIN1);



  if (currentMenu == ANALOGICO_MENU) {
    drawAnalogicoMenu();
    return;
  }
  // Navegação do index
  if (millis() - lastMoveTime > moveDelay) {
```

```
  if (vert0 < 20 || horz0 < 20 || horz1 < 20) {  // CIMA
    if (currentMenu == MAIN_MENU) {
      mainIndex--;
      if (mainIndex < 0) mainIndex = 0;
      drawMenu();
    } else if (currentMenu == REGRAS_MENU) {
      regrasIndex--;
      if (regrasIndex < 0) regrasIndex = 0;
      drawRegrasMenu();
    } else if (currentMenu == RANDOM_MENU) {
      regrasRandomIndex--;
      if (regrasRandomIndex < 0) regrasRandomIndex = 0;
      drawRandomMenu();
    }
    lastMoveTime = millis();
  } else if (vert0 > 1000 || horz0 > 1000 || horz1 > 1000) {  // BAIXO
    if (currentMenu == MAIN_MENU) {
      mainIndex++;
      if (mainIndex >= mainLength) mainIndex = mainLength - 1;
      drawMenu();
    } else if (currentMenu == REGRAS_MENU) {
      regrasIndex++;
      if (regrasIndex >= regrasLength) regrasIndex = regrasLength - 1;
      drawRegrasMenu();
    } else if (currentMenu == RANDOM_MENU) {
      regrasRandomIndex++;
      if (regrasRandomIndex >= regrasRandomLength) regrasRandomIndex =
regrasRandomLength - 1;
```

```
      drawRandomMenu();
    }
    lastMoveTime = millis();
  }
}

// Selecionar função
if (selPressed0 || selPressed1) {
  delay(200);

  if (currentMenu == MAIN_MENU) {
    switch (mainIndex) {
      case 0:  // Jogar
        cena = 1;
        display.clearDisplay();
        drawMap();
        Troll();
      return;
      case 1:  // Regras
        currentMenu = REGRAS_MENU;
        drawRegrasMenu();
      return;
      case 2:  // Random
        currentMenu = RANDOM_MENU;
        drawRandomMenu();
      break;
      case 3:  // Analogico
        currentMenu = ANALOGICO_MENU;
```

```
      drawAnalogicoMenu();
      break;
    }
  } else if (currentMenu == REGRAS_MENU) {
    switch (regrasIndex) {
      case 0:
        pontoc = true;
        showMessage(F("Pontuacao: 5"));
      break;
      case 1:
        pontod = true;
        showMessage(F("Pontuacao: 10"));
      break;
      case 2:
      showMessage(F("infinito"));
      break;
    }
    currentMenu = MAIN_MENU;
    drawMenu();
  } else if (currentMenu == RANDOM_MENU) {
    switch (regrasRandomIndex) {
      case 0:

        showMessage(F("Pixel aparecera a cada batida de raquete"));
      break;
      case 1:

        showMessage(F("Pixel aparecera a cada 2 batida de raquete"));
```

```
      break;
      case 2:

      showMessage(F("aguardando..."));
      break;
    }
    currentMenu = MAIN_MENU;
    drawMenu();
   }
  }
}

void drawMenu() {
  display.clearDisplay();
  display.setTextSize(1);
  display.setTextColor(SSD1306_WHITE);
  display.setCursor(0, 0);
  display.print((mainIndex == 0 ? "> " : "  ")); display.println(F("1 - Jogar"));
  display.setCursor(0, 10);
  display.print((mainIndex == 1 ? "> " : "  ")); display.println(F("2 - Regras"));
  display.setCursor(0, 20);
  display.print((mainIndex == 2 ? "> " : "  ")); display.println(F("3 - Random"));
  display.setCursor(0, 30);
  display.print((mainIndex == 3 ? "> " : "  ")); display.println(F("4 - Analogico"));

  display.display();
}
```

```
void drawRegrasMenu() {
  display.clearDisplay();
  display.setTextSize(1);
  display.setTextColor(SSD1306_WHITE);
  display.setCursor(0, 0);
  display.print((regrasIndex == 0 ? "> " : "  ")); display.println(F("Pontuacao: 5"));
  display.setCursor(0, 10);
  display.print((regrasIndex == 1 ? "> " : "  ")); display.println(F("Pontuacao: 10"));
  display.setCursor(0, 20);
  display.print((regrasIndex == 2 ? "> " : "  ")); display.println(F("Infinito"));

  display.display();
}

void drawRandomMenu() {
  display.clearDisplay();
  display.setTextSize(1);
  display.setTextColor(SSD1306_WHITE);
  display.setCursor(0, 0);
  display.print((regrasRandomIndex == 0 ? "> " : "  ")); display.println(F("A cada
batida"));
  display.setCursor(0, 10);
  display.print((regrasRandomIndex == 1 ? "> " : "  ")); display.println(F("A cada 2
batidas"));
  display.setCursor(0, 20);
  display.print((regrasRandomIndex == 2 ? "> " : "  ")); display.println(F(" "));

