/\*

 \* ╔═══════════════════════════════════════════════════════════════╗

 \* ║   ESP32-C6 专业游戏合集系统 v2.0                              ║

 \* ║   10个完整游戏 + 双难度 + 独立配乐 + 最高分系统                ║

 \* ║   作者： XIU YUAN                                   ║

 \* ║   日期：2025-10-14                                               ║

 \* ╚═══════════════════════════════════════════════════════════════╝

 \*/

#include <Wire.h>

#include <Adafruit\_GFX.h>

#include <Adafruit\_SSD1306.h>

#include <Preferences.h>

// ═══════════════════════════════════════════════════════════════

// 硬件配置

// ═══════════════════════════════════════════════════════════════

#define BUZZER\_PIN 8

#define JOYSTICK\_X 1

#define JOYSTICK\_Y 2

#define JOYSTICK\_SW 3

#define OLED\_WIDTH 128

#define OLED\_HEIGHT 64

#define OLED\_RESET -1

#define SCREEN\_ADDRESS 0x3C

Adafruit\_SSD1306 display(OLED\_WIDTH, OLED\_HEIGHT, &Wire, OLED\_RESET);

Preferences preferences;

// ═══════════════════════════════════════════════════════════════

// 音符频率定义

// ═══════════════════════════════════════════════════════════════

#define NOTE\_B0  31

#define NOTE\_C1  33

#define NOTE\_CS1 35

#define NOTE\_D1  37

#define NOTE\_DS1 39

#define NOTE\_E1  41

#define NOTE\_F1  44

#define NOTE\_FS1 46

#define NOTE\_G1  49

#define NOTE\_GS1 52

#define NOTE\_A1  55

#define NOTE\_AS1 58

#define NOTE\_B1  62

#define NOTE\_C2  65

#define NOTE\_CS2 69

#define NOTE\_D2  73

#define NOTE\_DS2 78

#define NOTE\_E2  82

#define NOTE\_F2  87

#define NOTE\_FS2 93

#define NOTE\_G2  98

#define NOTE\_GS2 104

#define NOTE\_A2  110

#define NOTE\_AS2 117

#define NOTE\_B2  123

#define NOTE\_C3  131

#define NOTE\_CS3 139

#define NOTE\_D3  147

#define NOTE\_DS3 156

#define NOTE\_E3  165

#define NOTE\_F3  175

#define NOTE\_FS3 185

#define NOTE\_G3  196

#define NOTE\_GS3 208

#define NOTE\_A3  220

#define NOTE\_AS3 233

#define NOTE\_B3  247

#define NOTE\_C4  262

#define NOTE\_CS4 277

#define NOTE\_D4  294

#define NOTE\_DS4 311

#define NOTE\_E4  330

#define NOTE\_F4  349

#define NOTE\_FS4 370

#define NOTE\_G4  392

#define NOTE\_GS4 415

#define NOTE\_A4  440

#define NOTE\_AS4 466

#define NOTE\_B4  494

#define NOTE\_C5  523

#define NOTE\_CS5 554

#define NOTE\_D5  587

#define NOTE\_DS5 622

#define NOTE\_E5  659

#define NOTE\_F5  698

#define NOTE\_FS5 740

#define NOTE\_G5  784

#define NOTE\_GS5 831

#define NOTE\_A5  880

#define NOTE\_AS5 932

#define NOTE\_B5  988

#define NOTE\_C6  1047

#define NOTE\_CS6 1109

#define NOTE\_D6  1175

#define NOTE\_DS6 1245

#define NOTE\_E6  1319

#define NOTE\_F6  1397

#define NOTE\_FS6 1480

#define NOTE\_G6  1568

#define NOTE\_GS6 1661

#define NOTE\_A6  1760

#define NOTE\_AS6 1865

#define NOTE\_B6  1976

#define NOTE\_C7  2093

#define NOTE\_CS7 2217

#define NOTE\_D7  2349

#define NOTE\_DS7 2489

#define NOTE\_E7  2637

#define NOTE\_F7  2794

#define NOTE\_FS7 2960

#define NOTE\_G7  3136

#define NOTE\_GS7 3322

#define NOTE\_A7  3520

#define NOTE\_AS7 3729

#define NOTE\_B7  3951

#define NOTE\_C8  4186

#define NOTE\_CS8 4435

#define NOTE\_D8  4699

#define NOTE\_DS8 4978

#define REST 0

// ═══════════════════════════════════════════════════════════════

// 游戏状态

// ═══════════════════════════════════════════════════════════════

enum GameState {

  SPLASH\_SCREEN,

  MAIN\_MENU,

  DIFFICULTY\_SELECT,

  GAME\_SNAKE,

  GAME\_RUNNER,

  GAME\_FLAPPY,

  GAME\_BREAKOUT,

  GAME\_TETRIS,

  GAME\_INVADERS,

  GAME\_PONG,

  GAME\_MAZE,

  GAME\_RACING,

  GAME\_CATCHER,

  GAME\_OVER\_SCREEN,

  HIGH\_SCORE\_SCREEN,

  SETTINGS\_SCREEN,

  PAUSE\_SCREEN

};

GameState currentState = SPLASH\_SCREEN;

GameState pausedFrom = MAIN\_MENU;

// ═══════════════════════════════════════════════════════════════

// 全局变量

// ═══════════════════════════════════════════════════════════════

const int GAME\_COUNT = 10;

int menuSelection = 0;

int difficultySelection = 0;

bool isDifficultMode = false;

bool soundEnabled = true;

int currentScore = 0;

int currentGameIndex = 0;

const char\* gameNames[] = {

  "Snake", "Runner", "Flappy", "Breakout", "Tetris",

  "Invaders", "Pong", "Maze", "Racing", "Catcher"

};

int highScores[20]; // [0-9]简单, [10-19]困难

unsigned long lastInputTime = 0;

unsigned long musicTimer = 0;

const int INPUT\_DELAY = 180;

// ═══════════════════════════════════════════════════════════════

// 音乐系统

// ═══════════════════════════════════════════════════════════════

struct MusicNote {

  int freq;

  int dur;

};

int musicIndex = 0;

MusicNote\* currentMusic = nullptr;

int currentMusicLen = 0;

// 超级马里奥主题

MusicNote music1[] = {

  {NOTE\_E5,150},{NOTE\_E5,150},{REST,150},{NOTE\_E5,150},{REST,150},

  {NOTE\_C5,150},{NOTE\_E5,150},{REST,150},{NOTE\_G5,300},{REST,300}

};

int music1\_len = 10;

// 俄罗斯方块

MusicNote music2[] = {

  {NOTE\_E5,200},{NOTE\_B4,100},{NOTE\_C5,100},{NOTE\_D5,200},{NOTE\_C5,100},{NOTE\_B4,100},

  {NOTE\_A4,200},{NOTE\_A4,100},{NOTE\_C5,100},{NOTE\_E5,200},{NOTE\_D5,100},{NOTE\_C5,100},

  {NOTE\_B4,300},{NOTE\_C5,100},{NOTE\_D5,200},{NOTE\_E5,200}

};

int music2\_len = 16;

// 塞尔达

MusicNote music3[] = {

  {NOTE\_A4,200},{NOTE\_D5,400},{NOTE\_F5,200},{NOTE\_G5,150},{NOTE\_A5,200},

  {NOTE\_AS5,200},{NOTE\_G5,300},{NOTE\_F5,400}

};

int music3\_len = 8;

// 神庙逃亡

MusicNote music4[] = {

  {NOTE\_C5,150},{NOTE\_D5,150},{NOTE\_E5,150},{NOTE\_G5,300},

  {NOTE\_E5,150},{NOTE\_D5,150},{NOTE\_C5,300}

};

int music4\_len = 7;

