Evolution of Inherently Interpretable Visual Control Policies

Camilo de la Torre, Giorgia Nadizar, Yuri Lavinas, Hervé Luga, Dennis G Wilson, Sylvain Cussat-Blanc

Evolution of Inherently Interpretable Visual Control Policies

Camilo De La Torre
University Toulouse Capitole, IRIT - CNRS UMR5505
Toulouse, France
camilo.de-la-torre-villacis@ut-capitole.fr

Giorgia Nadizar
DIA, University of Trieste
Trieste, Italy
giorgia.nadizar@dia.units.it

Yuri Lavinas
University Toulouse Capitole, IRIT - CNRS UMR5505
Toulouse, France
yuri.lavinas@ut-capitole.fr

Hervé Luga
University Toulouse Capitole, IRIT - CNRS UMR5505
Toulouse, France
herve.luga@univ-toulouse.fr

Dennis G. Wilson
ISAE-Supaero, University of Toulouse
Toulouse, France
dennis.wilson@isae.fr

Sylvain Cussat-Blanc
University Toulouse Capitole, IRIT - CNRS UMR5505, Institut Universitaire de France
Toulouse, France
sylvain.cussat-blanc@irit.fr

ABSTRACT

Vision-based decision-making tasks encompass a wide range of applications, including safety-critical domains where trustworthiness is as key as performance. These tasks are often addressed using Deep Reinforcement Learning (DRL) techniques, based on Artificial Neural Networks (ANNs), to automate sequential decision making. However, the "black-box" nature of ANNs limits their applicability in these settings, where transparency and accountability are essential. To address this, various explanation methods have been proposed; however, they often fall short in fully elucidating the decision-making pipeline of ANNs, a critical aspect for ensuring reliability in safety-critical applications. To bridge this gap, we propose an approach based on Graph-based Genetic Programming (GGP) to generate transparent policies for vision-based control tasks. Our evolved policies are constrained in size and composed of simple and well-understood operational modules, enabling inherent interpretability. We evaluate our method on three Atari games, comparing explanations derived from common explainability techniques to those derived from interpreting the agent's true computational graph. We demonstrate that interpretable policies offer a more complete view of the decision process than explainability methods, enabling a full comprehension of competitive game-playing policies.

1 INTRODUCTION

The applications of Artificial Intelligence (AI) and, more specifically, Reinforcement Learning (RL) is driving innovation in fields such as autonomous drones [27] and plays a crucial role in various robotics [57] and automatic control applications [52], enhancing the capability of controllers to learn and adapt to complex scenarios. RL techniques are also being exploited in video games, enabling breakthroughs in game-play optimization thanks to AI-driven strategies [7, 36, 44, 53]. RL is being explored for applications in critical systems like self-driving cars [21], automated airplane operations [35], or robotic surgery [13] .

However, there is a significant issue in the application of an automatic decision system, such as a control policy trained through RL, to critical systems. The control system must be trusted and verified for possible failure cases [1]. Recent work focuses on making RL-based systems more robust [18] to failure cases, or on using post-hoc analysis to explain the decision pipeline of artificial agents [26]. The goal of understanding a given decision pipeline is rendered more difficult by the use of deep neural networks, which are highly parameterized and black-box, as the key decision model [50].

Vision-based decision-making tasks, such as those involved in self-driving cars [33] or robots interacting with their environments through cameras [15], further complicate the understanding of an agent's decision pipeline. These tasks, by nature, treat high-dimensional information in order to make sequential decisions over time; some of the information will be used for a control policy at certain moments, and different information will be used at others. A variety of explainability methods have been developed to address this challenge, using post-hoc statistical analysis to determine which part of the visual information is being used for a given decision [64]. However, there is growing recognition that these techniques often serve more as exploratory tools than as definitive explanations [3]. This is because the explanations inferred with such methods—from visualization, in particular—are strongly influenced by subjective cognitive biases and rarely offer falsifiable insights, limiting their utility for rigorous explanation and validation [3]. Furthermore, Explainable Artificial Intelligence (XAI) methods can be tricked or attacked, failing to provide genuine insights into what is happening within the model [6].