  display.display();
}
```

```
void drawAnalogicoMenu() {
  int vert0 = analogRead(VERT_PIN0);
  int horz0 = analogRead(HORZ_PIN0);
  int vert1 = analogRead(VERT_PIN1);
  int horz1 = analogRead(HORZ_PIN1);

  display.clearDisplay();
  display.setTextSize(1);
  display.setTextColor(WHITE);

  display.setCursor(0, 0);
  display.println(F("Player 1 (Direita)"));
  display.print(F("V: ")); display.println(vert0);
  display.print(F("H: ")); display.println(horz0);

  display.setCursor(0, 30);
  display.println(F("Player 2 (Esquerda)"));
  display.print(F("V: ")); display.println(vert1);
  display.print(F("H: ")); display.println(horz1);

  display.display();

  // Voltar ao menu principal com qualquer botão
  if (digitalRead(SEL_PIN0) == LOW || digitalRead(SEL_PIN1) == LOW) {
    delay(200);
    currentMenu = MAIN_MENU;
    drawMenu();
```

```
  }
}

void showMessage(const __FlashStringHelper *msg) {
  display.clearDisplay();
  display.setTextSize(1);
  display.setTextColor(SSD1306_WHITE);
  display.setCursor(0, 20);
  display.println(msg);
  display.display();
  delay(1000);
}

//////////////////////////////////////////////////////////////////////

void loop() {
  switch(cena)
  {
    case 0:  // Menu
      drawInicio();
      break;
    case 1: // Jogo
      drawJogo();
      atualizarTetris();
      break;
    case 2: //Minigame
      jogoIniciado = false; // só para nao usar toda vez a variavel
```

```
      break;
    case 3:
      rapido();
      break;
    case 4:
      memoria();
      break;
    case 5:
      matematica();
  }
}
void Troll(){
  uint8_t i;

  for(i=0; i < 64; i++){
    if(!fruit[i][2]){
      fruit[i][0] = random(6, 122);
      fruit[i][1] = random(2, 63);
      fruit[i][2] = 1;
      break;
    }
  }

}

bool modoWalless = false;
void walless (){
  modoWalless = true;
```

```
  Serial.println(F("walless"));
}

//////////////////////////////////////////////////////////////////////

//////////////////////////Parede//////////////////////////

class Parede {
public:
  uint8_t x, y;       // posição de origem
  uint8_t tamanho;
  bool horizontal;    // true = horizontal false = vertical
  bool ativa;

  Parede() {
    ativa = false;
  }

  void spawn() {
    ativa = true;
    horizontal = random(0, 2); // 0 = vertical, 1 = horizontal

    if (horizontal) {
      tamanho = random(20, 50);
      x = random(5, 128 - tamanho - 5);
      y = random(10, 54);
    } else {
      tamanho = random(15, 40);
```

```
    x = random(10, 120);
    y = random(5, 64 - tamanho - 5);
  }
}

void draw() {
  if (!ativa) return;
  if (horizontal)
    display.drawFastHLine(x, y, tamanho, WHITE);
  else
    display.drawFastVLine(x, y, tamanho, WHITE);
  display.display();
}

void apagar() {
  if (!ativa) return;
  if (horizontal)
    display.drawFastHLine(x, y, tamanho, BLACK);
  else
    display.drawFastVLine(x, y, tamanho, BLACK);
  display.display();
  ativa = false;
}

bool colidiu(uint8_t bolaX, uint8_t bolaY, int &dirX, int &dirY) {
  if (!ativa) return false;
```

```
// colisão com parede horizontal
if (horizontal &&
    (bolaY + raio >= y && bolaY - raio <= y + 1) && // faixa vertical com raio
    (bolaX + raio >= x && bolaX - raio <= x + tamanho)) {

  dirY = -dirY;

  int centro = x + (tamanho / 2);
  int dif = bolaX - centro;
  if (abs(dif) > tamanho / 3) {
    dirX = (dif < 0) ? -1 : 1;
  }

  apagar();
  return true;
}

// Colisão com parede vertical
if (!horizontal &&
    (bolaX + raio >= x && bolaX - raio <= x + 1) && // faixa horizontal com raio
    (bolaY + raio >= y && bolaY - raio <= y + tamanho)) {

  dirX = -dirX;

  int centro = y + (tamanho / 2);
  int dif = bolaY - centro;
  if (abs(dif) > tamanho / 3) {
```

```
      dirY = (dif < 0) ? -1 : 1;
    }

    apagar();
    return true;
  }

  return false;
 }

};
#define MAX_PAREDE  5
Parede parede[MAX_PAREDE];