// 吃豆人

MusicNote music5[] = {

  {NOTE\_B4,100},{NOTE\_B5,100},{NOTE\_FS5,100},{NOTE\_DS5,100},

  {NOTE\_B5,200},{NOTE\_FS5,200},{NOTE\_DS5,100},{NOTE\_C5,100}

};

int music5\_len = 8;

// 太空入侵者

MusicNote music6[] = {

  {NOTE\_D4,100},{NOTE\_D4,100},{NOTE\_D4,100},{NOTE\_D4,100},

  {NOTE\_A3,100},{NOTE\_A3,100},{NOTE\_A3,100},{NOTE\_A3,100}

};

int music6\_len = 8;

// Pong简约音

MusicNote music7[] = {

  {NOTE\_C5,200},{NOTE\_G4,200},{NOTE\_C5,200},{REST,200}

};

int music7\_len = 4;

// 迷宫探险

MusicNote music8[] = {

  {NOTE\_E4,200},{NOTE\_G4,200},{NOTE\_A4,200},{NOTE\_C5,400},{REST,200}

};

int music8\_len = 5;

// 赛车狂飙

MusicNote music9[] = {

  {NOTE\_G5,100},{NOTE\_E5,100},{NOTE\_C5,100},{NOTE\_G4,100},

  {NOTE\_G5,100},{NOTE\_E5,100},{NOTE\_C5,100},{NOTE\_G4,100}

};

int music9\_len = 8;

// 接星星

MusicNote music10[] = {

  {NOTE\_C5,150},{NOTE\_D5,150},{NOTE\_E5,150},{NOTE\_F5,150},{NOTE\_G5,150},{NOTE\_A5,150}

};

int music10\_len = 6;

void setMusic(int gameIdx) {

  musicIndex = 0;

  switch(gameIdx) {

    case 0: currentMusic = music1; currentMusicLen = music1\_len; break;

    case 1: currentMusic = music2; currentMusicLen = music2\_len; break;

    case 2: currentMusic = music3; currentMusicLen = music3\_len; break;

    case 3: currentMusic = music4; currentMusicLen = music4\_len; break;

    case 4: currentMusic = music5; currentMusicLen = music5\_len; break;

    case 5: currentMusic = music6; currentMusicLen = music6\_len; break;

    case 6: currentMusic = music7; currentMusicLen = music7\_len; break;

    case 7: currentMusic = music8; currentMusicLen = music8\_len; break;

    case 8: currentMusic = music9; currentMusicLen = music9\_len; break;

    case 9: currentMusic = music10; currentMusicLen = music10\_len; break;

  }

}

void playMusic() {

  if (!soundEnabled || currentMusic == nullptr) return;

  unsigned long now = millis();

  if (now - musicTimer >= currentMusic[musicIndex].dur) {

    ledcWriteTone(BUZZER\_PIN, 0);

    musicIndex = (musicIndex + 1) % currentMusicLen;

    if (currentMusic[musicIndex].freq != REST) {

      ledcWriteTone(BUZZER\_PIN, currentMusic[musicIndex].freq);

    }

    musicTimer = now;

  }

}

void stopMusic() {

  ledcWriteTone(BUZZER\_PIN, 0);

  currentMusic = nullptr;

}

void playTone(int f, int d) {

  if (!soundEnabled) return;

  ledcWriteTone(BUZZER\_PIN, f);

  delay(d);

  ledcWriteTone(BUZZER\_PIN, 0);

}

void beepSelect() { playTone(NOTE\_E5, 40); }

void beepStart() { playTone(NOTE\_C5, 60); playTone(NOTE\_E5, 60); playTone(NOTE\_G5, 100); }

void beepScore() { playTone(NOTE\_G5, 50); playTone(NOTE\_A5, 50); }

void beepGameOver() { playTone(NOTE\_E4,150); playTone(NOTE\_C4,150); playTone(NOTE\_A3,300); }

// ═══════════════════════════════════════════════════════════════

// 输入系统

// ═══════════════════════════════════════════════════════════════

int readJoystick() {

  int x = analogRead(JOYSTICK\_X);

  int y = analogRead(JOYSTICK\_Y);

  if (x > 3000) return 0; // 右

  if (x < 1000) return 2; // 左

  if (y > 3000) return 3; // 上

  if (y < 1000) return 1; // 下

  return -1;

}

bool btnPressed() {

  static bool last = false;

  bool now = digitalRead(JOYSTICK\_SW) == LOW;

  if (now && !last) {

    last = true;

    return true;

  }

  if (!now) last = false;

  return false;

}

// ═══════════════════════════════════════════════════════════════

// 数据存储

// ═══════════════════════════════════════════════════════════════

void loadScores() {

  preferences.begin("gameBox", false);

  for (int i = 0; i < 20; i++) {

    char key[10];

    sprintf(key, "hs%d", i);

    highScores[i] = preferences.getInt(key, 0);

  }

  soundEnabled = preferences.getBool("sound", true);

  preferences.end();

}

void saveScore(int gameIdx, bool hard, int score) {

  int idx = gameIdx + (hard ? 10 : 0);

  if (score > highScores[idx]) {

    highScores[idx] = score;

    preferences.begin("gameBox", false);

    char key[10];

    sprintf(key, "hs%d", idx);

    preferences.putInt(key, score);

    preferences.end();

  }

}

void saveSound() {

  preferences.begin("gameBox", false);

  preferences.putBool("sound", soundEnabled);

  preferences.end();

}

int getHighScore(int gameIdx, bool hard) {

  return highScores[gameIdx + (hard ? 10 : 0)];

}

// ═══════════════════════════════════════════════════════════════

// UI工具

// ═══════════════════════════════════════════════════════════════

void drawBorder() {

  display.drawRect(0, 0, OLED\_WIDTH, OLED\_HEIGHT, SSD1306\_WHITE);

}

void drawTitle(const char\* txt, int y = 8) {

  display.setTextSize(2);

  int16\_t x1, y1;

  uint16\_t w, h;

  display.getTextBounds(txt, 0, 0, &x1, &y1, &w, &h);

  display.setCursor((OLED\_WIDTH - w) / 2, y);

  display.print(txt);

}

void drawCenter(const char\* txt, int y, int sz = 1) {

  display.setTextSize(sz);

  int16\_t x1, y1;

  uint16\_t w, h;

  display.getTextBounds(txt, 0, 0, &x1, &y1, &w, &h);

  display.setCursor((OLED\_WIDTH - w) / 2, y);

  display.print(txt);

}

void drawProgress(int x, int y, int w, int h, int val, int max) {

  display.drawRect(x, y, w, h, SSD1306\_WHITE);

  int fill = (val \* (w - 2)) / max;

  if (fill > 0) display.fillRect(x + 1, y + 1, fill, h - 2, SSD1306\_WHITE);

}

// ═══════════════════════════════════════════════════════════════

// 启动画面

// ═══════════════════════════════════════════════════════════════

void showSplash() {

  static int frame = 0;

  if (frame == 0) {

    display.clearDisplay();

    drawBorder();

    drawTitle("GAME", 12);

    drawTitle("BOX", 30);

    display.display();

    playTone(NOTE\_C5, 80);

  } else if (frame == 8) {

    display.setTextSize(1);

    drawCenter("v2.0 Professional", 50, 1);

    display.display();

    playTone(NOTE\_E5, 80);

  } else if (frame == 16) {

    drawCenter("Loading...", 58, 1);

    display.display();

  } else if (frame > 16 && frame < 36) {

    drawProgress(15, 56, 98, 6, frame - 16, 20);

    display.display();

  } else if (frame >= 36) {

    currentState = MAIN\_MENU;

    playTone(NOTE\_G5, 100);

    frame = -1;

