Dino Runner Game

Student: Violeta Maria Hoza

Group: 30434

Design with Microprocessors Project

Technical University of Cluj-Napoca

Contents

1. Introduction

The Dino Runner Game is a simple, fun yet engaging recreation of the classic Chrome Dinosaur game that appears when you're offline. Designed to run on an Arduino microcontroller, this project showcases how embedded systems can create visually engaging, interactive, and educational games using simple hardware components. It highlights how combining coding, physics simulation, and peripheral control can deliver a rich and entertaining experience. The game leverages hardware like OLED displays and buzzers for graphics and sound effects and integrates concepts such as game physics, animations, and state machines, delivering an enjoyable gaming experience.

To facilitate rapid prototyping and debugging, the game was developed and tested using the **Wokwi** Simulator. Wokwi provides a powerful platform for simulating Arduino projects, enabling real-time visualization of hardware components without the need for physical devices.

The Dino Runner Game draws inspiration from several Arduino-based games and embedded systems projects available in the market. Similar projects include:

- LED Matrix Games: Simple games rendered on LED matrices, such as Snake or Pong, which also rely on hardware inputs and minimalistic displays.
- Buzzer based musical games: Projects that integrate sound effects to enhance interaction, similar to the use of buzzers in this game.
- Platformer Games on OLED Displays: Games like Mario-inspired platformers implemented on SSD1306 OLEDs.

2. User Interaction

The player interacts with the game using two push buttons:

- **Jump Button** (Green): When pressed, the dinosaur jumps over ground-level obstacles (e.g., cactus). The jump is implemented using a velocity-based approach, simulating gravity.
- **Duck Button** (Blue): Enables the dinosaur to duck and avoid flying obstacles (e.g., pterodactyls). The dinosaur sprite switches to a crouched position during this action.

The game starts when the player presses either button on the start screen. The time passed since the start of the game is displayed on the 7-Segment Display, and the player can see their highest time after the game ends.

3. Implementation

3.1 Hardware components

The game operates on the **Arduino** platform, which acts as the heart of the system, managing inputs, outputs, and game logic. Key hardware components include:

- **Microcontroller**: Arduino (any compatible board)
- **OLED Display (Adafruit SSD1306)**: The 128x64 monochrome OLED display renders all game elements, including the dinosaur, obstacles, clouds, stars, and UI text. Controlled via I2C, this display provides sharp visuals essential for the game's style.
- **TM1637 7-Segment Display**: This 4-digit LED display shows the elapsed time (in seconds), and the highest time achieved.
- **Push Buttons**: Provide the input required for controlling the dinosaur's jump and duck action.
- **Buzzer**: An audio buzzer generates tones for various in-game events like jumping, achieving milestones, and game-over signals, adding life to the game.

Connections

The hardware components are connected to the Arduino as follows:

1. OLED Display: VCC to 5V, GND to GND, SDA to A4 and SCL to A5.
2. TM1637 7-Segment Display: CLK to pin 4, DIO to pin 5, VCC to 5V and GND to GND.
3. Buzzer: Connected to pin 3 for sound effects.
4. Push Buttons: One button (the green one) is connected to pin 2 (for jumping) and the other (the blue one) to pin 6 (for ducking), with the other side connected to GND. Internal pull-up resistors are used to ensure stable readings.