////////////////////////////Parede//////////////////////////////

////////////////////////////Tetris//////////////////////////////

const uint8_t peca[6][8][8] PROGMEM = {
 { {0,0,0,0,0,0,0,0}, {0,1,1,1,1,1,1,0}, {0,1,1,1,1,1,1,0}, {0,1,1,1,1,1,1,0}, {0,1,1,1,1,1,1,0},
{0,1,1,1,1,1,1,0}, {0,1,1,1,1,1,1,0}, {0,0,0,0,0,0,0,0} },
 { {0,0,0,0,0,0,0,0}, {0,1,0,0,0,0,0,0}, {0,1,0,0,0,0,0,0}, {0,1,1,1,1,0,0,0}, {0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0}, {0,0,0,0,0,0,0,0}, {0,0,0,0,0,0,0,0} },
 { {0,0,0,0,0,0,0,0}, {0,1,1,0,0,0,0,0}, {0,0,1,1,0,0,0,0}, {0,0,0,0,0,0,0,0}, {0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0}, {0,0,0,0,0,0,0,0}, {0,0,0,0,0,0,0,0} },
 { {0,0,1,1,1,1,0,0}, {0,0,0,0,0,0,0,0}, {0,0,0,0,0,0,0,0}, {0,0,0,0,0,0,0,0}, {0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0}, {0,0,0,0,0,0,0,0}, {0,0,0,0,0,0,0,0} },
 { {0,0,1,1,1,0,0,0}, {0,0,0,1,0,0,0,0}, {0,0,0,1,0,0,0,0}, {0,0,0,0,0,0,0,0}, {0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0}, {0,0,0,0,0,0,0,0}, {0,0,0,0,0,0,0,0} },
```

```
  { {0,0,0,1,0,0,0,0}, {0,0,0,1,0,0,0,0}, {0,1,1,1,1,1,0,0}, {0,0,0,1,0,0,0,0}, {0,0,0,1,0,0,0,0},
{0,0,0,0,0,0,0,0}, {0,0,0,0,0,0,0,0}, {0,0,0,0,0,0,0,0} }
};

int dist(int x1, int y1, int x2, int y2) {
  return (x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2);
}
class Bloco{
  public:
    uint8_t x,y;
    bool ativo;
    uint8_t tipo;


    Bloco(uint8_t xPos, uint8_t yPos, uint8_t t)
      : x(xPos), y(yPos), tipo(t), ativo(true) {}

    bool colidiu(uint8_t xBola, uint8_t yBola) {
      uint8_t t = tipo % 6;
      for (int8_t i = 0; i < 8; i++) {
        for (int8_t j = 0; j < 8; j++) {
          uint8_t v = pgm_read_byte(&peca[t][i][j]);
          if (v == 1) {
            // coordenadas reais do pixel desenhado da peça (cada célula 2×2)
            uint8_t px = x + j * 2 + 1;
            uint8_t py = y + i * 2 + 1;

            // colisão: a borda da bola toca a borda da célula da peça
```

```
        if (dist(xBola, yBola, px, py) <= raio * raio) {
          return true;
        }
       }
      }
     }
     return false;
    }

    void apagarPeca() {
     uint8_t t = tipo % 6;
     for (int i = 0; i < 8; i++)
      for (int j = 0; j < 8; j++) {
       uint8_t v = pgm_read_byte(&peca[t][i][j]);
       if (v == 1)
        display.fillRect(x + j * 2, y + i * 2, 2, 2, SSD1306_BLACK);
      }
    }

    void drawPeca() {
     uint8_t t = tipo % 6;
     for (int i = 0; i < 8; i++)
      for (int j = 0; j < 8; j++) {
       uint8_t v = pgm_read_byte(&peca[t][i][j]);
       if (v == 1)
       display.fillRect(x + j * 2, y + i * 2, 2, 2, v == 0 ? SSD1306_BLACK :
SSD1306_WHITE);
      }
```

```
    }

    void descer() {
      apagarPeca();
      y += 1;
      drawPeca();
    }
}; Bloco bloco(60, 10, 0);

unsigned long ultimoTetrisUpdate = 0;
const unsigned long intervaloTetris = 300;
bool blocoAtivo = false;

void iniciarTetris() {
  bloco.x = 60;
  bloco.y = 0;
  bloco.tipo = random(0, 6);
  bloco.ativo = true;
  blocoAtivo = true;
  ultimoTetrisUpdate = millis();
  bloco.drawPeca();
}

void atualizarTetris() {
  if (blocoAtivo && bloco.ativo && millis() - ultimoTetrisUpdate >= intervaloTetris) {
    bloco.descer();
    display.display();
    ultimoTetrisUpdate = millis();
```

