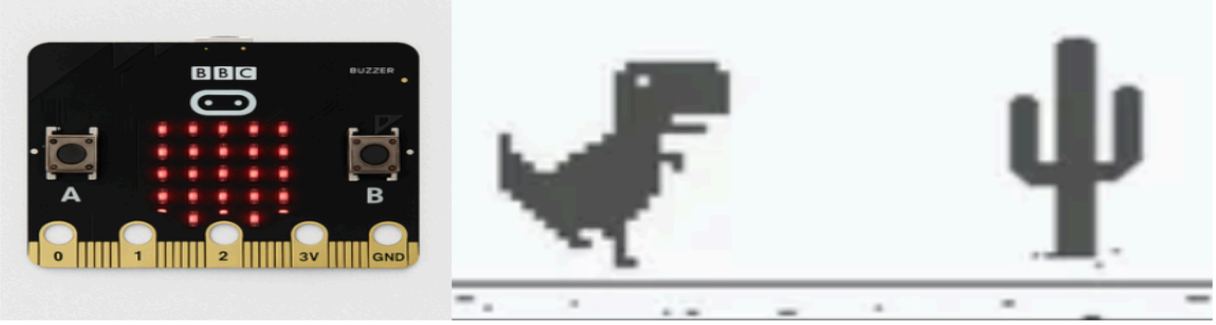
EGC 121 / Computer Architecture

Chrome Dino Game

Micro:Bit Project Report



Team Members:

- Ayush Patel (BT2024054)
- Kabir Ahuja (BT2024004)
- Kanav Kumar (BT2024021)
- Dayal Gupta (BT2024167)
- Parth Malhotra (BT2024197)
- Sachin Nain (BT2024201)
- Tanmay Dixit (BT2024016)

Introduction

This project recreates a simplified version of the classic Chrome Dino game on the BBC micro:bit v2. The game logic is written in MicroPython and uses the 5x5 LED matrix. The player controls a dinosaur that must jump over or duck under obstacles represented by LEDs moving horizontally from right to left. The game tracks the score based on survival time and ends when an obstacle overlaps with the dino's row. A "Game Over" message and the final score are scrolled at the end, with options to restart or quit.
The game is fully self-contained on the micro:bit, demonstrating how minimal hardware can be used to recreate classic gameplay mechanics in a constrained environment.

Objectives

● **Functional**: Develop a micro:bit-based version of the Chrome Dino game where the player can control a dinosaur using buttons to jump and duck, avoiding obstacles scrolling across a 5×5 LED grid.

● **Performance**: Ensure the game runs smoothly with a consistent frame rate and that button inputs (jump/duck) result in a visible change in the dinosaur's position within one frame cycle (~150 ms). The jump/duck action lasts for 1 second (1000 ms) before returning to normal.

● **Validation**: Confirm functionality by manually playing the game and verifying that player inputs (button A for jump, button B for duck) trigger correct in-game actions, and that the score increments accurately with each frame survived until a collision.

Micro:Bit Functionality

• **Buttons**:

– A pressed → transition to “up” state (jump) for JUMP_DURATION (500 ms).
– B pressed → transition to “down” state (duck) for the same duration.

• **LED Matrix**:

– Dino occupies INITIAL_ROWS ([1, 2]) in “normal,” UP_ROWS ([0, 1]) when jumping, and DOWN_ROWS ([2, 3]) when ducking.
– Cacti are single-column pixels spawned at the rightmost column on one of the INITIAL_ROWS lanes. They shift left by one column each frame.
– Collision is detected when a cactus’s (x,y) matches any of the dino’s current LED positions.

• **Timing**:

– Main loop advances every FRAME_DELAY (150 ms).
– Obstacle spawn countdown is randomized between SPAWN_MIN_FRAMES (10) and SPAWN_MAX_FRAMES (20).
– running_time() tracks state durations to revert to “normal.”

Video Recording

A **demonstration** of the working project has been recorded and uploaded to Google Drive. The video showcases the micro:bit detecting **various button presses and gestures**

Watch here:

https://drive.google.com/file/d/1vZhc-wmto0BQOWm81tPAqW0lz1AOGBVR/view?usp=drivesdk

Implementation Details

The game begins by setting configuration constants like jump duration, frame delay, and obstacle spawn intervals. The main loop handles input (Button A for jump, B for duck), manages state transitions based on timing, and updates the position of randomly spawned obstacles that move left each frame.
The dino is displayed on specific rows depending on its current state—normal, up, or down. Cacti spawn on random lanes and are drawn as two stacked pixels for visibility. Collision is checked each frame between cactus positions and the dino's LED coordinates. Score increases over time.
Upon collision, the screen is cleared, and "Game Over" with the score is shown. The player can restart with Button A or exit with Button B. The game ensures all rendering and logic occur within a consistent delay to maintain smooth gameplay.

Challenges & Solutions

- **Fitting the Game into a 5×5 Grid**

Challenge: Representing both the dinosaur and varying-height obstacles on only 5 rows.
Solution: Define three discrete row-sets (normal, up, down) and limit cacti to two normal lanes; this abstraction preserves gameplay while fitting in the grid.

- **Randomized Spawn Timing**

Challenge: Preventing predictable obstacle patterns that bore players.
Solution: Use Python's random.randint(SPAWN_MIN_FRAMES, SPAWN_MAX_FRAMES) to reset the spawn counter after each cactus,

ensuring unpredictable intervals.

- **Accurate Collision Detection**

Challenge: LED flicker or timing misalignment could miss or falsely detect collisions.

Solution: Update and draw all entities within the same loop iteration before checking overlaps, and only clear the display after collision logic runs, guaranteeing reliable detection.

Future Improvements

- **Leaderboard for Top Scorers**

– Store high scores in nonvolatile memory or via serial link, then display initials and scores at startup.

- **Dynamic Difficulty Scaling**

– Gradually decrease FRAME_DELAY or increase cactus speed/spawn rate as the player's score rises, keeping challenge fresh.

- **Gesture Controls**

– Use the micro:bit's motion sensor to detect a flick upward or downward for jump/duck, creating hands-free play.

Conclusion

This micro:bit implementation successfully captures the essence of the Chrome Dino game within severe hardware constraints. By mapping clear button inputs to discrete states, exploiting configurable timing constants, and managing LED drawing efficiently, the system offers responsive, engaging

gameplay. The modular design separating input handling, state management, spawn logic, and collision detection makes the code easy to extend. Overall, it demonstrates how simple microcontroller platforms can host interactive games with minimal resources.