This has led to the encouragement for the use of *inherent interpretability* [50], where the model itself is designed to be understandable rather than relying on post-hoc explanations. For our purposes, we define interpretability as "the ability of a user to deduce why a model is making certain decisions, given sufficient time to analyze it". This primarily hinges on two principles: *simulatability*, where the entire model can be mentally simulated by a human, and *decomposability*, where individual components of the model correspond to specific, understandable functions [34].

To this end, we propose an approach based on Graph-based Genetic Programming (GGP) to evolve an end-to-end interpretable controller for vision-based decision-making tasks. Specifically, we use Multimodal Adaptive Graph Evolution (MAGE) [12], a Cartesian Genetic Programming (CGP) [42] variant designed to handle diverse data types while ensuring type correctness during the evolution process. MAGE optimizes a graph composed of simple building blocks: the optimization determines which blocks to use and how to connect them effectively. By design, this approach provides transparency and interpretability as the optimization encourages decomposability—due to straightforward building blocks—and simulatability—enabled by compact graph size. Moreover, it also enables users to understand, verify, and debug the complete control policy, taking corrective actions in cases of unwanted decisions.

We demonstrate the advantage of a transparent controller on three Atari games: Pong, Freeway, and Bowling. The interpretable controllers evolved using MAGE achieve performance comparable to state-of-the-art methods. More importantly, though, is that the interpretable nature of these controllers allows for rigorous inspection of the decision-making pipelines, allowing us to identify the key factors driving decisions and to infer how the system might behave in uncontrolled scenarios—for instance, predicting triggers for unwanted behavior. For comparison, we analyze the evolved controllers using common XAI tools, specifically saliency visualizations, which show which visual information is being used per time frame. While these tools can complement visual inspections as exploratory aids, they are not strictly necessary due to the inherent interpretability of our method. We show that the explanations derived from interpretation of the control policy graphs are more precise, detailed, and understandable than the explanations derived from the saliency visualizations.

All in all, the contributions of this paper are threefold:

- we use MAGE to generate policies that are of comparable performance to state-of-the-art methods on three Atari games,
- these policies are analyzed based on the generated code to interpret the decision-making mechanisms,
- interpretations are compared to two state-of-the-art post-hoc explainability approaches to evaluate the added benefits of our full-interpretable approach.

Our work serves as a starting point for further research into interpretable RL methods, while also raising awareness about the critical importance of interpretability in decision-making systems. Crucially, we demonstrate that achieving interpretability does not always require sacrificing performance.

2 RELATED WORK

2.1 Evolutionary Computation for Interpretable Reinforcement Learning

Evolutionary Computation (EC), and more specifically Genetic Programming (GP), have been recognized as well-established tools for obtaining inherently interpretable Machine Learning (ML) models [41, 47, 65]. Within interpretable RL [20], GP has been widely used for both simple discrete control tasks, such as Cart-Pole and Mountain Car [8, 24], and more complex continuous control tasks, where it has achieved notable results in various robotics applications [30, 39, 40, 46, 60, 62]. Interestingly, several studies have leveraged Quality-Diversity (QD) to address premature convergence in both discrete and continuous control tasks [19, 45].

GP has also been applied to address vision-based control tasks, namely Atari games. Tangled Program Graphs, which involve selecting actions according to the highest bid from competing programs, have shown impressive results on Atari games, competitive with deep RL [28, 29, 31]. These methods employ discretized pixels as inputs, rendering the full understanding of the visual analysis needed for control less clear. However, the interpretation of which program in the Tangled Program Graph is being used has led to interesting analysis; for a single multi-task agent trained to play different games, the subprograms used for individual games or shared between games could be identified [29]. Another recent work [9] proposed to divide the task into two components: visual processing via a kernel and decision-making through a decision tree. Their approach focused on the deterministic version of the Atari environment, which has been identified to be an easier task [37].

