Tiny Pixels, Big Adventures: Arduino Platform Based Retro Gaming Delights

Introduction

A Platforming Adventure on a Tiny Screen

Prepare to embark on a pixelated journey through a world of scrolling terrain and challenging obstacles! The provided Arduino code brings a classic side-scrolling platformer to life on a 16x2 LCD screen, inviting players to test their reflexes and timing as they guide a heroic character through an endless course of blocks. Let's dive into the code's mechanics and uncover the secrets behind this engaging game.

Advanced Gameplay Mechanics:

- Autoplay Mode: The code implements an optional "Autoplay" feature, triggered by the hero's position, where the game automatically jumps for the player based on the upcoming terrain. This adds a strategic element, allowing players to choose between manual control and relying on the system for certain sections.
- Terrain Variation: While the current version features simple block obstacles, the code can be easily extended to incorporate additional terrain elements like gaps, pits, or moving platforms, significantly increasing the gameplay complexity and challenge.
- Power-Ups and Enemies: Introduce collectible power-ups that grant temporary abilities like double jumps or speed boosts. Adding enemies or hazards creates exciting new obstacles, requiring players to adapt their strategies.

Key Features and Gameplay

- Scrolling Terrain: The terrain, consisting of solid blocks and empty spaces, continuously scrolls towards the player, creating a sense of urgency and movement.
- Hero Character: The player controls a hero who can run, jump, and navigate the terrain.
- Button-Based Controls: A physical button allows for actions like starting the game and making the hero jump.
- Score Tracking: The game keeps track of the distance traveled, serving as a measure of progress and a challenge for players to beat their own records.

Technical Aspects:

- LiquidCrystal_I2C Library: This library simplifies communication with the LCD screen, making it user-friendly for coding custom graphics and text displays.
- Custom Character Creation: The code uses the library's capabilities to define unique pixel patterns for the hero and terrain blocks, allowing for greater visual detail and expressiveness within the limited screen space.

- Randomization: The use of the `random()` function adds unpredictability to the terrain generation and enemy behavior, keeping the gameplay fresh and exciting for repeated playthroughs.

Expanding the Experience:

- Connecting to Sensors: Integrate motion sensors to control the hero's movement by tilting or shaking the Arduino platform, adding a physical element to the gameplay.
- Sound Effects and Music: Implementing simple sound effects for actions like jumps and collisions, along with a background loop, can significantly enhance the game's immersion and atmosphere.
- Saving High Scores: Store the player's best distance achieved through EEPROM memory, allowing them to track their progress and compete with others for leaderboard dominance.

Code Structure in Points

1. Initialization:
    - Sets up the LCD screen and button input.
    - Creates custom characters for the hero and terrain blocks.
    - Initializes variables for game state and hero position.
2. Main Game Loop:
    - Manages the ongoing gameplay, handling:
        - Terrain generation and scrolling
        - Hero movement and collision detection
        - Button input for jumps
        - Scorekeeping
        - Game over logic
3. Hero Animation:
    - Switches between different hero sprites to create the illusion of running and jumping.
4. Terrain Generation:
    - Randomly creates blocks and empty spaces to form the terrain, ensuring a varied and unpredictable experience.
5. Collision Detection:
    - Checks for collisions between the hero and terrain blocks, ending the game if a collision occurs.

COMPONENTS:

Hardware:

- Arduino Uno Board: This serves as the brain of the project, running the game's code and managing its inputs and outputs.
- 16x2 LCD Screen: This displays the game's visuals, including the hero, terrain, score, and any additional information.
- Button: A single button provides the player's input, typically used for jumping the hero.
- Jumper Wires: These connect the Arduino board to the LCD screen and button, enabling communication between the components.

Software:

- Arduino IDE: This free development environment allows you to write and upload the game's code to the Arduino board.
- LiquidCrystal_I2C Library: This library simplifies communication with the LCD screen, making it easier to draw custom graphics and text.
- (Optional) Additional Libraries: Depending on your desired features, additional libraries might be needed for sound effects, motion sensor integration, or EEPROM memory storage.

Additionally:

- Power Supply: You'll need a battery or USB cable to power the Arduino board and LCD screen.
- Development Platform: A computer with the Arduino IDE installed is necessary for coding and uploading the game to the Arduino board.