```
    // verifica se toda a peça saiu da tela
    bool saiuDaTela = true;
    uint8_t t = bloco.tipo % 6;

    for (uint8_t i = 0; i < 8; i++) {
      for (uint8_t j = 0; j < 8; j++) {
        uint8_t v = pgm_read_byte(&peca[t][i][j]);
        if (v == 1) {
          uint8_t py = bloco.y + i * 2;
          if (py < SCREEN_HEIGHT) {
            saiuDaTela = false; // parte da peça ainda visível
            break;
          }
        }
      }
      if (!saiuDaTela) break;
    }

    if (saiuDaTela) {
      bloco.apagarPeca();
      bloco.ativo = false;
      blocoAtivo = false;
    }
  }
}

//////////////////////Tetris////////////////////////////////
```

```
class Paddle {
  public:
    uint8_t x;
    uint8_t y;
    uint8_t height;
    int pinAnalog;
    bool invertido;

    Paddle(uint8_t xPos, uint8_t yPos, uint8_t h, int analogPin, bool inv = false)
      : x(xPos), y(yPos), height(h), pinAnalog(analogPin), invertido(inv) {}

    void move() {
      int val = analogRead(pinAnalog);
      if (!invertido) {
        if (val == 1023) y++;
        if (val == 0) y--;
      } else {
        if (val == 1023) y--;  //orientação do player 2
        if (val == 0) y++;
      }

      if (y < 1) y = 1;
      if (y + height > 63) y = 63 - height;
    }

    void draw(bool erase = false) {
      display.drawFastVLine(x, y, height, erase ? SSD1306_BLACK : SSD1306_WHITE);
```

```
  }
};
Paddle cpu(4, 16, PADDLE_HEIGHT, HORZ_PIN1, true);
Paddle player(124, 16, PADDLE_HEIGHT, HORZ_PIN0, false);

class Ball {
 public:
  uint8_t x, y;
  int dirX, dirY;
  uint8_t raio = 2;

  Ball(uint8_t startX, uint8_t startY)
    : x(startX), y(startY), dirX(1), dirY(1) {}

  void reset() {
    x = 64;
    y = 32;
    raio = 2;
    dirX = random(0, 2) == 0 ? -1 : 1;
    dirY = random(0, 2) == 0 ? -1 : 1;
  }

  void draw(bool erase = false) {
    display.fillCircle(x, y, raio, erase ? SSD1306_BLACK : SSD1306_WHITE);
  }

  void telafim(uint8_t vencedor) {
    display.clearDisplay();
```

```
    display.setTextSize(2);
    display.setTextColor(WHITE);
    display.setCursor(10, 20);

    if (vencedor == 0)
      display.println(F("Player 0 Venceu!"));
    else
      display.println(F("Player 1 Venceu!"));

    display.display();
    delay(2000);

    cena = 0;
    drawMenu();
  }

  void drawPlacar(uint8_t scoreL, uint8_t scoreR) {
    display.clearDisplay();
    display.setTextSize(1);
    display.setTextColor(WHITE);
    display.setCursor(2,2);
    display.println(scoreL);
    display.setCursor(114,4);
    display.println(scoreR);
    display.display();
}
```

```
  void update(Paddle &left, Paddle &right, uint8_t &scoreLeft, uint8_t &scoreRight,
Bloco &bloco) {
    draw(true);

    int newX = x + dirX;
    int newY = y + dirY;

    // Efeito de parede inferior/superior (modo walless)
    if (modoWalless) {
      if (newY < 0) {
        newY = 61;
      } else if (newY > 63) {
        newY = 2;
      }
    } else {
      // Colisão com parede superior
      if (newY <= 0) {
        dirY = abs(dirY);  // força a direção para baixo
        newY = 0;
      }
      // Colisão com parede inferior
      if (newY >= 63) {
        dirY = -abs(dirY); // força a direção para cima
        newY = 63;
      }
    }
```

```
// Pontuação



if (newX <= 0) {
  scoreRight++;
  drawPlacar(scoreLeft, scoreRight);
  if ((scoreRight >= 10 && pontod) || (scoreRight >= 5 && pontoc)) { // fim de jogo
    telafim(1);
    scoreLeft = 0;
    scoreRight = 0;
    return;
  }
  drawMap();
  reset();
  modoWalless = false;
  for (uint8_t p = 0; p < MAX_PAREDE; p++) { // paredes continuam apos pontuação
    if (parede[p].ativa) {
      parede[p].draw();
      break;
    }
  }
  player.height = 20;
  cpu.height = 20;
  return;
}
if (newX >= 127) {
  scoreLeft++;
  drawPlacar(scoreLeft, scoreRight);
```