Most similar to our work is the application of Mixed-Type CGP (MT-CGP), another CGP variant [23], to Atari [61]. This work demonstrated that simple but effective policies can arise when using MT-CGP, some of which employed static strategies that ignored the visual input. While those simple policies were interpretable, policies that relied on visual analysis, such as in the Boxing game, were determined too complex for exhaustive interpretation. In contrast, our work uses a relatively small set of impactful image analysis functions and ensures type-correctness of the pipelines through MAGE, enhancing the decomposability and eventually interpretability of the overall models.

2.2 Explaining Visual Decision-Making

Explanations of the decisions of an artificial agent often rely on a secondary model or statistical analysis of the agent's behavior [34, 43]. The explanations arising from these methods aim to provide either global or local interpretability [14].

Global interpretability focuses on understanding the model without reference to specific inputs or decisions. For example, Activation Maximization (AM) methods visualize the features that maximize the activations in Artificial Neural Networks (ANNs). These techniques have been applied in domains like image classification Convolutional Neural Networks (CNNs) [55]. Their application to Atari games has been limited [51], as the decision process of a game-playing agent often relies on a complex mix of information. Generative modeling has been used to find counter-example inputs for a given model, demonstrating potential weaknesses or failure states of the agent outside of the standard game loop [51].

Local interpretability aims to explain decisions for specific input data points; this form of explanation has been widely used for Atari [26]. Methods such as RISE [48] and Occlusion Sensitivity [2, 63] function by randomly perturbing or occluding a part of the visual input and then determining the change in action resulting from the input perturbation. By doing so over a full image, a relation between the important input information and the action taken at each timestep can be drawn; this relation is often represented as a saliency map over the inputs. AM has been similarly applied to Atari games to visualize the important input information [26, 55]. [22] provided examples of such visualizations for Atari games. Other methods, such as the work of [32], map input images to textual strategy explanations, a concept increasingly relevant with advancements in Large Language Models (LLMs) [38].

While post-hoc analysis, such as saliency visualization, can give an idea of what information is being used for each decision in a sequence, it does not give a full explanation of the policy's behavior [3]. Specifically, it is difficult to construct, based on post-hoc statistical analysis alone, falsifiable hypotheses; if a claim made in explaining a policy is false, then the falsehood should be identifiable from an experiment or observation. This is not the always the case for explanations given by saliency maps or LLMs, as they may not represent the entirety of the information. This limitation of XAI for visual analysis is known [50, 64], however, there are few alternatives. One such alternative is to create inherently interpretable surrogate models [49], which has been used for Atari games [54], creating fully explainable models that imitate the behavior of a black-box one. In this work, we argue rather for the direct optimization of interpretable visual control policies.

3 MULTIMODAL ADAPTIVE GRAPH EVOLUTION FOR VISION-BASED CONTROL

Our approach focuses on vision-based RL with the primary goal of synthesizing policies that are both high-performing and inherently interpretable. Specifically, we target tasks where the input consists of pixel-based observations from the environment (e.g., the screen of a game), and the output corresponds to an action selected from a predefined discrete set (e.g., the button that would be pressed during gameplay).

In this scenario, a policy π is a function that maps the pixel-based input to an action. The optimization of π is driven by the cumulative reward collected during an episode, corresponding to the score achieved in a simulated game. This cumulative reward serves as the fitness function f to be maximized during policy evolution.

In our approach, the policy π is represented as a graph optimized with MAGE. MAGE is designed to handle multiple data types while ensuring type correctness throughout the optimization process [11]. In other words, by being type-aware, MAGE ensures that each mutation is "type-safe" enabling a more efficient search, by only allowing for the existence of valid programs.

As with standard CGP, MAGE represents graphs as a sequence of nodes positioned in a Cartesian plane, where each node represents a specific function and its connections to other nodes. However, MAGE introduces a key difference: it groups nodes based on their output data types. For this application to vision-based RL, we use three categories of nodes for three types: images, scalars and tuples of integers with associated libraries composed of image-to-image, image-to-scalar, scalar-to-scalar or image-to-coordinates functions, e.g., image dilation (image-to-image), image mean (image-to-scalar), logarithm (scalar-to-scalar), image argmax (image-to-coordinates). Connections between nodes are allowed only when type constraints, defined by the function's signature, are satisfied, ensuring graph correctness.