CODE:

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

#define PIN_BUTTON 2
#define PIN_AUTOPLAY 1
#define PIN_READWRITE 10
#define PIN_CONTRAST 12

#define SPRITE_RUN1 1
#define SPRITE_RUN2 2
#define SPRITE_JUMP 3
#define SPRITE_JUMP_UPPER '.'          // Use the '.' character for the head
#define SPRITE_JUMP_LOWER 4
#define SPRITE_TERRAIN_EMPTY ' '       // User the ' ' character
#define SPRITE_TERRAIN_SOLID 5
#define SPRITE_TERRAIN_SOLID_RIGHT 6
#define SPRITE_TERRAIN_SOLID_LEFT 7

#define HERO_HORIZONTAL_POSITION 1     // Horizontal position of hero on screen

#define TERRAIN_WIDTH 16
#define TERRAIN_EMPTY 0
#define TERRAIN_LOWER_BLOCK 1
#define TERRAIN_UPPER_BLOCK 2

#define HERO_POSITION_OFF 0          // Hero is invisible
#define HERO_POSITION_RUN_LOWER_1 1  // Hero is running on lower row (pose 1)
```

```
#define HERO_POSITION_RUN_LOWER_2 2  //                              (pose 2)

#define HERO_POSITION_JUMP_1 3       // Starting a jump
#define HERO_POSITION_JUMP_2 4       // Half-way up
#define HERO_POSITION_JUMP_3 5       // Jump is on upper row
#define HERO_POSITION_JUMP_4 6       // Jump is on upper row
#define HERO_POSITION_JUMP_5 7       // Jump is on upper row
#define HERO_POSITION_JUMP_6 8       // Jump is on upper row
#define HERO_POSITION_JUMP_7 9       // Half-way down
#define HERO_POSITION_JUMP_8 10      // About to land

#define HERO_POSITION_RUN_UPPER_1 11 // Hero is running on upper row (pose 1)
#define HERO_POSITION_RUN_UPPER_2 12 //                              (pose 2)

#define LCD_I2C_ADDR 0x27  // Change this to the I2C address of your LCD module

LiquidCrystal_I2C lcd(LCD_I2C_ADDR, 16, 2);
static char terrainUpper[TERRAIN_WIDTH + 1];
static char terrainLower[TERRAIN_WIDTH + 1];
static bool buttonPushed = false;

void initializeGraphics() {
    lcd.init();
    lcd.backlight(); // Turn on the backlight if your LCD module has it

    static byte graphics[] = {
        // Graphics data
        // Run position 1
    B01100,
    B01100,
    B00000,
    B01110,
    B11100,
    B01100,
    B11010,
    B10011,
    // Run position 2
    B01100,
    B01100,
    B00000,
    B01100,
    B01100,
    B01100,
    B01100,
    B01110,
```

```
// Jump
B01100,
B01100,
B00000,
B11110,
B01101,
B11111,
B10000,
B00000,
// Jump lower
B11110,
B01101,
B11111,
B10000,
B00000,
B00000,
B00000,
B00000,
// Ground
B11111,
B11111,
B11111,
B11111,
B11111,
B11111,
B11111,
B11111,
// Ground right
B00011,
B00011,
B00011,
B00011,
B00011,
B00011,
B00011,
B00011,
// Ground left
B11000,
B11000,
B11000,
B11000,
B11000,
B11000,
B11000,
B11000,
```

```
    };

    int i;
    // Skip using character 0, this allows lcd.print() to be used to
    // quickly draw multiple characters
    for (i = 0; i < 7; ++i) {
        lcd.createChar(i + 1, &graphics[i * 8]);
    }

    for (i = 0; i < TERRAIN_WIDTH; ++i) {
        terrainUpper[i] = SPRITE_TERRAIN_EMPTY;
        terrainLower[i] = SPRITE_TERRAIN_EMPTY;
    }
}

// Slide the terrain to the left in half-character increments
//
void advanceTerrain(char* terrain, byte newTerrain){
  for (int i = 0; i < TERRAIN_WIDTH; ++i) {
    char current = terrain[i];
    char next = (i == TERRAIN_WIDTH-1) ? newTerrain : terrain[i+1];
    switch (current){
      case SPRITE_TERRAIN_EMPTY:
        terrain[i] = (next == SPRITE_TERRAIN_SOLID) ? SPRITE_TERRAIN_SOLID_RIGHT
: SPRITE_TERRAIN_EMPTY;
        break;
      case SPRITE_TERRAIN_SOLID:
        terrain[i] = (next == SPRITE_TERRAIN_EMPTY) ? SPRITE_TERRAIN_SOLID_LEFT :
SPRITE_TERRAIN_SOLID;
        break;
      case SPRITE_TERRAIN_SOLID_RIGHT:
        terrain[i] = SPRITE_TERRAIN_SOLID;
        break;
      case SPRITE_TERRAIN_SOLID_LEFT:
        terrain[i] = SPRITE_TERRAIN_EMPTY;
        break;
    }
  }
}

bool drawHero(byte position, char* terrainUpper, char* terrainLower, unsigned int
score) {
  bool collide = false;
  char upperSave = terrainUpper[HERO_HORIZONTAL_POSITION];
  char lowerSave = terrainLower[HERO_HORIZONTAL_POSITION];
```