  }

  frame++;

  delay(50);

}

// ═══════════════════════════════════════════════════════════════

// 主菜单

// ═══════════════════════════════════════════════════════════════

void handleMainMenu() {

  display.clearDisplay();

  drawBorder();

  // 标题动画

  display.setTextSize(1);

  int titleY = 3 + sin(millis() / 200.0);

  display.setCursor(8, titleY);

  display.print("< SELECT GAME >");

  // 游戏列表

  int start = max(0, menuSelection - 2);

  int end = min(GAME\_COUNT, start + 5);

  for (int i = start; i < end; i++) {

    int y = 14 + (i - start) \* 10;

    if (i == menuSelection) {

      display.fillRoundRect(2, y - 1, OLED\_WIDTH - 4, 9, 2, SSD1306\_WHITE);

      display.setTextColor(SSD1306\_BLACK);

    } else {

      display.setTextColor(SSD1306\_WHITE);

    }

    display.setCursor(6, y);

    display.print(i + 1);

    display.print(".");

    display.print(gameNames[i]);

    // 最高分

    int hs = getHighScore(i, false);

    display.setCursor(90, y);

    if (hs > 0) display.print(hs);

  }

  display.setTextColor(SSD1306\_WHITE);

  display.drawLine(0, 57, OLED\_WIDTH, 57, SSD1306\_WHITE);

  display.setTextSize(1);

  display.setCursor(2, 58);

  display.print("<Scores  Set>");

  display.display();

  // 输入

  int dir = readJoystick();

  unsigned long now = millis();

  if (now - lastInputTime > INPUT\_DELAY) {

    if (dir == 3) {

      menuSelection = (menuSelection - 1 + GAME\_COUNT) % GAME\_COUNT;

      beepSelect();

      lastInputTime = now;

    } else if (dir == 1) {

      menuSelection = (menuSelection + 1) % GAME\_COUNT;

      beepSelect();

      lastInputTime = now;

    } else if (dir == 2) {

      currentState = HIGH\_SCORE\_SCREEN;

      beepSelect();

      lastInputTime = now;

    } else if (dir == 0) {

      currentState = SETTINGS\_SCREEN;

      beepSelect();

      lastInputTime = now;

    }

  }

  if (btnPressed()) {

    currentGameIndex = menuSelection;

    currentState = DIFFICULTY\_SELECT;

    beepStart();

    delay(200);

  }

}

// ═══════════════════════════════════════════════════════════════

// 难度选择

// ═══════════════════════════════════════════════════════════════

void handleDifficulty() {

  display.clearDisplay();

  drawBorder();

  display.setTextSize(1);

  drawCenter(gameNames[currentGameIndex], 6, 1);

  display.drawLine(10, 15, OLED\_WIDTH - 10, 15, SSD1306\_WHITE);

  drawCenter("SELECT DIFFICULTY", 20, 1);

  // 简单

  if (difficultySelection == 0) {

    display.fillRoundRect(12, 30, 104, 11, 2, SSD1306\_WHITE);

    display.setTextColor(SSD1306\_BLACK);

  } else {

    display.drawRoundRect(12, 30, 104, 11, 2, SSD1306\_WHITE);

    display.setTextColor(SSD1306\_WHITE);

  }

  drawCenter("EASY MODE", 32, 1);

  // 困难

  display.setTextColor(SSD1306\_WHITE);

  if (difficultySelection == 1) {

    display.fillRoundRect(12, 43, 104, 11, 2, SSD1306\_WHITE);

    display.setTextColor(SSD1306\_BLACK);

  } else {

    display.drawRoundRect(12, 43, 104, 11, 2, SSD1306\_WHITE);

    display.setTextColor(SSD1306\_WHITE);

  }

  drawCenter("HARD MODE", 45, 1);

  display.setTextColor(SSD1306\_WHITE);

  display.setCursor(20, 58);

  display.print("Press to start");

  display.display();

  // 输入

  int dir = readJoystick();

  if (millis() - lastInputTime > INPUT\_DELAY) {

    if (dir == 3 || dir == 1) {

      difficultySelection = 1 - difficultySelection;

      beepSelect();

      lastInputTime = millis();

    }

  }

  if (btnPressed()) {

    isDifficultMode = (difficultySelection == 1);

    currentScore = 0;

    currentState = (GameState)(GAME\_SNAKE + currentGameIndex);

    setMusic(currentGameIndex);

    // 初始化游戏

    initGame();

    beepStart();

    delay(300);

  }

}

// ═══════════════════════════════════════════════════════════════

// 游戏结束

// ═══════════════════════════════════════════════════════════════

void handleGameOver() {

  stopMusic();

  int hs = getHighScore(currentGameIndex, isDifficultMode);

  bool newRecord = (currentScore > hs);

  if (newRecord) {

    saveScore(currentGameIndex, isDifficultMode, currentScore);

  }

  display.clearDisplay();

  drawBorder();

  drawTitle("GAME", 5);

  drawTitle("OVER", 23);

  display.setTextSize(1);

  display.setCursor(10, 42);

  display.print("Score:");

  display.setCursor(70, 42);

  display.print(currentScore);

  display.setCursor(10, 50);

  display.print("Best:");

  display.setCursor(70, 50);

  display.print(newRecord ? currentScore : hs);

  if (newRecord) {

    drawCenter("NEW RECORD!", 58, 1);

  } else {

    drawCenter("Press to menu", 58, 1);

  }

  display.display();

  if (btnPressed()) {

    delay(300);

    currentState = MAIN\_MENU;

  }

}

// ═══════════════════════════════════════════════════════════════

// 高分榜

// ═══════════════════════════════════════════════════════════════

void handleHighScores() {

  static int page = 0;

  display.clearDisplay();

  drawBorder();

  display.setTextSize(1);

  drawCenter("HIGH SCORES", 4, 1);

  display.drawLine(5, 13, OLED\_WIDTH - 5, 13, SSD1306\_WHITE);

  // 表头

  display.setCursor(5, 16);

  display.print("Game");

  display.setCursor(70, 16);

  display.print("Easy");

  display.setCursor(100, 16);

  display.print("Hard");

  // 数据

  int start = page \* 4;

  for (int i = 0; i < 4 && (start + i) < GAME\_COUNT; i++) {

    int idx = start + i;

    int y = 26 + i \* 9;

    display.setCursor(5, y);

    display.print(gameNames[idx]);

    display.setCursor(70, y);

    int easy = getHighScore(idx, false);

    if (easy > 0) display.print(easy);

    else display.print("-");

    display.setCursor(100, y);

    int hard = getHighScore(idx, true);

    if (hard > 0) display.print(hard);

    else display.print("-");

  }

  display.drawLine(5, 56, OLED\_WIDTH - 5, 56, SSD1306\_WHITE);

  display.setCursor(5, 58);

  display.print("U/D:Pg ");

  display.print(page + 1);

  display.print("/3  B:Back");

  display.display();

  // 输入

  int dir = readJoystick();

  if (millis() - lastInputTime > INPUT\_DELAY) {

    if (dir == 3 && page > 0) {

      page--;

      beepSelect();

      lastInputTime = millis();

    } else if (dir == 1 && page < 2) {

      page++;

      beepSelect();

      lastInputTime = millis();

    }

  }

  if (btnPressed()) {

    currentState = MAIN\_MENU;

    page = 0;

    beepSelect();

    delay(200);

  }

}

// ═══════════════════════════════════════════════════════════════

// 设置

// ═══════════════════════════════════════════════════════════════

void handleSettings() {

  static int sel = 0;

  display.clearDisplay();

  drawBorder();

  drawTitle("SETTINGS", 6);

  display.drawLine(10, 24, OLED\_WIDTH - 10, 24, SSD1306\_WHITE);

  display.setTextSize(1);