In MAGE, each node is encoded by $1 + 2 \cdot n_{\text{arity}}$ integers, where arity is the highest functions' arity encountered in the libraries. The first specifies the function the node performs from a predefined typed function library. The remaining $2 \cdot n_{\text{arity}}$ integers indicate the nodes and their corresponding type, from which the function takes its inputs. These are selected ensuring that (1) the input's type is in accordance with the function's signature (2) the input node precedes the caller node. Outputs are derived from the last n_{out} scalar nodes of the graph, as in [45]. For our tasks, we define n_{out} as the reduced number of available actions chosen for each game environment, a detailed description for each game is available at Table 1. Since each output node is mapped to an allowed action in the environment, the action to be performed is selected by applying a hardmax function to the value of these outputs. Thus, the genome of a MAGE individual is composed of $n_{\text{types}} \cdot (1 + 2 \cdot n_{\text{arity}}) \cdot n_{\text{nodes}}$; for vision-based RL $n_{\text{types}} = 3$, as we work with images, scalars and specific coordinates (i.e., tuples of integers) of the image.

As demonstrated in [11], the mutation in MAGE is *type-safe*, i.e., MAGE ensures that any mutation maintains type correctness by respecting the constraints on input and output types, and is *active-only*, i.e., MAGE restricts mutations to the active parts of the graph—those nodes and connections that contribute to the final outputs—avoiding changes to inactive parts of the graph.

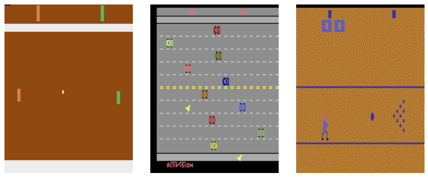
Evolution of the graph is performed with a mutation-only Genetic Algorithm (GA), rather than the canonical $1 + \lambda$ evolutionary algorithm often used for CGP. This choice is motivated by recent work in GGP which uses a GAs in solving control tasks [46] and performing function synthesis [12].

4 EXPERIMENTAL EVALUATION

Through experimental evaluation, we aim to demonstrate that MAGE can generate policies that effectively act in vision-based control tasks while remaining interpretable by a human. To this end, we experiment with three distinct vision-based tasks, namely three Atari games, and optimize policy graphs to solve them. These policies are compared to other approaches to assess their performance. While our goal is not to outperform non-interpretable approaches, MAGE must still generate policies of high quality compared to deep RL and human performance on Atari. The best policy for each game is then manually analyzed to fully interpret the discovered strategies, enabling an assessment of their strengths and weaknesses. Our manual interpretation of the policies is compared to two post-hoc visual inspection techniques—namely, visual importance maps and occlusion maps—highlighting the limitations of both methods when compared to comprehensive interpretation of the generated graphs.

4.1 Atari Games

We consider three Atari games—*Pong*, *Freeway*, and *Bowling*—as vision-based control tasks [7, 36]. For all tasks, we use the v4 variant provided by the Gymnasium library [59]. Atari games have built-in stochasticity to emulate the original console for comparison with human scores. This random noise makes new instances of the game different from each other, and there are multiple ways of implementing this noise. We use the "*G*NoFrameskip-v4" variant for all games, where *G* is either "Pong", "Freeway" or "Bowling". This variant ensures that the *repeat action probability* is set to 0.. Then, as is commonly done, we manually set the *frameskip* (e.g., action repeat) parameter to 4 [5] and we stack the previous 4 frames at



(a) Pong **(b) Freeway** **(c) Bowling**

Figure 1: The Atari games employed as vision-based control tasks.

each time step [16]. This allows us to have random "seeds" per game, enabling a study of robustness to different random conditions.

For each game, the visual *input*, i.e., the screen, was converted to grayscale and resized to 84×84 pixels [16]. We pre-process these images to focus on the relevant game field by cropping out the score area and other irrelevant elements. The exact pixels left out per game are described in Table 1. After cropping, the inputs are of size 63×84, 63×78, 58×72 pixels for *Pong*, *Freeway* and *Bowling*, respectively.

Table 1: Details of Games, Action Sets, and Screen Crops.