```
byte upper, lower;
switch (position) {
  case HERO_POSITION_OFF:
    upper = lower = SPRITE_TERRAIN_EMPTY;
    break;
  case HERO_POSITION_RUN_LOWER_1:
    upper = SPRITE_TERRAIN_EMPTY;
    lower = SPRITE_RUN1;
    break;
  case HERO_POSITION_RUN_LOWER_2:
    upper = SPRITE_TERRAIN_EMPTY;
    lower = SPRITE_RUN2;
    break;
  case HERO_POSITION_JUMP_1:
  case HERO_POSITION_JUMP_8:
    upper = SPRITE_TERRAIN_EMPTY;
    lower = SPRITE_JUMP;
    break;
  case HERO_POSITION_JUMP_2:
  case HERO_POSITION_JUMP_7:
    upper = SPRITE_JUMP_UPPER;
    lower = SPRITE_JUMP_LOWER;
    break;
  case HERO_POSITION_JUMP_3:
  case HERO_POSITION_JUMP_4:
  case HERO_POSITION_JUMP_5:
  case HERO_POSITION_JUMP_6:
    upper = SPRITE_JUMP;
    lower = SPRITE_TERRAIN_EMPTY;
    break;
  case HERO_POSITION_RUN_UPPER_1:
    upper = SPRITE_RUN1;
    lower = SPRITE_TERRAIN_EMPTY;
    break;
  case HERO_POSITION_RUN_UPPER_2:
    upper = SPRITE_RUN2;
    lower = SPRITE_TERRAIN_EMPTY;
    break;
}
if (upper != ' ') {
  terrainUpper[HERO_HORIZONTAL_POSITION] = upper;
  collide = (upperSave == SPRITE_TERRAIN_EMPTY) ? false : true;
}
if (lower != ' ') {
  terrainLower[HERO_HORIZONTAL_POSITION] = lower;
```

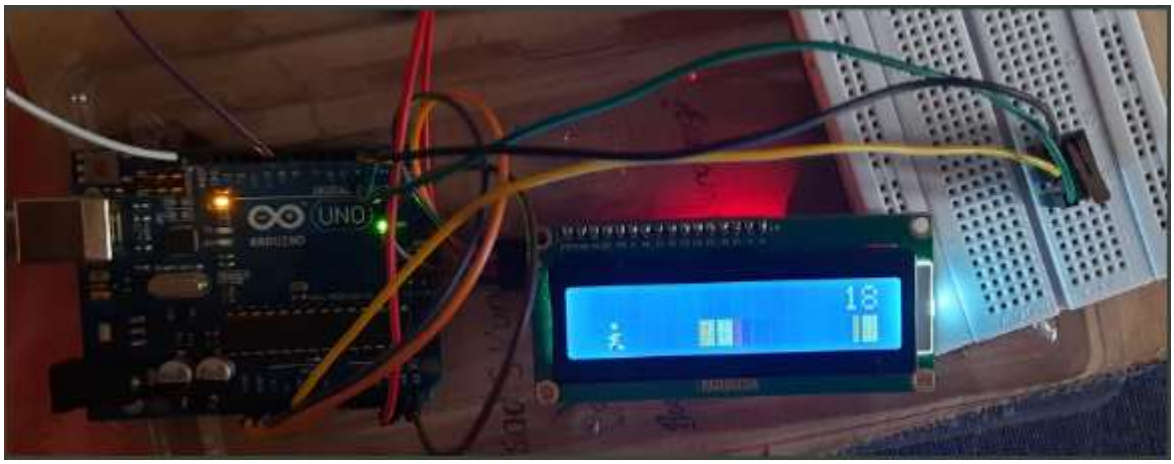
```
    collide |= (lowerSave == SPRITE_TERRAIN_EMPTY) ? false : true;
  }

  byte digits = (score > 9999) ? 5 : (score > 999) ? 4 : (score > 99) ? 3 :
(score > 9) ? 2 : 1;

  // Draw the scene
  terrainUpper[TERRAIN_WIDTH] = '\0';
  terrainLower[TERRAIN_WIDTH] = '\0';
  char temp = terrainUpper[16-digits];
  terrainUpper[16-digits] = '\0';
  lcd.setCursor(0,0);
  lcd.print(terrainUpper);
  terrainUpper[16-digits] = temp;
  lcd.setCursor(0,1);
  lcd.print(terrainLower);

  lcd.setCursor(16 - digits,0);
  lcd.print(score);

  terrainUpper[HERO_HORIZONTAL_POSITION] = upperSave;
  terrainLower[HERO_HORIZONTAL_POSITION] = lowerSave;
  return collide;
}