  // 声音

  if (sel == 0) {

    display.fillRect(10, 28, 108, 10, SSD1306\_WHITE);

    display.setTextColor(SSD1306\_BLACK);

  } else {

    display.setTextColor(SSD1306\_WHITE);

  }

  display.setCursor(15, 30);

  display.print("Sound: ");

  display.print(soundEnabled ? "ON" : "OFF");

  // 清除分数

  display.setTextColor(SSD1306\_WHITE);

  if (sel == 1) {

    display.fillRect(10, 40, 108, 10, SSD1306\_WHITE);

    display.setTextColor(SSD1306\_BLACK);

  }

  display.setCursor(15, 42);

  display.print("Clear Scores");

  display.setTextColor(SSD1306\_WHITE);

  display.setCursor(10, 56);

  display.print("U/D:Sel B:Menu");

  display.display();

  // 输入

  int dir = readJoystick();

  if (millis() - lastInputTime > INPUT\_DELAY) {

    if (dir == 3 || dir == 1) {

      sel = 1 - sel;

      beepSelect();

      lastInputTime = millis();

    } else if (dir == 0 || dir == 2) {

      if (sel == 0) {

        soundEnabled = !soundEnabled;

        saveSound();

        beepSelect();

      }

      lastInputTime = millis();

    }

  }

  if (btnPressed()) {

    if (sel == 1) {

      // 清除分数确认

      display.clearDisplay();

      drawBorder();

      drawCenter("Clear all scores?", 25, 1);

      drawCenter("Press again", 38, 1);

      display.display();

      delay(1000);

      if (btnPressed()) {

        preferences.begin("gameBox", false);

        for (int i = 0; i < 20; i++) {

          char key[10];

          sprintf(key, "hs%d", i);

          preferences.putInt(key, 0);

          highScores[i] = 0;

        }

        preferences.end();

        beepSelect();

        delay(500);

      }

    } else {

      currentState = MAIN\_MENU;

      beepSelect();

      delay(200);

    }

  }

}

// ═══════════════════════════════════════════════════════════════

// 游戏 1: 贪吃蛇

// ═══════════════════════════════════════════════════════════════

struct Point { int x, y; };

Point snake[100];

int snakeLen;

Point food;

int snakeDir;

bool snakeOver;

unsigned long snakeTime;

void initSnake() {

  snakeLen = 3;

  snakeDir = 0;

  snakeOver = false;

  currentScore = 0;

  snakeTime = millis();

  snake[0] = {60, 30};

  snake[1] = {57, 30};

  snake[2] = {54, 30};

  food.x = random(0, OLED\_WIDTH / 3) \* 3;

  food.y = random(0, OLED\_HEIGHT / 3) \* 3;

}

void updateSnake() {

  if (snakeOver) {

    currentState = GAME\_OVER\_SCREEN;

    beepGameOver();

    return;

  }

  playMusic();

  // 控制

  int dir = readJoystick();

  int speed = isDifficultMode ? 80 : 130;

  if (dir != -1 && millis() - lastInputTime > 100) {

    if ((dir == 0 && snakeDir != 2) || (dir == 2 && snakeDir != 0) ||

        (dir == 1 && snakeDir != 3) || (dir == 3 && snakeDir != 1)) {

      snakeDir = dir;

      lastInputTime = millis();

    }

  }

  // 移动

  if (millis() - snakeTime > speed) {

    for (int i = snakeLen - 1; i > 0; i--) {

      snake[i] = snake[i - 1];

    }

    switch (snakeDir) {

      case 0: snake[0].x += 3; break;

      case 1: snake[0].y += 3; break;

      case 2: snake[0].x -= 3; break;

      case 3: snake[0].y -= 3; break;

    }

    // 边界

    if (snake[0].x < 0 || snake[0].x >= OLED\_WIDTH || snake[0].y < 0 || snake[0].y >= OLED\_HEIGHT) {

      snakeOver = true;

      return;

    }

    // 自撞

    for (int i = 1; i < snakeLen; i++) {

      if (snake[0].x == snake[i].x && snake[0].y == snake[i].y) {

        snakeOver = true;

        return;

      }

    }

    // 吃食物

    if (snake[0].x == food.x && snake[0].y == food.y) {

      snakeLen++;

      currentScore += 10;

      food.x = random(0, OLED\_WIDTH / 3) \* 3;

      food.y = random(0, OLED\_HEIGHT / 3) \* 3;

      beepScore();

    }

    snakeTime = millis();

  }

  // 绘制

  display.clearDisplay();

  // 蛇

  for (int i = 0; i < snakeLen; i++) {

    if (i == 0) {

      display.fillRect(snake[i].x, snake[i].y, 3, 3, SSD1306\_WHITE);

    } else {

      display.drawRect(snake[i].x, snake[i].y, 3, 3, SSD1306\_WHITE);

    }

  }

  // 食物闪烁

  if (millis() % 400 < 200) {

    display.fillRect(food.x, food.y, 3, 3, SSD1306\_WHITE);

  }

  // 分数

  display.setTextSize(1);

  display.setCursor(0, 0);

  display.print("S:");

  display.print(currentScore);

  display.display();

}

// ═══════════════════════════════════════════════════════════════

// 游戏 2: 跑酷

// ═══════════════════════════════════════════════════════════════

int runY;

int runVel;

bool runJump;

int runObsX;

int runObsH;

bool runOver;

unsigned long runTime;

void initRunner() {

  runY = 50;

  runVel = 0;

  runJump = false;

  runObsX = OLED\_WIDTH;

  runObsH = random(8, 16);

  runOver = false;

  currentScore = 0;

  runTime = millis();

}

void updateRunner() {

  if (runOver) {

    currentState = GAME\_OVER\_SCREEN;

    beepGameOver();

    return;

  }

  playMusic();

  int delay\_ms = isDifficultMode ? 25 : 40;

  if (millis() - runTime < delay\_ms) return;

  runTime = millis();

  // 跳跃

  if (btnPressed() && !runJump && runY >= 50) {

    runVel = -8;

    runJump = true;

    playTone(NOTE\_C5, 30);

  }

  // 物理

  runVel++;

  runY += runVel;

  if (runY >= 50) {

    runY = 50;

    runVel = 0;

    runJump = false;

  }

  if (runY < 10) {

    runY = 10;

    runVel = 1;

  }

  // 障碍物

  runObsX -= (isDifficultMode ? 3 : 2);

  if (runObsX < -10) {

    runObsX = OLED\_WIDTH;

    runObsH = random(8, 16);

    currentScore += 10;

    beepScore();

  }

  // 碰撞

  if (runObsX >= 8 && runObsX <= 18 && runY + 8 >= 58 - runObsH) {

    runOver = true;

    return;

  }

  // 绘制

  display.clearDisplay();

  // 地面

  display.drawLine(0, 58, OLED\_WIDTH, 58, SSD1306\_WHITE);

  for (int i = 0; i < OLED\_WIDTH; i += 8) {

    display.drawPixel((i + (millis() / 50) % 8), 59, SSD1306\_WHITE);

  }

  // 玩家

  display.fillRect(10, runY, 8, 8, SSD1306\_WHITE);

  // 障碍

  display.fillRect(runObsX, 58 - runObsH, 8, runObsH, SSD1306\_WHITE);

  // 分数

  display.setTextSize(1);

  display.setCursor(0, 0);

  display.print("S:");

  display.print(currentScore);

  display.display();