```
    if ((scoreLeft >= 10 && pontod) || (scoreLeft >= 5 && pontoc)) { // fim de jogo
      telafim(0);
      scoreLeft = 0;
      scoreRight = 0;
      return;
    }
    drawMap();
    reset();
    modoWalless = false;
    for (uint8_t p = 0; p < MAX_PAREDE; p++) { // paredes continuam apos pontuação
      if (parede[p].ativa) {
        parede[p].draw();
        break;
      }
    }
    player.height = 20;
    cpu.height = 20;
    return;
  }

  // Colisão com parede
  for (uint8_t p = 0; p < MAX_PAREDE; p++) {
    if (parede[p].ativa && parede[p].colidiu(newX, newY, dirX, dirY)) // apaga a parede
na propria função
      break;
  }
  // Colisão com bloco tetris
  if (bloco.ativo && blocoAtivo && bloco.y > 0 && bloco.y < SCREEN_HEIGHT &&
bloco.colidiu(newX, newY)) {
```

```
      // Calcula posição relativa da colisão
      int relY = (uint8_t)newY - (uint8_t)(bloco.y + 8); // centro vertical do bloco (aproximado)
      int relX = (uint8_t)newX - (uint8_t)(bloco.x + 8); // centro horizontal

      // Define a saída preferencial
      if (abs(relY) > abs(relX)) {
        dirY = (relY < 0) ? -abs(dirY) : abs(dirY); // força direção para fora
      } else {
        dirX = (relX < 0) ? -abs(dirX) : abs(dirX);
      }
      // Empurra a bola levemente para fora para evitar repetição de colisão no próximo quadro
      newX += dirX;
      newY += dirY;
    }

    // Colisão com raquete esquerda
    if (newX == left.x + 1 && newY >= left.y && newY <= left.y + left.height) {
      dirX = -dirX;
      count++;
      int centro = left.y + (left.height / 2);
      int diferenca = newY - centro;

      // Converte em direção vertical proporcional e evita dirY = 0
      dirY = (diferenca < 0) ? -1 : 1;
      if (abs(diferenca) > (left.height / 3)) dirY *= 2; // curva maior se mais longe do centro
    }
```

```
    // Colisão com raquete direita
    if (newX == right.x - 1 && newY >= right.y && newY <= right.y + right.height) {
      dirX = -dirX;
      count++;
      int centro = right.y + (right.height / 2);
      int diferenca = newY - centro;

      dirY = (diferenca < 0) ? -1 : 1;
      if (abs(diferenca) > (right.height / 3)) dirY *= 2;
    }

    x = newX;
    y = newY;
    draw();
  }

};
Ball ball(64, 32);

void drawMap() {
  display.drawRect(0, 0, 128, 64, WHITE);
  display.display();
}

////////////////////////////////////////////////////////////////

void checkTroll() {
```

```
uint8_t troll = random(0,5);
uint8_t minigame = random(0,3);



 for(uint8_t i=0; i<64; i++){
    if(fruit[i][2]){
      if(fruit[i][0] >= ball.x-1 && fruit[i][0] <= ball.x+1 && fruit[i][1] >= ball.y-1 &&
fruit[i][1] <= ball.y+1){
        //Serial.println(F("acertou"));


        switch (troll){
          case 0:
            cena = 2; // pausa o Pong
            switch (minigame){
              case 0:
                mostrarPreparar();
                cena = 3 ;
                //rapido();
                break;
              case 1:
                mostrarPreparar();
                cena =4;
                //memoria();
                break;
              case 2:
                mostrarPreparar();
                cena =5;
                //matematica();
```

```
      break;


    }
    break;
  case 1:
    iniciarTetris();
    display.clearDisplay();
    drawMap();
    break;
  case 2:
    pequeno();
    display.clearDisplay();
    drawMap();
    break;
  case 3:
    walless();
    break;
  case 4:
    for (uint8_t p = 0; p < MAX_PAREDE; p++) {
      if (!parede[p].ativa) {
        parede[p].spawn();
        parede[p].draw();
        break;
      }
    }


    break;
}
```

```
        // Retira o pixel encostado
        fruit[i][2] = 0;
       }
     }
   }
}

////////////////////////////////////////////////////////////////

uint8_t i=0;
uint8_t player0 = 0;
uint8_t player1 = 0;
bool poderPlayer0 = false;
bool poderPlayer1 = false;

void rebote(Ball &ball, uint8_t jogador);

void drawJogo() {
  time = millis();
  bool bupdate = false;
  bool botaoPlayer0 = digitalRead(SEL_PIN0) == LOW;
  bool botaoPlayer1 = digitalRead(SEL_PIN1) == LOW;

  if (!jogoIniciado) {
    ball_update = time + BALL_RATE;
    paddle_update = time + PADDLE_RATE;
    jogoIniciado = true;
  }
```

