Interpretable Control Policies in Atari Pong

Christina Berghegger
Université Toulouse Capitole - IRIT
Toulouse, France

Camilo De La Torre
Université Toulouse Capitole - IRIT
Toulouse, France

Sylvain Cussat-Blanc
Université Toulouse Capitole - IRIT
Institut Universitaire de France
Toulouse, France

David Simoncini
Université Toulouse Capitole - IRIT
Toulouse, France

Yuri Lavinas
Université Toulouse Capitole - IRIT
Toulouse, France

Abstract

We apply Multimodal Adaptive Graph Evolution (MAGE) to evolve interpretable visual control policies for Atari Pong. MAGE generates graph-based programs that process game frames to predict actions, ensuring generalization and human interpretability. By training against diverse opponents, we obtained a generic policy. We also interpret the generated graph into an executable code and a natural language strategy. The resulting compact policies achieve perfect performance while remaining full interpretable.

Keywords

Cartesian Genetic Programming, Control Policies, Interpretability

ACM Reference Format:
Christina Berghegger, Camilo De La Torre, Sylvain Cussat-Blanc, David Simoncini, and Yuri Lavinas. 2025. Interpretable Control Policies in Atari Pong. In *Proceedings of Genetic and Evolutionary Computation Conference (GECCO '25).* ACM, New York, NY, USA, 3 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Learning control policies directly from visual input is a central challenge in reinforcement learning and evolutionary computation. While deep learning approaches can achieve high performance, they often result in black-box models that lack interpretability.

Here, we focus on generating interpretable visual control policies for the Atari *Pong-v4* game using Multimodal Adaptive Graph Evolution (MAGE) [3], an extension of Cartesian Genetic Programming (CGP) [2] with type awareness. MAGE evolves graph-based programs that process game frames to predict actions, enabling structured and type-aware policy representations. We show that MAGE can produce compact, generalizable policies while preserving interpretability. By training against diverse opponents, we improve robustness across game seeds. Finally, we propose a multi-level interpretability framework, translating evolved graphs into code and natural language, making the policies understandable even to non-experts.

GECCO '25, Málaga, Spain

ACM ISBN 978-1-4503-XXXX-X/2025/07
https://doi.org/XXXXXXX.XXXXXXX

2 Methods - Policy generation

MAGE's ability to handle multiple data types is crucial in our application, given that we need to predict game actions from visual input in terms of game screens. MAGE generates solutions as a directed graph, each node representing a function from a pre-defined function library. The graph takes as input the four previous game screens and constants, and generates three predictions, one for each possible game action the player can take next.

We performed three independent runs with a population size of 12, an offspring size of 3, and a tournament size of 2. The number of generations was set to 2,500. To accelerate optimization, the frame limit per episode was progressively increased: up to generation 500, we set the limit to 600 frames; from generation 501 to 1,500, we raise it to 2,000 frames; and from generation 1,501 onward, we set it to 18,000 frames. Although we determined these parameters empirically, a more extensive parameter study and a larger number of independent runs would be required to identify the most suitable configuration for this task.

A key factor enabling policy generalization across all seeds was a thorough analysis of the Pong game environment. We observed that exposing a policy to opponents (i.e. seeds) exhibiting diverse playing behaviors enhances both generalization and performance. Based on this analysis, we identified four distinct groups of opponents. Each individual in the population was exposed to one seed from each group, which remained fixed throughout the evolutionary process. The total number of frames used during optimization can therefore be computed by summing the frame limits across generations and multiplying by 4 (number of seeds) and 12 (population size). The best-performing policy presented in this study was obtained from a run with a population size of 12 and after 1,500 iterations.

3 Policy Performance

We tested the best derived policy five times in the Atari game environment *Pong-v4*[1], each run covering 100 different random seeds in the range of 1 to 10.000. The policy shows a high degree of generalizability and performance: it wins against all opponents with the maximum score of 21 after on average 5811 game actions. The opponent, on the other hand, never scores. From a thorough analysis of the game, we can see that the player applies a repetitive behaviour, after which the opponent is not able to defend the ball, until reaching 21 points and winning the game. The strategy is explained in Section 4 and visualised in the project's GitHub repository[2].

[1] https://ale.farama.org/environments/pong/
[2] https://github.com/chrissi-b/gecco-pong-competition/