}

// ═══════════════════════════════════════════════════════════════

// 游戏 3: Flappy Bird

// ═══════════════════════════════════════════════════════════════

int birdY;

int birdVel;

int pipeX;

int pipeGap;

int pipeY;

bool flappyOver;

unsigned long flappyTime;

void initFlappy() {

  birdY = 32;

  birdVel = 0;

  pipeX = OLED\_WIDTH;

  pipeGap = isDifficultMode ? 20 : 25;

  pipeY = random(10, 40);

  flappyOver = false;

  currentScore = 0;

  flappyTime = millis();

}

void updateFlappy() {

  if (flappyOver) {

    currentState = GAME\_OVER\_SCREEN;

    beepGameOver();

    return;

  }

  playMusic();

  int delay\_ms = isDifficultMode ? 30 : 45;

  if (millis() - flappyTime < delay\_ms) return;

  flappyTime = millis();

  // 飞行

  if (btnPressed()) {

    birdVel = -3;

    playTone(NOTE\_E5, 25);

    delay(80);

  }

  // 物理

  birdVel++;

  birdY += birdVel;

  // 边界

  if (birdY < 0 || birdY > OLED\_HEIGHT - 5) {

    flappyOver = true;

    return;

  }

  // 管道移动

  pipeX -= 2;

  if (pipeX < -10) {

    pipeX = OLED\_WIDTH;

    pipeY = random(10, 40);

    currentScore += 10;

    beepScore();

  }

  // 碰撞

  if (pipeX >= 18 && pipeX <= 30) {

    if (birdY < pipeY || birdY > pipeY + pipeGap) {

      flappyOver = true;

      return;

    }

  }

  // 绘制

  display.clearDisplay();

  // 小鸟

  display.fillCircle(24, birdY, 3, SSD1306\_WHITE);

  display.drawPixel(26, birdY - 1, SSD1306\_WHITE);

  // 管道

  display.fillRect(pipeX, 0, 10, pipeY, SSD1306\_WHITE);

  display.fillRect(pipeX, pipeY + pipeGap, 10, OLED\_HEIGHT, SSD1306\_WHITE);

  // 分数

  display.setTextSize(1);

  display.setCursor(0, 0);

  display.print("S:");

  display.print(currentScore);

  display.display();

}

// ═══════════════════════════════════════════════════════════════

// 游戏 4: 打砖块

// ═══════════════════════════════════════════════════════════════

int paddleX;

int ballX, ballY;

int ballDX, ballDY;

bool bricks[10][4];

bool breakoutOver;

unsigned long breakoutTime;

void initBreakout() {

  paddleX = OLED\_WIDTH / 2 - 10;

  ballX = OLED\_WIDTH / 2;

  ballY = 50;

  ballDX = 2;

  ballDY = -2;

  breakoutOver = false;

  currentScore = 0;

  breakoutTime = millis();

  for (int i = 0; i < 10; i++) {

    for (int j = 0; j < 4; j++) {

      bricks[i][j] = true;

    }

  }

}

void updateBreakout() {

  if (breakoutOver) {

    currentState = GAME\_OVER\_SCREEN;

    beepGameOver();

    return;

  }

  playMusic();

  int delay\_ms = isDifficultMode ? 25 : 35;

  if (millis() - breakoutTime < delay\_ms) return;

  breakoutTime = millis();

  // 挡板控制

  int x = analogRead(JOYSTICK\_X);

  if (x > 3000) paddleX += (isDifficultMode ? 5 : 4);

  if (x < 1000) paddleX -= (isDifficultMode ? 5 : 4);

  paddleX = constrain(paddleX, 0, OLED\_WIDTH - 20);

  // 球移动

  ballX += ballDX;

  ballY += ballDY;

  // 边界

  if (ballX <= 0 || ballX >= OLED\_WIDTH) ballDX = -ballDX;

  if (ballY <= 0) ballDY = -ballDY;

  // 挡板反弹

  if (ballY >= OLED\_HEIGHT - 6 && ballY <= OLED\_HEIGHT - 3 &&

      ballX >= paddleX && ballX <= paddleX + 20) {

    ballDY = -ballDY;

    playTone(NOTE\_C5, 25);

  }

  // 掉落

  if (ballY >= OLED\_HEIGHT) {

    breakoutOver = true;

    return;

  }

  // 砖块碰撞

  for (int i = 0; i < 10; i++) {

    for (int j = 0; j < 4; j++) {

      if (bricks[i][j]) {

        int bx = i \* 13;

        int by = j \* 6;

        if (ballX >= bx && ballX <= bx + 12 && ballY >= by && ballY <= by + 5) {

          bricks[i][j] = false;

          ballDY = -ballDY;

          currentScore += 10;

          beepScore();

        }

      }

    }

  }

  // 绘制

  display.clearDisplay();

  // 砖块

  for (int i = 0; i < 10; i++) {

    for (int j = 0; j < 4; j++) {

      if (bricks[i][j]) {

        display.fillRect(i \* 13, j \* 6, 12, 5, SSD1306\_WHITE);

      }

    }

  }

  // 挡板

  display.fillRect(paddleX, OLED\_HEIGHT - 3, 20, 3, SSD1306\_WHITE);

  // 球

  display.fillCircle(ballX, ballY, 2, SSD1306\_WHITE);

  // 分数

  display.setTextSize(1);

  display.setCursor(0, 56);

  display.print("S:");

  display.print(currentScore);

  display.display();

}

// ═══════════════════════════════════════════════════════════════

// 游戏 5: 俄罗斯方块

// ═══════════════════════════════════════════════════════════════

const int GRID\_W = 10;

const int GRID\_H = 16;

const int BLOCK\_SIZE = 4;

byte grid[GRID\_H][GRID\_W];

int tetromino[4][4];

int tetX, tetY;

int tetType;

bool tetrisOver;

unsigned long tetrisTime;

int tetrisSpeed;

const byte shapes[7][4][4] = {

  {{0,1,0,0},{0,1,0,0},{0,1,0,0},{0,1,0,0}}, // I

  {{1,1,0,0},{1,1,0,0},{0,0,0,0},{0,0,0,0}}, // O

  {{0,1,0,0},{1,1,0,0},{1,0,0,0},{0,0,0,0}}, // S

  {{1,0,0,0},{1,1,0,0},{0,1,0,0},{0,0,0,0}}, // Z

  {{1,0,0,0},{1,1,0,0},{1,0,0,0},{0,0,0,0}}, // T

  {{1,0,0,0},{1,0,0,0},{1,1,0,0},{0,0,0,0}}, // L

  {{0,1,0,0},{0,1,0,0},{1,1,0,0},{0,0,0,0}}  // J

};

void newTetromino() {

  tetType = random(7);

  tetX = 3;

  tetY = 0;

  for (int i = 0; i < 4; i++) {

    for (int j = 0; j < 4; j++) {

      tetromino[i][j] = shapes[tetType][i][j];

    }

  }

  // 检查游戏结束

  for (int i = 0; i < 4; i++) {

    for (int j = 0; j < 4; j++) {

      if (tetromino[i][j] && grid[tetY + i][tetX + j]) {

        tetrisOver = true;

      }

    }

  }

}

bool checkCollision(int dx, int dy) {

  for (int i = 0; i < 4; i++) {

    for (int j = 0; j < 4; j++) {

      if (tetromino[i][j]) {

        int nx = tetX + j + dx;

        int ny = tetY + i + dy;

        if (nx < 0 || nx >= GRID\_W || ny >= GRID\_H) return true;

        if (ny >= 0 && grid[ny][nx]) return true;

      }

    }

  }

  return false;

}

void lockTetromino() {

  for (int i = 0; i < 4; i++) {

    for (int j = 0; j < 4; j++) {

      if (tetromino[i][j] && tetY + i >= 0) {

        grid[tetY + i][tetX + j] = 1;