```
if (time > ball_update) {
  ball.update(cpu, player, player0, player1, bloco);
  ball_update += BALL_RATE;
  bupdate = true;
}

if (bupdate) {
  checkTroll();  // pode ativar efeitos
}

if (bupdate) {
  for (uint8_t i = 0; i < 64; i++) {
    if (fruit[i][2]) {
      display.writePixel(fruit[i][0], fruit[i][1], WHITE);
    }
  }
}

if (time > paddle_update) {
  cpu.draw(true);
  player.draw(true);

  cpu.move();
  player.move();

  cpu.draw();
  player.draw();
```

```
    paddle_update += PADDLE_RATE;
    bupdate = true;
  }

  // rebote
  if (botaoPlayer0) {
    rebote(ball, 0);
  }
  if (botaoPlayer1) {
    rebote(ball, 1);
  }

  if (bupdate) {
    display.display();
  }

  if (i < count) {
    Troll();
    i++;
  }
}

//////////////////////////MINIGAMES////////////////////////////////////

uint8_t vencedor;
uint8_t buff;
```

```
void mostrarPreparar() {
  display.clearDisplay();
  display.setTextSize(1.5);
  display.setTextColor(WHITE);
  display.setCursor(15, 25);
  display.println(F("PREPARAR..."));
  display.display();
  delay(1000);
}

//////////////////////Minigame rapido//////////////////////////////
void setaCima(uint8_t x, uint8_t y) {
  display.fillTriangle(x, y, x - 10, y + 20, x + 10, y + 20, WHITE);
}
void setaBaixo(int x, int y) {
  display.fillTriangle(x, y + 20, x - 10, y, x + 10, y, WHITE);
}
void setaEsquerda(int x, int y) {
  display.fillTriangle(x, y, x + 20, y - 10, x + 20, y + 10, WHITE);
}
void setaDireita(int x, int y) {
  display.fillTriangle(x, y, x - 20, y - 10, x - 20, y + 10, WHITE);
}
void setaCimaEsq(int x, int y) {
  display.fillTriangle(x, y, x + 14, y + 5, x + 5, y + 14, WHITE);
}
void setaCimaDir(int x, int y) {
  display.fillTriangle(x, y, x - 14, y + 5, x - 5, y + 14, WHITE);
```

```
}
void setaBaixoEsq(int x, int y) {
  display.fillTriangle(x, y, x + 14, y - 5, x + 5, y - 14, WHITE);
}
void setaBaixoDir(int x, int y) {
  display.fillTriangle(x, y, x - 14, y - 5, x - 5, y - 14, WHITE);
}

int lerDirecaoJoystick(int horz, int vert) {
  const int deadzoneLow = 400;
  const int deadzoneHigh = 600;

  // diagonais primeiro!!!
  if (vert < 100 && horz > 900) return 4; // cima + esquerda
  if (vert < 100 && horz < 100) return 5; // cima + direita
  if (vert > 900 && horz > 900) return 6; // baixo + esquerda
  if (vert > 900 && horz < 100) return 7; // baixo + direita
  if (vert < 100) return 0; // cima
  if (vert > 900) return 1; // baixo
  if (horz > 900) return 2; // esquerda
  if (horz < 100) return 3; // direita

  // Zona morta (parado)
  if (vert > deadzoneLow && vert < deadzoneHigh &&
      horz > deadzoneLow && horz < deadzoneHigh) {
    return -1;
  }
```

```
  return -1;
}

//////////////////////////////////////////////////////////////////

void mostrarVencedor(uint8_t jogador) {

  buff = random(0,2);
  display.clearDisplay();
  display.setTextSize(1);
  display.setTextColor(SSD1306_WHITE);

  if (jogador == 0){
    display.setRotation(3); // só gira 1 texto por vez
    display.setCursor(0, 90);
    display.print(F("Ganhou:   "));



  }
  else if(jogador == 1){
    display.setRotation(1);
    display.setCursor(0, 90);
    display.println(F("Ganhou:   "));

  }

  switch(buff){
     case 0:
```

```
      display.println(F("Grande"));
      grande();
    break;
    case 1:
      display.println(F("Rebote"));
      if (vencedor == 0)
      poderPlayer0 = true;
      else if (vencedor == 1)
      poderPlayer1 = true;

    break;
   }
  display.display();
  display.setRotation(0);
  delay(1000);
}

////////////////////////////Rapido////////////////////////////

void rapido() {
  static bool iniciado = false;
  static unsigned long inicioTempo = 0;
  static bool esperandoMovimento = false;
  static bool resultadoMostrado = false;
  static uint8_t direcao = 0;

  if (!iniciado) {
    iniciado = true;
```

```
  direcao = random(0, 8);
  inicioTempo = millis();
  esperandoMovimento = false;
  resultadoMostrado = false;

  display.clearDisplay();
  display.setTextSize(2);
  display.setTextColor(WHITE);

  int xCentro = 64;
  int ySeta   = 22;

  switch (direcao) {
    case 0: setaCima(xCentro, ySeta); break;
    case 1: setaBaixo(xCentro, ySeta); break;
    case 2: setaEsquerda(xCentro, ySeta); break;
    case 3: setaDireita(xCentro, ySeta); break;
    case 4: setaCimaEsq(xCentro, ySeta); break;
    case 5: setaCimaDir(xCentro, ySeta); break;
    case 6: setaBaixoEsq(xCentro, ySeta); break;
    case 7: setaBaixoDir(xCentro, ySeta); break;
  }

  display.display();
}

if (!esperandoMovimento && millis() - inicioTempo > 800) {
  esperandoMovimento = true;
```