Game	Action Set	n_{out}	Screen Crop
Pong	$\{0, 4, 5\}$	3	$[15:77, 1:84]$
Freeway	$\{0, 1, 2\}$	3	$[11:74, 6:84]$
Bowling	$\{0, 1, 2, 3\}$	4	$[14:72, 6:78]$

The *action space* in Atari games includes up to 18 possible actions (no-op, fire, movements in 8 directions, and their combinations with fire). However, the actions actually available to use depend on the game. For Pong, these are: `NOOP`, `FIRE`, `LEFT`, `RIGHT`, `LEFTFIRE`, and `RIGHTFIRE`, but we limit them to: `NOOP`, `RIGHTFIRE`, and `LEFTFIRE`. For *Freeway*, the available actions are: `NOOP`, `UP`, and `DOWN` and we used all three of them. For *Bowling*, these consist of `NOOP`, `FIRE`, `UP`, `DOWN`, `UPFIRE` and `DOWNFIRE` but we limit them to the first 4 (i.e., `NOOP`, `FIRE`,`UP` and `DOWN`).

Concerning the *rewards*, they are calculated per frame and are game-dependent. In Pong, rewards are based on scoring against the opponent by passing the ball beyond their paddle or losing points if the ball passes one's own paddle. In Freeway, rewards are earned by successfully moving the chicken across the freeway. In Bowling, rewards are granted by making pins fall after throwing the ball. As per common standard [44], we simulate real Atari gameplay dynamics using a *frame-skip* of 4 and an action repeat probability of 0.0.

4.2 Optimization

For the application of MAGE to the three chosen games Atari, we focused on maximizing the performance on each game, rather than performing an exhaustive parameter study across the Atari benchmark. We explored two distinct training schedules, each designed to explore the trade-off between computational efficiency and evolutionary progress. The first schedule used a fixed duration of $36,000$ frames for all episodes. The second schedule aimed to accelerate the early phases of evolution by progressively increasing the episode duration based on the evolution's progress. Specifically, the duration started at 600 frames for the first 500 generations, increased to $2,000$ frames for the subsequent $1,000$ generations, and then used $36,000$ frames for the final phase. This adaptive scheduling approach helps minimize computational overhead during the early evolution stages, when policies typically exhibit poor performance, which is especially important given the computationally demanding nature of these experiments.

The policies were evolved using three fixed seeds, which remained constant throughout the entire evolutionary process. To effectively carry one elite population from one generation to the next one, the games were restarted with their corresponding seeds. The fitness function f was defined as the average cumulative reward of the policy π over three independent instances of the environment. To encourage more effective policies, we introduced a bonus for the use of image-to-scalar functions, which was calculated as the average entropy of these function's outputs over all episode's time steps. For the second training schedule, this bonus was capped at a maximum value of one.

We used a population size of 36, as this was the number of CPUs available in a single computational node used for the experimental evaluation, allowing for parallel evaluation. After manual hyperparameter optimization, we chose a number of offspring of 8, and tournament selection with a size 3. We evolve the population for 2500 generations in total or 200 hours of wall time, whichever is reached first.

We set $n_{\text{nodes}} = 100$ and we use the functions listed in Tables 2 to 4 in the Appendix, which have a maximum arity of $n_{\text{arity}} = 3$. If a function has a lower arity than $n_{\text{arity}} = 3$ the genes are still present in the genome and are simply ignored. Thus, there are $3 \cdot 100 = 300$ nodes in total. We set the mutation to alter exactly one node.

While the optimization of hyperparameters and choice of training schedules were not performed exhaustively, the objective of this work is the demonstration of GP as an alternative for creating inherently interpretable control policies, rather than a rigorous evaluation across the Atari benchmark. A more exhaustive comparison of optimization parameters for MAGE on the Atari benchmark, and a complete comparison with deep RL methods, is considered for future work, with the understanding that the Atari benchmark has a high computational cost [58].