      }

    }

  }

  // 消行

  int lines = 0;

  for (int i = GRID\_H - 1; i >= 0; i--) {

    bool full = true;

    for (int j = 0; j < GRID\_W; j++) {

      if (!grid[i][j]) {

        full = false;

        break;

      }

    }

    if (full) {

      lines++;

      for (int k = i; k > 0; k--) {

        for (int j = 0; j < GRID\_W; j++) {

          grid[k][j] = grid[k - 1][j];

        }

      }

      for (int j = 0; j < GRID\_W; j++) {

        grid[0][j] = 0;

      }

      i++;

    }

  }

  if (lines > 0) {

    currentScore += lines \* 100;

    beepScore();

  }

  newTetromino();

}

void rotateTetromino() {

  int temp[4][4];

  for (int i = 0; i < 4; i++) {

    for (int j = 0; j < 4; j++) {

      temp[i][j] = tetromino[3 - j][i];

    }

  }

  // 临时保存

  int old[4][4];

  for (int i = 0; i < 4; i++) {

    for (int j = 0; j < 4; j++) {

      old[i][j] = tetromino[i][j];

      tetromino[i][j] = temp[i][j];

    }

  }

  // 检查碰撞

  if (checkCollision(0, 0)) {

    for (int i = 0; i < 4; i++) {

      for (int j = 0; j < 4; j++) {

        tetromino[i][j] = old[i][j];

      }

    }

  } else {

    playTone(NOTE\_E5, 30);

  }

}

void initTetris() {

  for (int i = 0; i < GRID\_H; i++) {

    for (int j = 0; j < GRID\_W; j++) {

      grid[i][j] = 0;

    }

  }

  tetrisOver = false;

  currentScore = 0;

  tetrisSpeed = isDifficultMode ? 300 : 500;

  tetrisTime = millis();

  newTetromino();

}

void updateTetris() {

  if (tetrisOver) {

    currentState = GAME\_OVER\_SCREEN;

    beepGameOver();

    return;

  }

  playMusic();

  // 控制

  int dir = readJoystick();

  if (millis() - lastInputTime > 150) {

    if (dir == 2 && !checkCollision(-1, 0)) {

      tetX--;

      lastInputTime = millis();

    } else if (dir == 0 && !checkCollision(1, 0)) {

      tetX++;

      lastInputTime = millis();

    } else if (dir == 1) {

      tetrisSpeed = 50;

    } else if (dir == 3) {

      rotateTetromino();

      lastInputTime = millis();

    }

  }

  if (btnPressed()) {

    rotateTetromino();

    delay(150);

  }

  // 下落

  if (millis() - tetrisTime > tetrisSpeed) {

    if (!checkCollision(0, 1)) {

      tetY++;

    } else {

      lockTetromino();

    }

    tetrisTime = millis();

    tetrisSpeed = isDifficultMode ? 300 : 500;

  }

  // 绘制

  display.clearDisplay();

  // 网格边框

  display.drawRect(0, 0, GRID\_W \* 6 + 2, OLED\_HEIGHT, SSD1306\_WHITE);

  // 固定方块

  for (int i = 0; i < GRID\_H; i++) {

    for (int j = 0; j < GRID\_W; j++) {

      if (grid[i][j]) {

        display.fillRect(j \* 6 + 1, i \* 4, 5, 3, SSD1306\_WHITE);

      }

    }

  }

  // 当前方块

  for (int i = 0; i < 4; i++) {

    for (int j = 0; j < 4; j++) {

      if (tetromino[i][j] && tetY + i >= 0) {

        display.fillRect((tetX + j) \* 6 + 1, (tetY + i) \* 4, 5, 3, SSD1306\_WHITE);

      }

    }

  }

  // 分数

  display.setTextSize(1);

  display.setCursor(65, 20);

  display.print("Score");

  display.setCursor(65, 30);

  display.print(currentScore);

  display.display();

}

// ═══════════════════════════════════════════════════════════════

// 游戏 6: 太空入侵者

// ═══════════════════════════════════════════════════════════════

int shipX;

int bulletX, bulletY;

bool bulletActive;

int aliens[3][8];

int alienX, alienY;

int alienDX;

bool invadersOver;

unsigned long invadersTime;

void initInvaders() {

  shipX = OLED\_WIDTH / 2;

  bulletActive = false;

  alienX = 0;

  alienY = 10;

  alienDX = 1;

  invadersOver = false;

  currentScore = 0;

  invadersTime = millis();

  for (int i = 0; i < 3; i++) {

    for (int j = 0; j < 8; j++) {

      aliens[i][j] = 1;

    }

  }

}

void updateInvaders() {

  if (invadersOver) {

    currentState = GAME\_OVER\_SCREEN;

    beepGameOver();

    return;

  }

  playMusic();

  int delay\_ms = isDifficultMode ? 40 : 60;

  if (millis() - invadersTime < delay\_ms) return;

  invadersTime = millis();

  // 飞船控制

  int x = analogRead(JOYSTICK\_X);

  if (x > 3000) shipX += 3;

  if (x < 1000) shipX -= 3;

  shipX = constrain(shipX, 0, OLED\_WIDTH - 8);

  // 发射子弹

  if (btnPressed() && !bulletActive) {

    bulletX = shipX + 4;

    bulletY = OLED\_HEIGHT - 10;

    bulletActive = true;

    playTone(NOTE\_E5, 30);

    delay(100);

  }

  // 子弹移动

  if (bulletActive) {

    bulletY -= 3;

    if (bulletY < 0) {

      bulletActive = false;

    }

    // 碰撞检测

    for (int i = 0; i < 3; i++) {

      for (int j = 0; j < 8; j++) {

        if (aliens[i][j]) {

          int ax = alienX + j \* 12;

          int ay = alienY + i \* 10;

          if (bulletX >= ax && bulletX <= ax + 8 && bulletY >= ay && bulletY <= ay + 6) {

            aliens[i][j] = 0;

            bulletActive = false;

            currentScore += 10;

            beepScore();

          }

        }

      }

    }

  }

  // 外星人移动

  alienX += alienDX;

  if (alienX <= 0 || alienX >= 32) {

    alienDX = -alienDX;

    alienY += 5;

  }

  // 外星人到达底部

  if (alienY >= OLED\_HEIGHT - 20) {

    invadersOver = true;

    return;

  }

  // 检查胜利

  bool anyAlive = false;

  for (int i = 0; i < 3; i++) {

    for (int j = 0; j < 8; j++) {

      if (aliens[i][j]) anyAlive = true;

    }

  }

  if (!anyAlive) {

    invadersOver = true;

    currentScore += 100;

    return;

  }

  // 绘制

  display.clearDisplay();

  // 外星人

  for (int i = 0; i < 3; i++) {

    for (int j = 0; j < 8; j++) {

      if (aliens[i][j]) {

        int ax = alienX + j \* 12;

        int ay = alienY + i \* 10;

        display.fillRect(ax, ay, 8, 6, SSD1306\_WHITE);

      }

    }

  }

  // 飞船

  display.fillRect(shipX, OLED\_HEIGHT - 8, 8, 6, SSD1306\_WHITE);

  display.drawPixel(shipX + 4, OLED\_HEIGHT - 9, SSD1306\_WHITE);

  // 子弹

  if (bulletActive) {

    display.fillRect(bulletX, bulletY, 2, 4, SSD1306\_WHITE);

  }

  // 分数

  display.setTextSize(1);

  display.setCursor(0, 0);

  display.print("S:");

  display.print(currentScore);

  display.display();

