Learning Levels of Mario AI Using Genetic Algorithms

Team Members:
Ferran Mirabent, Carlos Bellanco, Roger Pieres,
Martí Lorente, Marc Martínez
Master's Course in Optimization

December 14, 2025

1 Motivation

We selected the paper *"Learning Levels of Mario AI Using Genetic Algorithms"* by Baldominos et al. to apply the concepts learned in the Optimization course to a video game environment. The primary motivation lies in treating the gameplay not as a reactive agent problem—where the agent must decide actions based on real-time visual processing—but as a pure optimization problem. By defining the level completion as finding the optimal timed sequence of keystrokes, we can focus strictly on the mechanics of the Genetic Algorithm (GA) and the objective function without the computational overhead of state-space dimensionality or vision processing.

2 Introduction to the Optimization Problem

The optimization problem addresses the *Learning Track* of the Mario AI Championship. The objective is to maximize the score obtained in a specific fixed level of *Infinite Mario Bros.* Unlike standard gameplay where an agent reacts to the environment, this approach seeks a pre-defined sequence of actions that maximizes the fitness function S (Score).

The constraints and conditions of the problem are:

- **Search Space:** The agent must determine a sequence of inputs (actions) for every time tick of the game.
- **Computational Limit:** The agent is allowed a maximum of $N = 10,000$ games (evaluations) to learn the level.
- **Input Constraints:** The controller simulates a D-Pad and two buttons (A and B). While there are $2^6 = 64$ combinations, domain knowledge reduces this to 22 feasible distinct actions (e.g., pressing Left and Right simultaneously is impossible).

The objective function to maximize is the **Score (S)**, defined by the Championship rules:

$$S = D + 64d_f + 58d_m + 58d_{gm} + 42k + 12k_{st} + 17k_{sh} + 4k_f + 1024s + 32m + 24b_h + 16c + 8t' \quad (1)$$

Where:

- D: Physical distance traveled.
- s: Final status (1 if level completed, 0 otherwise).
- t': Time remaining.
- k, k_{st}, k_{sh}, k_f: Enemies killed (total and by specific method).

- d_f, d_m, d_{gm}: Items collected (flowers, mushrooms, green mushrooms).
- c, b_h: Coins collected and hidden blocks found.
- m: Mario's final mode (Small, Big, Fire).

3 Explanation of the Used Algorithms

The authors propose a Genetic Algorithm (GA) to evolve the sequence of actions. The approach is divided into two stages: a naive stage using domain-independent operators and a refined stage using domain-specific knowledge.

3.1 Encoding

The chromosome represents the sequence of actions Mario performs throughout the level.

- **Genes:** Each gene is an integer representing a specific combination of buttons pressed. In the first stage, the search space is reduced to 11 actions (assuming the Run button is always pressed). In the second stage, all 22 feasible actions are used.
- **Chromosome Length:** The length is fixed at 3,000 genes. This derives from the maximum level time (200 seconds) discretized into 15 ticks per second ($200 \times 15 = 3000$).

3.2 Genetic Operators

The algorithm uses Tournament Selection, Crossover, and Mutation, tailored differently in the two experimental stages.

3.2.1 Initialization

- **Naive:** Actions are chosen randomly.
- **Hybrid (Domain-Dependent):** Recognizing that moving right is the primary goal, a "guided" initialization favors *Right* and *Right+Jump* actions. The hybrid approach randomly selects between naive and guided initialization for each gene.

3.2.2 Selection

Tournament selection is used, where a subset of individuals (T_s) compete, and the one with the highest fitness becomes a parent. The domain-dependent stage adds elitism to preserve the best solutions.

3.2.3 Crossover

- **Single-Point Crossover:** A random point n ($n < 3000$) is selected. Offspring are created by splicing the parents' action sequences at n.
- **Domain-Aware Crossover:** The cut point n is not purely random. The algorithm searches for a point where Mario's physical position (x, y) is similar in both parent simulations (Euclidean distance $< \Delta$). This ensures continuity; if one parent plays well up to point A and the other plays well from point A onwards, splicing them at A (where Mario is at the same location) creates a viable trajectory.

3.2.4 Mutation

- **Random Mutation:** Modifies M genes randomly across the chromosome.
- **Windowed Mutation (Domain-Dependent):** Mutations are targeted specifically at the *end* of the effective action sequence (just before Mario dies or the level ends). This exploits the fact that early actions are likely "correct" if Mario reached that far. The mutation window size (W) doubles dynamically if fitness does not improve, balancing exploration and exploitation.

4 Proposal of Alternative Algorithms

While the Genetic Algorithm effectively optimizes a fixed sequence, other approaches could solve the sequence generation problem or the general gameplay agent problem.

4.1 A* Search Algorithm

A* can be used to search the state space of the game simulator. By looking ahead into the future of the physics engine, the agent builds a tree of possible next states (sequences of actions).

- **Comparison:** Unlike the GA, which evolves a full schedule blind to the immediate state, A* makes locally optimal decisions that aggregate to a solution. A* is generally more computationally expensive per step (requiring forward modeling) but can adapt to dynamic changes if the level were not fixed. In the context of the Mario AI benchmark, A* has historically outperformed other methods in the Gameplay track.

4.2 Deep Learning (Deep Q-Networks)

Instead of optimizing a predefined list of discrete actions, a Deep Learning approach (specifically Deep Reinforcement Learning) could train a neural network to output actions based on the screen's pixel data.

- **Comparison:** This method aligns with the "DeepMind Atari" approach. While the GA in the referenced paper ignores the game state (blind replay), a DQN agent learns a policy $\pi(state) \rightarrow action$. This would result in a more robust agent capable of playing *any* level, not just the specific seed it was trained on. However, training a DQN requires significantly more compute resources and samples than the 10,000-game limit imposed by the specific Learning Track rules addressed in the paper.

5 Reproduction of Methodology

This section will be completed by the team. It will include the implementation of the headless simulation mode, the chromosome encoding, and the reproduction of the domain-dependent operators described in Section 3.

6 Conclusions

The referenced paper demonstrates that level completion in platformers can be successfully modeled as an optimization problem of action sequences. The results highlight the importance of **domain knowledge** in optimization heuristics:

1. **Granularity Matters:** Using a granularity of $g = 5$ (holding buttons for 5 ticks) significantly outperformed $g = 1$, effectively reducing the search space and smoothing the agent's behavior.

2. **Guided Operators:** The domain-aware crossover (matching physical positions) and mutation (modifying the end-of-life sequence) were crucial. The naive GA often failed to converge on complex levels, whereas the tailored GA achieved scores surpassing the 2010 competition winner (avg. 12,059 vs 9,003).

The study concludes that while "blind" optimization is possible, incorporating the semantic meaning of the optimization variables (e.g., "spatial continuity" in crossover) is essential for high-dimensional discrete problems like this one.

References

[1] A. Baldominos, Y. Saez, G. Recio, and J. Calle, "Learning Levels of Mario AI Using Genetic Algorithms," in *Conference of the Spanish Association for Artificial Intelligence*, pp. 267-277, Springer International Publishing, 2015.