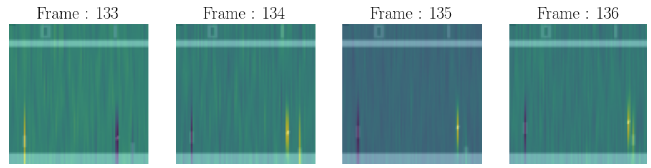
4.3 Visual Importance Maps

To compare the explanations attained from intepreting the MAGE graphs to those commonly found in XAI, we employed vision-based perturbation saliency mapping algorithms to identify regions of the input that are likely influential in the agent's decision-making process. Specifically, we employed the RISE [48] and the Occlusion Sensitivity methods [2, 63], both implemented through the *Xplique* python library [17]. These algorithms are widely used for interpreting RL agents [26]. Both are pertubartion-based methods, but they work differently.

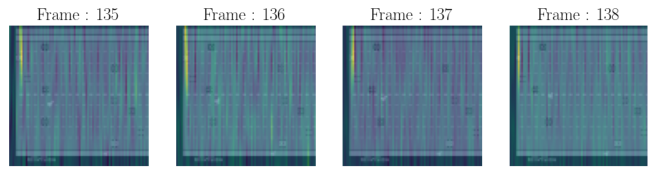
RISE [48] involves repeatedly applying random masks to the input image and observing how these affect the probability of the action of interest. Masking regions critical to the agent's decision is expected to result in a measurable reduction in the action's probability. The RISE saliency map is then constructed as a weighted average of the unmasked regions and their corresponding output probabilities for the action of interest.

The Occlusion Sensitivity method [2, 63] works by systematically sweeping a patch across the input image, occluding regions, and measures the magnitude of changes in the score for the action of interest. Regions where occlusion leads to significant changes in this score are inferred to be critical for the agent's decision-making process.

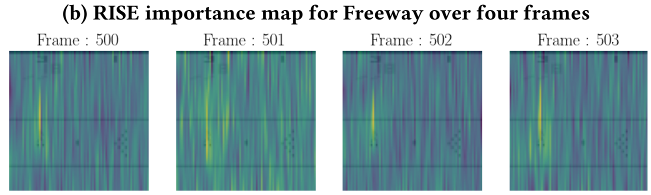
The hyperparameters of both methods have been demonstrated to impact the quality of the visual explanation [22, 26]. Given that the objective of hyperparameter tuning is in this case a subjective visual explanation, the choice of hyperparameters is often done manually. We visually analyzed the resulting saliency maps to determine hyperparameters, as some choices greatly deteriorated the explanatory nature of the saliency map. For the occlusion saliency maps, we used $patch\_size = 3$ and $patch\_stride = 3$, with $occlusion\_value = 0.$. For RISE saliency maps, we generated $n\_samples = 2000$ per frame, using a $grid\_size = 5$ and $preservation\_probability = 0.5$. For Pong and Bowling, we apply the perturbation methods to the 4th and final input frame, whereas for Freeway, we use the 3rd frame. These choices resulted in clear visual explanations of agent attention per action. We provide all code for optimization and explanation methods in our supplementary materials[1].



(a) RISE importance map for Pong over four frames

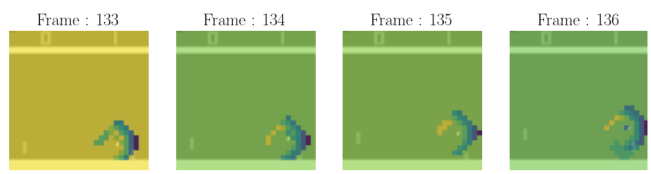


(b) RISE importance map for Freeway over four frames

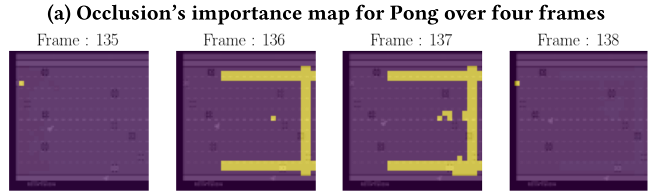


(c) RISE importance map for Bowling over four frames

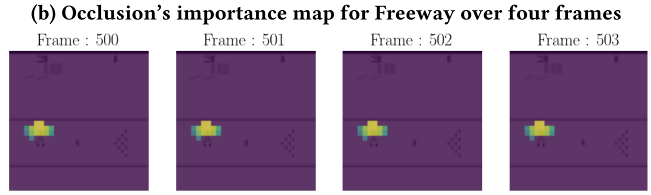
Figure 2: RISE importance maps for all games produced by perturbations over the fourth frame for (a), third frame for (b) and fourth frame for (c).