}

// ═══════════════════════════════════════════════════════════════

// 游戏 7: Pong

// ═══════════════════════════════════════════════════════════════

int paddle1Y, paddle2Y;

int pongBallX, pongBallY;

int pongDX, pongDY;

int score1, score2;

bool pongOver;

unsigned long pongTime;

void initPong() {

  paddle1Y = OLED\_HEIGHT / 2 - 8;

  paddle2Y = OLED\_HEIGHT / 2 - 8;

  pongBallX = OLED\_WIDTH / 2;

  pongBallY = OLED\_HEIGHT / 2;

  pongDX = 2;

  pongDY = 1;

  score1 = 0;

  score2 = 0;

  pongOver = false;

  currentScore = 0;

  pongTime = millis();

}

void updatePong() {

  if (pongOver) {

    currentState = GAME\_OVER\_SCREEN;

    beepGameOver();

    return;

  }

  playMusic();

  int delay\_ms = isDifficultMode ? 25 : 35;

  if (millis() - pongTime < delay\_ms) return;

  pongTime = millis();

  // 玩家控制

  int y = analogRead(JOYSTICK\_Y);

  if (y > 3000) paddle1Y += 3;

  if (y < 1000) paddle1Y -= 3;

  paddle1Y = constrain(paddle1Y, 0, OLED\_HEIGHT - 16);

  // AI控制

  if (pongBallY < paddle2Y + 8) paddle2Y -= (isDifficultMode ? 2 : 1);

  if (pongBallY > paddle2Y + 8) paddle2Y += (isDifficultMode ? 2 : 1);

  paddle2Y = constrain(paddle2Y, 0, OLED\_HEIGHT - 16);

  // 球移动

  pongBallX += pongDX;

  pongBallY += pongDY;

  // 上下边界

  if (pongBallY <= 0 || pongBallY >= OLED\_HEIGHT) {

    pongDY = -pongDY;

    playTone(NOTE\_C5, 30);

  }

  // 左挡板

  if (pongBallX <= 5 && pongBallY >= paddle1Y && pongBallY <= paddle1Y + 16) {

    pongDX = -pongDX;

    playTone(NOTE\_E5, 30);

  }

  // 右挡板

  if (pongBallX >= OLED\_WIDTH - 5 && pongBallY >= paddle2Y && pongBallY <= paddle2Y + 16) {

    pongDX = -pongDX;

    playTone(NOTE\_E5, 30);

  }

  // 得分

  if (pongBallX < 0) {

    score2++;

    pongBallX = OLED\_WIDTH / 2;

    pongBallY = OLED\_HEIGHT / 2;

    pongDX = 2;

  }

  if (pongBallX > OLED\_WIDTH) {

    score1++;

    currentScore += 10;

    pongBallX = OLED\_WIDTH / 2;

    pongBallY = OLED\_HEIGHT / 2;

    pongDX = -2;

    beepScore();

  }

  // 游戏结束

  if (score2 >= 5) {

    pongOver = true;

    return;

  }

  // 绘制

  display.clearDisplay();

  // 中线

  for (int i = 0; i < OLED\_HEIGHT; i += 4) {

    display.drawPixel(OLED\_WIDTH / 2, i, SSD1306\_WHITE);

  }

  // 挡板

  display.fillRect(2, paddle1Y, 3, 16, SSD1306\_WHITE);

  display.fillRect(OLED\_WIDTH - 5, paddle2Y, 3, 16, SSD1306\_WHITE);

  // 球

  display.fillCircle(pongBallX, pongBallY, 2, SSD1306\_WHITE);

  // 分数

  display.setTextSize(1);

  display.setCursor(OLED\_WIDTH / 2 - 20, 2);

  display.print(score1);

  display.setCursor(OLED\_WIDTH / 2 + 15, 2);

  display.print(score2);

  display.display();

}

// ═══════════════════════════════════════════════════════════════

// 游戏 8: 迷宫

// ═══════════════════════════════════════════════════════════════

const int MAZE\_W = 21;

const int MAZE\_H = 16;

byte maze[MAZE\_H][MAZE\_W];

int playerMX, playerMY;

int exitMX, exitMY;

bool mazeOver;

unsigned long mazeTime;

void generateMaze() {

  // 简单迷宫生成

  for (int i = 0; i < MAZE\_H; i++) {

    for (int j = 0; j < MAZE\_W; j++) {

      if (i == 0 || i == MAZE\_H - 1 || j == 0 || j == MAZE\_W - 1) {

        maze[i][j] = 1;

      } else if (i % 2 == 0 && j % 2 == 0) {

        maze[i][j] = random(2);

      } else {

        maze[i][j] = random(3) == 0 ? 1 : 0;

      }

    }

  }

  // 确保起点和终点可达

  playerMX = 1;

  playerMY = 1;

  maze[1][1] = 0;

  exitMX = MAZE\_W - 2;

  exitMY = MAZE\_H - 2;

  maze[exitMY][exitMX] = 0;

}

void initMaze() {

  generateMaze();

  mazeOver = false;

  currentScore = 0;

  mazeTime = millis();

}

void updateMaze() {

  if (mazeOver) {

    currentState = GAME\_OVER\_SCREEN;

    beepGameOver();

    return;

  }

  playMusic();

  // 控制

  int dir = readJoystick();

  if (millis() - lastInputTime > 150) {

    int nx = playerMX;

    int ny = playerMY;

    if (dir == 0) nx++;

    if (dir == 2) nx--;

    if (dir == 1) ny++;

    if (dir == 3) ny--;

    if (nx >= 0 && nx < MAZE\_W && ny >= 0 && ny < MAZE\_H && maze[ny][nx] == 0) {

      playerMX = nx;

      playerMY = ny;

      currentScore++;

      playTone(NOTE\_C5, 20);

      lastInputTime = millis();

    }

  }

  // 到达终点

  if (playerMX == exitMX && playerMY == exitMY) {

    mazeOver = true;

    currentScore += 500;

    beepScore();

    return;

  }

  // 绘制

  display.clearDisplay();

  // 迷宫

  for (int i = 0; i < MAZE\_H; i++) {

    for (int j = 0; j < MAZE\_W; j++) {

      if (maze[i][j]) {

        display.drawPixel(j \* 6, i \* 4, SSD1306\_WHITE);

      }

    }

  }

  // 玩家

  display.fillRect(playerMX \* 6 - 1, playerMY \* 4 - 1, 3, 3, SSD1306\_WHITE);

  // 终点闪烁

  if (millis() % 500 < 250) {

    display.fillRect(exitMX \* 6 - 1, exitMY \* 4 - 1, 3, 3, SSD1306\_WHITE);

  }

  // 分数

  display.setTextSize(1);

  display.setCursor(0, 0);

  display.print("M:");

  display.print(currentScore);

  display.display();

}

// ═══════════════════════════════════════════════════════════════

// 游戏 9: 赛车

// ═══════════════════════════════════════════════════════════════

int carX;

int road[4];

int roadY[4];

bool racingOver;

unsigned long racingTime;

void initRacing() {

  carX = OLED\_WIDTH / 2 - 4;

  for (int i = 0; i < 4; i++) {

    road[i] = random(20, OLED\_WIDTH - 28);

    roadY[i] = -i \* 20;

  }

  racingOver = false;

  currentScore = 0;

  racingTime = millis();

}

void updateRacing() {

  if (racingOver) {

    currentState = GAME\_OVER\_SCREEN;

    beepGameOver();

    return;

  }

  playMusic();

  int delay\_ms = isDifficultMode ? 30 : 45;

  if (millis() - racingTime < delay\_ms) return;

  racingTime = millis();

  // 控制

  int x = analogRead(JOYSTICK\_X);

  if (x > 3000) carX += 3;

  if (x < 1000) carX -= 3;

  carX = constrain(carX, 10, OLED\_WIDTH - 18);