```
  }

  if (esperandoMovimento && !resultadoMostrado) {
    int dir0 = lerDirecaoJoystick(analogRead(HORZ_PIN0), analogRead(VERT_PIN0));
    int dir1 = lerDirecaoJoystick(analogRead(HORZ_PIN1), analogRead(VERT_PIN1));

    if (dir0 == direcao) {
      mostrarVencedor(0);
      vencedor = 0;
      resultadoMostrado = true;
      inicioTempo = millis();
    }
    else if (dir1 == direcao) {
      mostrarVencedor(1);
      vencedor = 1;
      resultadoMostrado = true;
      inicioTempo = millis();
    }
  }

  if (resultadoMostrado && millis() - inicioTempo > 3000) {
    iniciado = false;
    cena = 1; // volta ao jogo
    display.clearDisplay();
    drawMap();
  }
}
```

```
//////////////////////////Minigame rapido//////////////////////

//////////////////////////Minigame memoria//////////////////////
void desenharSeta(uint8_t dir, int x, int y) {
  switch (dir) {
    case 0: setaCima(x, y); break;
    case 1: setaBaixo(x, y); break;
    case 2: setaEsquerda(x, y); break;
    case 3: setaDireita(x, y); break;
    case 4: setaCimaEsq(x, y); break;
    case 5: setaCimaDir(x, y); break;
    case 6: setaBaixoEsq(x, y); break;
    case 7: setaBaixoDir(x, y); break;
  }
}

void memoria() {
  static bool iniciado = false;
  static uint8_t sequencia[5];
  static const uint8_t tamanho = 4;
  static uint8_t indice0 = 0;
  static uint8_t indice1 = 0;
  static uint8_t indiceMostrar = 0;
  static unsigned long ultimoTempo = 0;
  static bool mostrando = true;
  static bool esperandoInput = false;
  static bool resultadoMostrado = false;
```

```
// controle de repetição
static int lastDir0 = -1;
static int lastDir1 = -1;

if (!iniciado) {
  iniciado = true;
  indice0 = 0;
  indice1 = 0;
  indiceMostrar = 0;
  mostrando = true;
  esperandoInput = false;
  resultadoMostrado = false;
  lastDir0 = -1;
  lastDir1 = -1;

  for (uint8_t i = 0; i < tamanho; i++) {
    sequencia[i] = random(0, 4); // sem diagonais
  }

  display.clearDisplay();
  display.setTextSize(1);
  display.setCursor(20, 20);
  display.println(F("Memorize:"));
  display.display();
  delay(500);
  ultimoTempo = millis();
}
```

```
// mostrar a sequência
if (mostrando && millis() - ultimoTempo > 1000 && indiceMostrar < tamanho) {
  display.clearDisplay();
  desenharSeta(sequencia[indiceMostrar], 64, 22);
  display.display();
  delay(500);
  display.clearDisplay();
  display.display();
  delay(500);

  indiceMostrar++;
  ultimoTempo = millis();
}

if (mostrando && indiceMostrar >= tamanho) {
  mostrando = false;
  esperandoInput = true;
  indice0 = 0;
  indice1 = 0;
  display.clearDisplay();
  display.setCursor(10, 20);
  display.println(F("Sua vez..."));
  display.display();
}

// entrada dos jogadores
if (esperandoInput && !resultadoMostrado) {
  int dir0 = lerDirecaoJoystick(analogRead(HORZ_PIN0), analogRead(VERT_PIN0));
```

```
int dir1 = lerDirecaoJoystick(analogRead(HORZ_PIN1), analogRead(VERT_PIN1));


// player 0
if (dir0 != -1 && dir0 != lastDir0) {
  if (dir0 == sequencia[indice0]) {
    indice0++;
    if (indice0 == tamanho) {
      mostrarVencedor(0);
      vencedor = 0;
      resultadoMostrado = true;
      esperandoInput = false;
      ultimoTempo = millis();
    }
  } else {
    mostrarVencedor(1); // errou  outro ganha
    vencedor = 1;
    resultadoMostrado = true;
    esperandoInput = false;
    ultimoTempo = millis();
  }
}
lastDir0 = dir0;