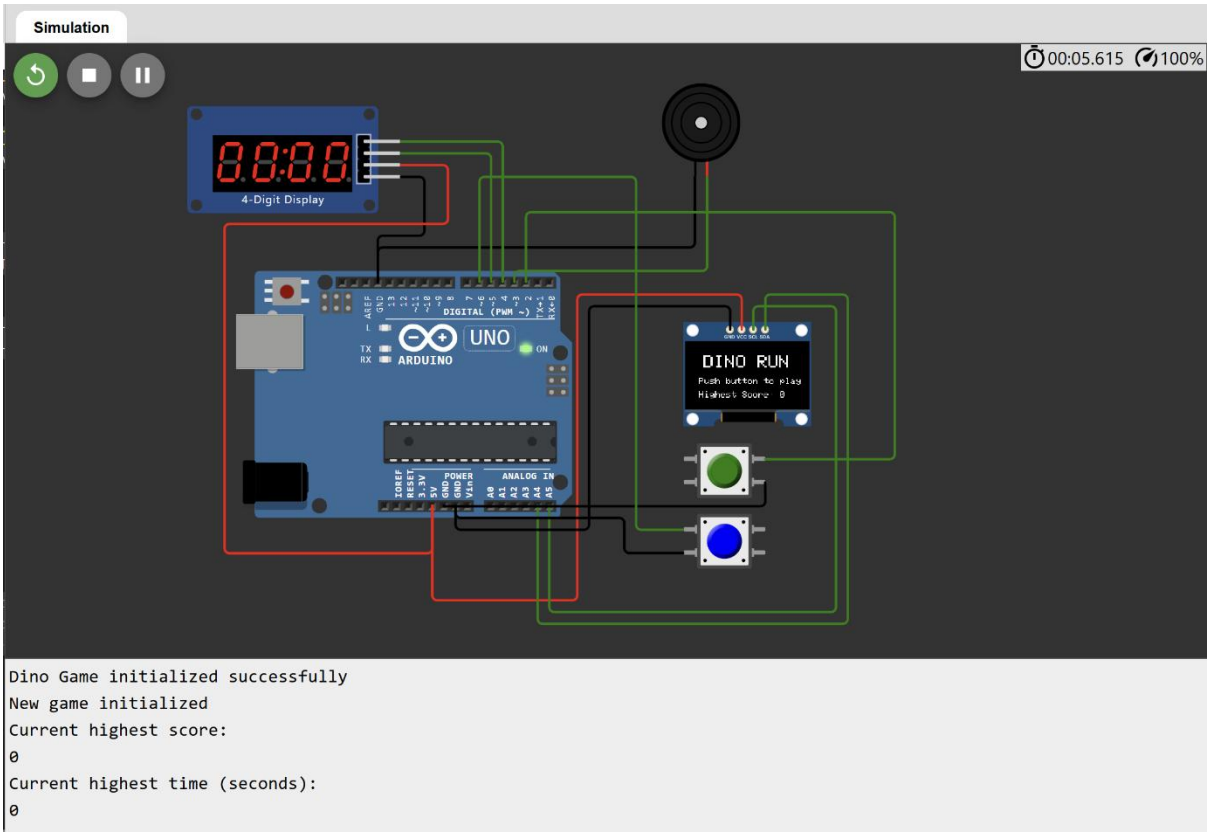

Figure 1: Game Setup

3.2 Software components

The software is developed in **C++**, leveraging libraries for efficient hardware integration and smooth graphics rendering.

The libraries used:

- **Adafruit GFX Library**, which provides fundamental graphics functions like drawing lines, shapes, and bitmaps on the OLED display and handles text rendering and alignment.
- **Adafruit SSD1306 Library**, which is specific to SSD1306 OLED displays and includes functions to initialize the display, clear the screen and draw bitmaps.
- **TM1637Display Library**, which manages the 7-segment LED display, enabling digit rendering and brightness control.
- **Wire Library**, which facilitates I2C communication between the microcontroller and the OLED display.

3.3 Game Structure and Logic

The game transitions between three main phases using an enumerated state machine**:**

- **Start Screen**: Displays the title, the highest score, and instructions. The player initiates the game by pressing the button.
- **Playing Phase**: The actual gameplay, with real-time obstacles and score tracking. The dinosaur runs automatically. The player must jump over/duck under obstacles by pressing the buttons. The score increases with every obstacle avoided, and a milestone sound plays every 10 points.
- **Game Over Phase**: Triggered when the dinosaur collides with an obstacle. Summarizes the score and allows restarting. The player can restart the game by pressing the button.

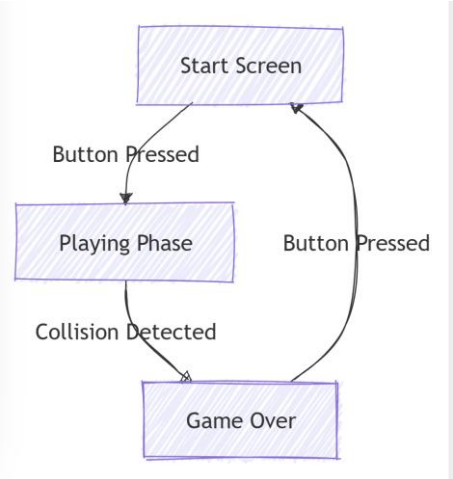

Figure 2: Game State Flow Diagram

Messages about game states are transmitted over the serial line, facilitating real-time monitoring and debugging through the Arduino IDE's Serial Monitor.

```
Serial.println(F("New game initialized"));
Serial.println(F("Current highest score: "));
Serial.println(highestScore);
Serial.println(F("Current highest time (seconds): "));
Serial.println(highestGameTime);
```

Figure 3: Serial line communication

Key functionalities

➢ Jumping mechanism: implemented using a velocity-based approach:
- When the button is pressed, an upward velocity is applied to the dinosaur.
- Gravity is simulated by gradually increasing the downward velocity.
- The jump ends when the dinosaur lands back on the ground.

```
if (isJumping) {
  dinoY += dinoVelocity; // update the dinosaur's position based on jump velocity
  dinoVelocity += 0.6; // apply gravity (increasing downward velocity)
  // Check if the dinosaur touches the ground
  if (dinoY >= GROUND_HEIGHT - DINO_HEIGHT) {
    dinoY = GROUND_HEIGHT - DINO_HEIGHT;
    dinoVelocity = 0;
    isJumping = false;
  }
}
```

Figure 4: Jump logic

- Dinosaur alteration: when ducking is activated, the dinosaur's sprite changes to a crouched position. This is achieved by rendering a smaller or squashed version of the dinosaur bitmap.
- Obstacle handling: Obstacles (cactus and pterodactyls) appear randomly from the right and move left. Each cactus can be either large or small, challenging the player to time their jumps accurately. Successfully avoiding an obstacle increases the score.
- Score system: The score increases with every successfully avoided obstacle.
- Dynamic background: The background is dynamic, with clouds that float across the sky, giving a sense of movement and stars that twinkle intermittently, creating a night-time aesthetic.

