uc3m | Universidad Carlos III de Madrid

DB in Telecomunication Technologies and Data Science

Programming 24/25

Group 196

Final Project

"Mario Bros."

Samuel Matamoros Alonso — 100583032

Darío Castro Vila — 100581710

Professor

Ángel García Olaya

Table of contents

1. Introduction

This project is a recreation of the LCD Mario Bros. from the Game & Watch series using Python and the Pyxel retro game engine.

The main goal was to implement the core mechanics of the original game (conveyor belts, box handling, scoring and failures) and extend them with custom features such as animations and a boss punishment system.

1.1. Game description

The game takes place in a factory where Mario and Luigi must move boxes along several conveyor belts to load them into a truck. The player controls each character's vertical position so they can catch and pass boxes in time, avoiding breaks; each delivered box increases the score, while each broken box increases the fail counter, and the game ends when three fails are reached.

2. Design and Architecture

The game logic is organized into several main Python classes: Main, Board, Character, Conveyor, Package, and Truck.

- The Main class initializes Pyxel, loads graphic resources, and delegates updating and drawing to the Board, which coordinates characters, packages, conveyors, truck, menu, difficulty and game state.
- The Board class manages global state (score, fails, difficulty, menu, game_over), the list of conveyors (Conveyor), the list of packages (Package), and the truck (Truck).
- Mario and Luigi are represented by the Character class, which stores their current level, whether they carry a package, whether they are resting or being reprimanded by the boss, and chooses the proper sprite using constants defined in config.

2.1. Core mechanics implementation

Character movement is implemented in Character.update by reading keyboard input, using different keys for Mario and Luigi, and limiting the allowed levels according to the number of conveyors in the current difficulty.

Conveyors have configurable speed, and each Package updates its position based on the speed of the conveyor at its current level, moving step by step until it reaches the end of each belt. When a box reaches the end of a conveyor, Board.package_update_all checks whether the correct character is at the correct level; if so, the box is moved to the next conveyor or to the truck and the score is increased. If no character is in position, the box is marked as broken, the fail counter increases, and the boss punishment state is triggered for the responsible character.

The Truck counts how many boxes it has loaded and, when a threshold is reached, switches to a “delivery” state for a fixed number of frames, during which the characters rest and the normal game logic is paused; after the delivery ends, the truck is emptied and normal gameplay resumes.

2.2. Boss punishment system

A boss system was added inspired by classic LCD reprimand animations: when a box breaks, a boss_active state is set in the Board. Depending on which character failed, boss_target is set to Mario or Luigi, and that character’s reprimand flag is enabled, forcing their sprite to a fixed position next to the boss (Luigi on the left, Mario on the right) using MARIO_REPRIMAND and LUIGI_REPRIMAND sprites from config.

While boss_active is true, a timer (boss_timer) is decreased each frame in Board.update, and the normal game update (character movement and package logic) is skipped to simulate the reprimand pause. When the timer reaches zero, boss_active is set to false, both characters reprimand flags are cleared, and the normal update logic resumes so the game can continue.

2.3. Difficulty and package generation

The game includes several difficulty levels that adjust the number of conveyors, their speeds, and the scoring and spawning parameters through methods such as difficulty0, difficulty1, etc., inside Board. Package generation is handled in Board.package_gen, which creates a new Package when there are no boxes on screen and certain score or timing conditions are met, allowing the pace of the game to scale with difficulty.

2.4. Issues and solutions

Several typical state-management issues appeared during development, such as the game “freezing” when the boss or truck entered a special state and the main loop stopped updating the logic that should exit that state. This was solved by centralizing state control inside Board.update, avoiding extra conditions in Main.update, and by using clear boolean flags (boss_active, delivering, resting, reprimand) plus timers to ensure that every special state always has a defined exit.

Another frequent issue was repeated boss activation when multiple boxes failed in a row; this was fixed by wrapping boss activation in if not self.boss_active so a new reprimand cannot start until the previous one has finished. Finally, the package spawn logic was adjusted to avoid ending up with no boxes after a series of fails, temporarily simplifying the spawn condition while debugging the boss and truck systems.

Bibliography