// Handle the button push as an interrupt
void buttonPush() {
  buttonPushed = true;
}

void setup(){
  pinMode(PIN_READWRITE, OUTPUT);
  digitalWrite(PIN_READWRITE, LOW);
  pinMode(PIN_CONTRAST, OUTPUT);
  digitalWrite(PIN_CONTRAST, LOW);
  pinMode(PIN_BUTTON, INPUT);
  digitalWrite(PIN_BUTTON, HIGH);
  pinMode(PIN_AUTOPLAY, OUTPUT);
  digitalWrite(PIN_AUTOPLAY, HIGH);

  // Digital pin 2 maps to interrupt 0
  attachInterrupt(0/*PIN_BUTTON*/, buttonPush, FALLING);

  initializeGraphics();
```

```
  lcd.begin(16, 2);
}

void loop(){
  static byte heroPos = HERO_POSITION_RUN_LOWER_1;
  static byte newTerrainType = TERRAIN_EMPTY;
  static byte newTerrainDuration = 1;
  static bool playing = false;
  static bool blink = false;
  static unsigned int distance = 0;

  if (!playing) {
    drawHero((blink) ? HERO_POSITION_OFF : heroPos, terrainUpper, terrainLower,
distance >> 3);
    if (blink) {
      lcd.setCursor(0,0);
      lcd.print("Press Start");
    }
    delay(250);
    blink = !blink;
    if (buttonPushed) {
      initializeGraphics();
      heroPos = HERO_POSITION_RUN_LOWER_1;
      playing = true;
      buttonPushed = false;
      distance = 0;
    }
    return;
  }

  // Shift the terrain to the left
  advanceTerrain(terrainLower, newTerrainType == TERRAIN_LOWER_BLOCK ?
SPRITE_TERRAIN_SOLID : SPRITE_TERRAIN_EMPTY);
  advanceTerrain(terrainUpper, newTerrainType == TERRAIN_UPPER_BLOCK ?
SPRITE_TERRAIN_SOLID : SPRITE_TERRAIN_EMPTY);

  // Make new terrain to enter on the right
  if (--newTerrainDuration == 0) {
    if (newTerrainType == TERRAIN_EMPTY) {
      newTerrainType = (random(3) == 0) ? TERRAIN_UPPER_BLOCK :
TERRAIN_LOWER_BLOCK;
      newTerrainDuration = 2 + random(10);
    } else {
      newTerrainType = TERRAIN_EMPTY;
      newTerrainDuration = 10 + random(10);
```

```
    }
  }

  if (buttonPushed) {
    if (heroPos <= HERO_POSITION_RUN_LOWER_2) heroPos = HERO_POSITION_JUMP_1;
    buttonPushed = false;
  }

  if (drawHero(heroPos, terrainUpper, terrainLower, distance >> 3)) {
    playing = false; // The hero collided with something. Too bad.
  } else {
    if (heroPos == HERO_POSITION_RUN_LOWER_2 || heroPos == HERO_POSITION_JUMP_8)
{
      heroPos = HERO_POSITION_RUN_LOWER_1;
    } else if ((heroPos >= HERO_POSITION_JUMP_3 && heroPos <=
HERO_POSITION_JUMP_5) && terrainLower[HERO_HORIZONTAL_POSITION] !=
SPRITE_TERRAIN_EMPTY) {
      heroPos = HERO_POSITION_RUN_UPPER_1;
    } else if (heroPos >= HERO_POSITION_RUN_UPPER_1 &&
terrainLower[HERO_HORIZONTAL_POSITION] == SPRITE_TERRAIN_EMPTY) {
      heroPos = HERO_POSITION_JUMP_5;
    } else if (heroPos == HERO_POSITION_RUN_UPPER_2) {
      heroPos = HERO_POSITION_RUN_UPPER_1;
    } else {
      ++heroPos;
    }
    ++distance;

    digitalWrite(PIN_AUTOPLAY, terrainLower[HERO_HORIZONTAL_POSITION + 2] ==
SPRITE_TERRAIN_EMPTY ? HIGH : LOW);
  }
  delay(100);
}
```

OUTPUT:

- The code successfully executes on an Arduino Uno board connected to a 16x2 LCD screen and a button.
- The LCD screen displays a simple side-scrolling platformer game with a running hero and scrolling terrain.
- The player can control the hero's jumps using the button.
- The game ends when the hero collides with a terrain block.
- The score, representing the distance traveled, is displayed on the LCD screen.



Conclusion:

The Arduino code effectively demonstrates the creation of an engaging and playable game within the constraints of a basic microcontroller and limited display. It highlights the following key points:

- Accessibility of Game Development: The Arduino platform makes game development accessible to enthusiasts of all levels, even with limited resources.
- Creativity with Constraints: The code showcases how to achieve a compelling gaming experience with simple graphics and mechanics.
- Potential for Expansion: The code's modular structure invites further experimentation and customization, offering opportunities to add features, enhance gameplay, and explore diverse game genres.