(a) Occlusion's importance map for Pong over four frames



(b) Occlusion's importance map for Freeway over four frames



(c) Occlusion's importance map for Bowling over four frames

Figure 3: Occlusion's importance maps for all games produced by perturbations over the fourth frame for (a), third frame for (b) and fourth frame for (c).

5 EXPERIMENTAL RESULTS AND INTERPRETABILITY DISCUSSION

In this section, we analyze the resulting policies for the three studied games. For each game, we present the policy's performance, including its score over training as well as its ability to generalize to different instances of the same game. Then, we investigate what the saliency maps reveal about the policy's behavior. Finally, we compare these insights to a direct analysis of the evolved program[2]. By examining the program itself, we aim to uncover the underlying logic driving the policy's decision-making, enabling a deeper understanding of its behavior beyond what can be inferred from saliency maps alone.

5.1 Pong

Pong is a two-player game where, in Atari, the player controls a paddle to hit a ball towards a game agent. When the ball passes the paddle on one player's side, the adversary scores. The maximum score in this game is 21, the average human score is 14.6 [44], and deep RL methods often achieve 21 [4].

The best policy found using MAGE achieved 21 during optimization, the maximum score. However, we note that this performance was limited to the random game seeds used during the optimization process. The policy did not generalize well to all scenarios, indicating that the solution is not broadly adaptable ($\mu_{100\_seeds} = 0.84$, $\sigma_{100\_seeds} = 21.1$, $best_{100\_seeds} = 21.0$). Through interpretation, we note that policy exploits a repetitive behavior present in the seeds they were trained on, where once the agent learns how to score a point under those conditions, it repetitively replicates the same actions. As this policy is only partially effective, it characterizes a local optimum decision-making process. Through explanation and

[1] https://github.com/camilodlt/MAGE-for-Atari-Gecco-2025

[2] Examples of evolved computational graphs are provided in the Appendix. When translating these graphs into sequential code, as shown in Figures 4 to 6, computations that had no effect on the final output were omitted. The presented code remains strictly functionally equivalent to the evolved graphs, preserving all relevant computations.

interpretation of the agent, we aim to understand such limits to robustness.

Visualizing RISE saliency maps across multiple gameplays[3] suggests that (1) the agent tracks the ball's position when preparing to respond, and (2) during the opponent's turn, both the ball's position and the opponent's paddle appear to be key areas of focus. This is also noticeable from Figure 2a. Importance maps using the Occlusion method in Figure 3a also suggest that the ball is important for the player's decision-making process. These findings provide preliminary insights into how the policy utilizes visual information to make decisions in the game.

While saliency maps provide some indication of which elements of the screen influence the policy's output, they do not clarify how or to what extent these elements contribute to decision-making. For instance, when the ball is approaching the player's paddle, it remains unclear how the policy decides to move upward and when it determines that it should stop.