  // 道路移动

  for (int i = 0; i < 4; i++) {

    roadY[i] += (isDifficultMode ? 3 : 2);

    if (roadY[i] > OLED\_HEIGHT) {

      roadY[i] = -16;

      road[i] = random(20, OLED\_WIDTH - 28);

      currentScore += 10;

      if (currentScore % 50 == 0) beepScore();

    }

    // 碰撞

    if (roadY[i] >= OLED\_HEIGHT - 20 && roadY[i] <= OLED\_HEIGHT - 5) {

      if (carX >= road[i] - 8 && carX <= road[i] + 8) {

        racingOver = true;

        return;

      }

    }

  }

  // 绘制

  display.clearDisplay();

  // 道路边缘

  for (int i = 0; i < OLED\_HEIGHT; i += 8) {

    display.drawPixel(5, i + (millis() / 50) % 8, SSD1306\_WHITE);

    display.drawPixel(OLED\_WIDTH - 6, i + (millis() / 50) % 8, SSD1306\_WHITE);

  }

  // 障碍车

  for (int i = 0; i < 4; i++) {

    display.fillRect(road[i], roadY[i], 8, 12, SSD1306\_WHITE);

  }

  // 玩家车

  display.fillRect(carX, OLED\_HEIGHT - 12, 8, 12, SSD1306\_WHITE);

  // 分数

  display.setTextSize(1);

  display.setCursor(0, 0);

  display.print("S:");

  display.print(currentScore);

  display.display();

}

// ═══════════════════════════════════════════════════════════════

// 游戏 10: 接星星

// ═══════════════════════════════════════════════════════════════

int basketX;

int stars[5][2]; // x, y

bool catcherOver;

unsigned long catcherTime;

int missCount;

void initCatcher() {

  basketX = OLED\_WIDTH / 2 - 10;

  for (int i = 0; i < 5; i++) {

    stars[i][0] = random(5, OLED\_WIDTH - 5);

    stars[i][1] = -i \* 15 - random(20);

  }

  catcherOver = false;

  currentScore = 0;

  missCount = 0;

  catcherTime = millis();

}

void updateCatcher() {

  if (catcherOver) {

    currentState = GAME\_OVER\_SCREEN;

    beepGameOver();

    return;

  }

  playMusic();

  int delay\_ms = isDifficultMode ? 30 : 45;

  if (millis() - catcherTime < delay\_ms) return;

  catcherTime = millis();

  // 控制

  int x = analogRead(JOYSTICK\_X);

  if (x > 3000) basketX += 4;

  if (x < 1000) basketX -= 4;

  basketX = constrain(basketX, 0, OLED\_WIDTH - 20);

  // 星星下落

  for (int i = 0; i < 5; i++) {

    stars[i][1] += (isDifficultMode ? 2 : 1);

    if (stars[i][1] >= OLED\_HEIGHT) {

      missCount++;

      stars[i][0] = random(5, OLED\_WIDTH - 5);

      stars[i][1] = -10;

      if (missCount >= 5) {

        catcherOver = true;

        return;

      }

    }

    // 接住星星

    if (stars[i][1] >= OLED\_HEIGHT - 8 && stars[i][1] <= OLED\_HEIGHT - 3) {

      if (stars[i][0] >= basketX && stars[i][0] <= basketX + 20) {

        currentScore += 10;

        stars[i][0] = random(5, OLED\_WIDTH - 5);

        stars[i][1] = -10;

        beepScore();

      }

    }

  }

  // 绘制

  display.clearDisplay();

  // 篮子

  display.drawRect(basketX, OLED\_HEIGHT - 5, 20, 5, SSD1306\_WHITE);

  display.fillRect(basketX + 1, OLED\_HEIGHT - 4, 18, 1, SSD1306\_WHITE);

  // 星星

  for (int i = 0; i < 5; i++) {

    int sx = stars[i][0];

    int sy = stars[i][1];

    display.drawPixel(sx, sy, SSD1306\_WHITE);

    display.drawPixel(sx - 1, sy, SSD1306\_WHITE);

    display.drawPixel(sx + 1, sy, SSD1306\_WHITE);

    display.drawPixel(sx, sy - 1, SSD1306\_WHITE);

    display.drawPixel(sx, sy + 1, SSD1306\_WHITE);

  }

  // 分数和失误

  display.setTextSize(1);

  display.setCursor(0, 0);

  display.print("S:");

  display.print(currentScore);

  display.setCursor(OLED\_WIDTH - 30, 0);

  display.print("X:");

  display.print(missCount);

  display.display();

}

// ═══════════════════════════════════════════════════════════════

// 游戏调度

// ═══════════════════════════════════════════════════════════════

void initGame() {

  switch(currentState) {

    case GAME\_SNAKE: initSnake(); break;

    case GAME\_RUNNER: initRunner(); break;

    case GAME\_FLAPPY: initFlappy(); break;

    case GAME\_BREAKOUT: initBreakout(); break;

    case GAME\_TETRIS: initTetris(); break;

    case GAME\_INVADERS: initInvaders(); break;

    case GAME\_PONG: initPong(); break;

    case GAME\_MAZE: initMaze(); break;

    case GAME\_RACING: initRacing(); break;

    case GAME\_CATCHER: initCatcher(); break;

  }

}

void updateGame() {

  switch(currentState) {

    case GAME\_SNAKE: updateSnake(); break;

    case GAME\_RUNNER: updateRunner(); break;

    case GAME\_FLAPPY: updateFlappy(); break;

    case GAME\_BREAKOUT: updateBreakout(); break;

    case GAME\_TETRIS: updateTetris(); break;

    case GAME\_INVADERS: updateInvaders(); break;

    case GAME\_PONG: updatePong(); break;

    case GAME\_MAZE: updateMaze(); break;

    case GAME\_RACING: updateRacing(); break;

    case GAME\_CATCHER: updateCatcher(); break;

  }

}

// ═══════════════════════════════════════════════════════════════

// 主程序

// ═══════════════════════════════════════════════════════════════

void setup() {

  Serial.begin(115200);

  Serial.println("\n╔═══════════════════════════════════╗");

  Serial.println("║  ESP32-C6 Game Box v2.0          ║");

  Serial.println("║  Professional Edition            ║");

  Serial.println("╚═══════════════════════════════════╝\n");

  // 初始化OLED

  Wire.begin(21, 22);

  if (!display.begin(SSD1306\_SWITCHCAPVCC, SCREEN\_ADDRESS)) {

    Serial.println("❌ OLED initialization failed!");

    while (1);

  }

  Serial.println("✓ OLED initialized");

  // 初始化蜂鸣器

  ledcAttach(BUZZER\_PIN, 2000, 8);

  Serial.println("✓ Buzzer initialized");

  // 初始化摇杆

  pinMode(JOYSTICK\_SW, INPUT\_PULLUP);

  analogReadResolution(12);

  Serial.println("✓ Joystick initialized");

  // 加载数据

  loadScores();

  Serial.println("✓ Scores loaded");

  display.clearDisplay();

  display.display();

  randomSeed(analogRead(0));

  Serial.println("\n🎮 System ready! Enjoy gaming!\n");

}

void loop() {

  switch(currentState) {

    case SPLASH\_SCREEN:

      showSplash();

      break;

    case MAIN\_MENU:

      handleMainMenu();

      break;

    case DIFFICULTY\_SELECT:

      handleDifficulty();

      break;

    case GAME\_OVER\_SCREEN:

      handleGameOver();

      break;

    case HIGH\_SCORE\_SCREEN:

      handleHighScores();

      break;

    case SETTINGS\_SCREEN:

      handleSettings();

      break;

    default:

      updateGame();

      break;

  }

  delay(10);

}