// player 1
if (dir1 != -1 && dir1 != lastDir1) {
  if (dir1 == sequencia[indice1]) {
    indice1++;
    if (indice1 == tamanho) {
```

```
          mostrarVencedor(1);
          vencedor = 1;
          resultadoMostrado = true;
          esperandoInput = false;
          ultimoTempo = millis();
        }
      } else {
        mostrarVencedor(0); // errou  outro ganha
        vencedor = 0;
        resultadoMostrado = true;
        esperandoInput = false;
        ultimoTempo = millis();
      }
    }
    lastDir1 = dir1;
  }

  if (resultadoMostrado && millis() - ultimoTempo > 3000) {
    iniciado = false;
    cena = 1;
    display.clearDisplay();
    drawMap();
  }
}

////////////////////////////Minigame memoria////////////////////////

////////////////////////////Minigame matematica////////////////////////
```

```
int num1, num2, resultado;
char operador;
bool verdadeirom;

void gerarExpressaoMatematica() {
  num1 = random(1, 10);
  num2 = random(1, 10);
  int tipo = random(0, 3); // 0: +, 1: -, 2: *

  switch (tipo) {
    case 0: operador = '+'; resultado = num1 + num2; break;
    case 1: operador = '-'; resultado = num1 - num2; break;
    case 2: operador = '*'; resultado = num1 * num2; break;
  }

  // Decide se o resultado mostrado será correto ou falso
  verdadeirom = random(0, 2);
  if (!verdadeirom) {
    resultado += random(-2, 3);
    if (resultado == num1 + num2 && operador == '+') resultado += 1; // força erro
    if (resultado == num1 - num2 && operador == '-') resultado += 1;
    if (resultado == num1 * num2 && operador == '*') resultado += 1;
  }
}

void matematica() {
  static bool iniciado = false;
  static bool resultadoMostrado = false;
```

```
static unsigned long inicioTempo = 0;

int horz0 = analogRead(HORZ_PIN0);
int horz1 = analogRead(HORZ_PIN1);

if (!iniciado) {
  iniciado = true;
  resultadoMostrado = false;
  gerarExpressaoMatematica();
  inicioTempo = millis();

  // mostra expressão
  display.clearDisplay();
  display.setTextSize(1.7);
  display.setCursor(10, 20);
  display.print(num1);
  display.print(" ");
  display.print(operador);
  display.print(F(" "));
  display.print(num2);
  display.print(F(" = "));
  display.println(resultado);
  display.println(F(" <- verdadeiro"));
  display.println(F(" -> falso"));
  display.display();
}

// Espera movimento
```

```
  if (!resultadoMostrado && millis() - inicioTempo > 800) {
    bool jogador0_falso = horz0 < 200;
    bool jogador0_verdadeiro = horz0 > 800;

    bool jogador1_falso = horz1 < 200;
    bool jogador1_verdadeiro = horz1 > 800;

    if (jogador0_falso || jogador0_verdadeiro || jogador1_falso ||
jogador1_verdadeiro) {
      if (jogador0_verdadeiro && verdadeirom) {
        mostrarVencedor(0);
        vencedor = 0;
      } else if (jogador0_falso && !verdadeirom) {
        mostrarVencedor(0);
        vencedor = 0;
      } else if (jogador1_verdadeiro && verdadeirom) {
        mostrarVencedor(1);
        vencedor = 1;
      } else if (jogador1_falso && !verdadeirom) {
        mostrarVencedor(1);
        vencedor = 1;
      } else {
        // se um errou o outro vence
        mostrarVencedor(jogador0_verdadeiro || jogador0_falso ? 1 : 0);
        vencedor = jogador0_verdadeiro || jogador0_falso ? 1 : 0;
      }

      resultadoMostrado = true;
```

```
      delay(1000);
      iniciado = false;
      cena = 1;
      display.clearDisplay();
      drawMap();
    }
  }
}

////////////////////////////TROLL/////////////////////////////////////

void pequeno () {
  ball.raio = 1;
  player.height = 10;
  cpu.height = 10;
}

void grande () {
  if(vencedor == 0)
    player.height = 30;
  else if (vencedor == 1)
    cpu.height = 30;
}

void rebote(Ball &ball, uint8_t jogador) {
  if (jogador == 0 && poderPlayer0) {
    Serial.println(F("Rebote Player0!"));
    if (ball.dirY == 0) ball.dirY = 1;
```

```
      ball.dirY = -ball.dirY; // inverte Y
    poderPlayer0 = false;
  }
  else if (jogador == 1 && poderPlayer1) {
    Serial.println(F("Rebote Player1!"));
    if (ball.dirY == 0) ball.dirY = 1;
      ball.dirY = -ball.dirY;
    poderPlayer1 = false;
  }
  //Serial.println(ball.dirY);
}
```