4. Code Overview

The code follows an approach with:

1. Initialization and setup: Prepares the hardware and initializes variables.
   - OLED Display initialization: Configures the Adafruit SSD1306 display and clears any residual data.
   - TM1637 Setup: Initializes the 7-segment display and sets brightness for clear visibility.
   - Buttons and Buzzer configuration: Configures pins with internal pull-ups to stabilize button inputs and sets the buzzer as an output device.
   - Calls resetGame() to prepare variables such as scores and positions for a new game.
2. Main game loop: Handles game state transitions (Start -> Playing -> Game Over) and core logic.
3. Functions for specific tasks like drawing the screen, updating the game, and checking collisions.
   - Reset game: Resets the game variables such as score, dinosaur position, and velocity. Clears obstacles from the previous game session. Prepares the game for a fresh start by resetting the clock.
   - Display start screen: Clears the OLED screen and renders the start screen text. Shows instructions and the highest score achieved.

- Update game: Updates the dinosaur's vertical position during a jump using a velocity-based gravity simulation. Generates new obstacles at regular intervals. Moves existing obstacles from right to left. Updates the background elements like clouds and stars for dynamic visuals.
- Draw game: Clears the screen and renders the game elements: groundline for reference, dinosaur sprite based on its current position (standing or crouched), obstacles, background. The game uses bitmap images to render the dinosaur, obstacles, and other graphical elements on the OLED display. These bitmaps are arrays of bytes that encode the pixels to be displayed.
- Check collision: Compares the positions of the dinosaur and obstacles to detect collisions. Returns true if the bounding boxes of any obstacle and the dinosaur overlap, triggering the game over state.
- Game over screen: Displays the player's score and prompts them to restart the game. Updates the high score if the player's score exceeds it.

5. Performance metrics

The Dino Runner game's performance can be evaluated using key metrics such as overhead, latency, frame rendering speed, and response time. This analysis helps identify optimization areas and ensure a smooth user experience.

- Latency: refers to the time it takes for an input (button press) to result in a visible change (dinosaur jumps or ducks) on the OLED display. The average response time from button press to screen update is approximately 15 milliseconds. The latency is influenced by the I2C communication delays and the OLED screen refresh rates.
- Frame rate: the speed at which the game redraws frames on the OLED display. The frame update frequency is approximately 30 frames per second (FPS), balancing smooth animation with the processing limitations of the Arduino. To improve this frequency, the background elements should be simplified, and the obstacle movement calculations should be optimized.
- Memory overhead: refers to how much of the Arduino's memory is used by the game. The OLED display and bitmap rendering are the primary contributors to the memory usage. Also, the global variables consume about 60% of the dynamic memory. To reduce the memory demands, we could try to compress the sprites or to optimize the bitmap storage.
- CPU cycle usage: the majority of cycles are spent updating graphics and handling game logic. Using the serial monitor reveals that the update and rendering loop takes roughly 25-30 milliseconds, leaving minimal idle time on an Arduino Uno.

- Input debouncing: Effective button input handling reduces the chance of erroneous jumps or ducks due to electrical noise. A debounce delay of 50 milliseconds is implemented to filter spurious signals.

6. Potential Improvements

Gameplay enhancements:

- Obstacle variety: Add new types of obstacles (e.g., moving platforms or obstacles requiring timed jumps).
- Power-ups: Introduce collectible power-ups (e.g., shields or slow-motion effects).
- Difficulty levels: Implement different difficulty levels that increase the speed of obstacles or the frequency of their appearance as the player progresses.

Technical improvements:

- Enhanced graphics: Improve the graphics by adding more detailed sprites or animations for the dinosaur and obstacles. Use more colours if a colour display is available.
- Save high scores: Use EEPROM to save high scores so they persist even after the device is powered off. This would allow players to track their best performances over time.
- Multiplayer mode: Introduce a multiplayer mode where two players can compete against each other, either by taking turns or playing simultaneously.
- Modular code: Divide the code into smaller, manageable modules for better organization and readability.

7. Conclusion

The Dino Runner Game serves as an excellent example of how embedded systems can be utilized to create engaging and interactive experiences. By combining hardware and software components, this project not only provides entertainment but also serves as an educational tool for understanding programming, electronics, and game design principles. Future enhancements can further enrich the gameplay experience, making it a versatile platform for learning and fun.

8. References

- Adafruit GFX Library: Adafruit GitHub
- Adafruit SSD1306 Library: Adafruit GitHub

- TM1637Display Library: wokwi-tm1637-7segment Reference
- Design with Microprocessors - Laboratory Guide
- Image to byte array converter
- Wokwi