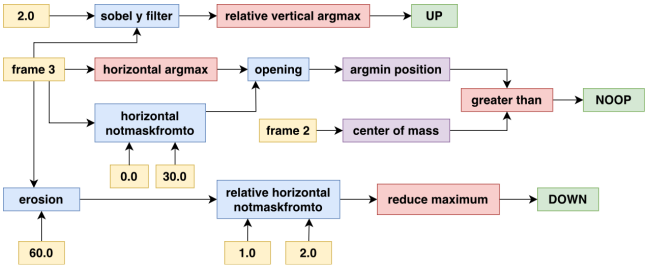

Figure 1: Simplified graph of the best policy colored by output type. Blue functions return images, red functions return floats and purple functions return image coordinates.

```
function evolved_pong_policy(frame1, frame2, frame3, frame4)

    ### Program NOOP
    # position of the ball
    ball_x_pos = experimental_horizontal_argmax(frame3)
    # hides the lower part of the screen
    img_hide_lower = notmaskfromtoh_image2D(frame3, 0.0, 30.0)
    # screen devided in white upper part and black lower part
    img_splitted = experimental_opening_2D(img_hide_lower, ball_x_pos)
    # mean index of pixel weighted by intensity (location of players and ball)
    location_bright_pixels = center_of_mass(frame2)
    # start of black lower part
    start_lower_part = argmin_position(img_splitted)
    # true if mean pixel is located in lower part, wrong if not
    bright_pixels_lower = true_gt(location_bright_pixels, start_lower_part)
    output_noop = identity_float(bright_pixels_lower)

    ### Program UP
    # only keeps ball pixels, players are blacked out
    screen_ball = sobely_image2D(frame3, 2.0)
    # relative y position of the ball (the further up, the lower)
    rel_ball_pos = experimental_vertical_relative_argmax(screen_ball)
    output_up = identity_float(rel_ball_pos)

    ### Program DOWN
    # turns entire screen black
    black_screen = erosion_2D(frame3, 60.0)
    # adds white rows in lower part of image
    white_rows = notmaskfromtoh_relative_image2D(black_screen, 1.0, 2.0)
    # gets minimum pixel value (white rows => always 0.34)
    maximum_pixel = reduce_maximum(white_rows)
    output_down = identity_float(maximum_pixel)

    outputs = (output_noop, output_up, output_down)
    return ACTIONS[argmax(outputs)]
end
```

Figure 2: Code of best policy with renamed functions and explanations of each step in the pipeline. Input frames correspond to the preceding game states t-4 to t-1.

4 Interpretability Analysis

To evaluate the policy in terms of interpretability, we propose three different levels of abstraction: First, we provide the generated graph from MAGE. Then, we transform the graph into code and translate function names into more intuitive and human-understandable actions. Finally, we propose a game strategy in natural language derived from the code, accessible to people without computer vision and programming expertise.

The policy has eleven active nodes, each representing a function (Fig. 1). To show that our graph is interpretable, we wrote the graph-generated program in Julia syntax, which we understand is a human-understandable approach (Fig. 2). Some function outputs, such as the output of *greather_than* and the evaluations of constants, were simplified as they remain the same throughout the evolutionary process, reducing the size of the code. To derive explanations from the graph policy, we analyzed different game states and evaluated the impact of function output on action predictions. The small graph size allows for this processing to be done rapidly. Moreover, some initial high-level analysis, showing stable prediction patterns, simplified the explanation retrieval process and helped defining the applied game strategy:

- ***NOOP* subprogram**: the *center_of_mass* function returns only two distinct values throughout the game, deciding the final prediction: 1 if the mean position of players and ball is in the lower screen part, 0 if in the upper part.
- ***DOWN* subprogram**: the prediction is fixed in all stages of the game, returning always a white-pixel intensity.
- ***UP* subprogram**: returns the relative position of the ball. It is the only prediction that changes over generations. Thus, most of the analysis was conducted to understand how the output of this subprogram relates to the predictions of both *NOOP* and *DOWN*.

These high-level analyses greatly reduced the time of analyzing the policy and demonstrate the advantages of decomposability, which we understand as one of the cornerstones of interpretability in control policies. The inherent decomposability of the graph, both on program and function level, allows us to break down the entire policy into meaningful components which are easier to interpret and understand [1]. Furthermore, the fact that we can translate the raw graph into a short strategy in natural language (Fig. 3) that is understandable to a wide range of users, without requiring any additional help from computer models, indicates a high level of simulatability, which is another key element of interpretability and refers to the ability to reproduce the decision process in a reasonable amount of time by a user [1].

> Our player goes **DOWN** searching the lowest position in the screen. It then starts to go **UP**, and **STOPS** one the ball, coming from the opponent's direction, approaches the lower wall and is about to bounce off. The player intercepts and the ball goes back towards the opponent.

Figure 3: Game strategy in natural language

5 Conclusion

This work demonstrates that interpretable and generalizable visual control policies can be evolved using a graph-based, type-aware approach. Using MAGE, we obtained compact policies for Atari Pong that achieve perfect in-game performance and can be translated into human-understandable strategies through a multi-level interpretability framework.

While our results are promising, the study remains preliminary. The current evaluation is based on a small number of independent runs and lacks comparisons with baseline approaches such as standard CGP or deep learning models. Future work will include a more extensive empirical study to assess robustness and compare against alternative methods. Nonetheless, these early findings highlight the potential of evolutionary programming to generate interpretable and effective control policies from visual input.

References

[1] Zachary C. Lipton. 2018. The Mythos of Model Interpretability: In machine learning, the concept of interpretability is both important and slippery. *ACM Queue* 16, 3 (2018), 31–57. doi:10.1145/3236386.3241340

[2] Julian F. Miller and Peter Thomson. 2000. Cartesian Genetic Programming. In *Genetic Programming (Lecture Notes in Computer Science)*, Riccardo Poli, Wolfgang Banzhaf, William B. Langdon, Julian Miller, Peter Nordin, and Terence C. Fogarty (Eds.). Springer, Berlin, Heidelberg.

[3] Camilo De La Torre, Yuri Lavinas, Kevin Cortacero, Hervé Luga, Dennis G Wilson, and Sylvain Cussat-Blanc. 2024. Multimodal Adaptive Graph Evolution for Program Synthesis. In *International Conference on Parallel Problem Solving from Nature*. Springer, 306–321.