For a more complete understanding, we interpret the MAGE graph. Figure 4 shows the computer code that maps exactly to the actions of the player. Careful analysis of this code reveals how each action is chosen. The Pong player chooses the NOOP action when the ball is on the left part of the screen, this enables it to way in its last position. Moreover, for the actions RIGHTFIRE and LEFTFIRE it tracks the ball twice, independently, but one of them introduces a small lag across frames which the Pong player uses to maneuver UP and DOWN movements. This lag biases the policy towards the RIGHTFIRE (i.e., the UP action) when the ball comes up diagonally towards it. The same visual lag helps it move down when the ball comes down diagonally towards it. Lastly, since both actions follow the same object in the screen, they often result in the same value, which seems to benefit the player as the paddle can gain some momentum if the same action has been pressed too many times. A simplified pseudocode version of this behavior can be seen in the appendix Algorithm 1.

```
ACTIONS = [0,4,5]
function evolved_pong_policy(frame1, frame2, frame3, frame4)
    # output NO OP
    mean_frame_2 = mean(frame2)
    size_axis = Float64(size(frame1)[1]) # 63
    frame2_closed = closing(frame2, 40.0)
    pos_last_line = argmaxposition_to(frame2_closed, size_axis,
      mean_frame_2) # sees if our paddle is in the bottom, it will
      either be (1,1) or (1,48)
    remove_top_and_bottom = notmaskfromto_vertical(frame1, 0., 10.)
    ball_pos = argmax_position(remove_top_and_bottom)
    approx_ball_has_crossed_middle = true_gt_or_eq(ball_pos,
      pos_last_line) # 1 if ball_pos >= pos_last_line on x and y
    ball_vertical_pos = horizontal_relative_argmax(
      remove_top_and_bottom)
    output_noop = ball_vertical_pos % approx_ball_has_crossed_middle
       # if the modulo returns NaN (a modulo by 0.), this action will
       be chosen

    # Output RIGHTFIRE (UP)
    closed = closing(frame2, 40.)
    edges_game = sobely(closed, 30.)
    expanded_players = morphogradient(edges_game, 10.) # get a view
      with the players and ball expanded
    edges_game_dilated = dilation(edges_game)
    expanded_players_last_frame = morphogradient(frame4, 30.) # also
       expand the objects in the last frame
    expanded_players_last_frame_removed_first_cols = maskfromtov(
      expanded_players_last_frame, 0., 10.)
    subtract_last_to_recent = subtract(
      expanded_players_last_frame_removed_first_cols,
      edges_game_dilated) # left bias when ball goes down, right bias
       when ball goes up
    recomposition = add(subtract_last_to_recent, expanded_players) #
       a lagged view
    output_up = exp(log_(horizontal_relative_argmax(recomposition)))

    # Output LEFTFIRE (DOWN)
    small_elements_in_game = tophat(frame2, -1.)
    output_down = horizontal_relative_argmax(small_elements_in_game)
       # usually the position of the ball

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

Figure 4: Code for the Pong player.

5.2 Freeway

Freeway is a simple game where you control a chicken trying to cross a busy highway without getting hit by cars. You can only move up or down, and each successful crossing earns a point. The goal is to cross as many times as possible before time runs out while avoiding traffic. The average human score in Freeway is 29.6 [44] and state-of-the-art deep RL methods achieve the maximum score of 34 [10].

For the game Freeway, the best policy out of the four runs demonstrated subhuman but satisfactory performance on the game seeds it was trained on ($\mu_{train\_seeds} = 21.3$, $\sigma_{train\_seeds} = 0.57$). Unlike Pong, however, the evolved policies also generalized effectively to unseen seeds, achieving consistent performance across different conditions ($\mu_{100\_seeds} = 22.5$, $\sigma_{100\_seeds} = 1.6$, $best_{100\_seeds} = 26.$). A visual inspection of the gameplay[3] reveals that the policies exhibit a clear intention to cross to the top of the screen as quickly as possible. From the saliency maps, we can observe in Figure 2b that a car in the top portion of the screen is also used for decisions. In Figure 3b, the same car is highlighted in some frames as well as other portions of the screen. Once again, the saliency maps generated provide useful information but it remains unclear how these important regions are used. The exact impact of the highlighted car on the action decision can only be assumed, but not precisely determined.

Figure 5 shows the computer code representing the evolved MAGE policy. Through careful analysis of it, it becomes clear that the policy indeed tracks a white car located at the top of the screen. It is the main car that the evolved policy tries to avoid. However, the policy's behavior is not limited to tracking the top car alone. As it is more clearly seen in the pseudocode in the appendix Algorithm 3, the policy predominantly moves UP when the white car is far from the left corner of the screen. The car running through the middle road line is also looked upon. Distances are calculated from this car to the center of the screen and from the top corner to the chicken's position. These quantities help determine if the middle car has crossed the chicken's vertical position. When it has and the white top car is close to the left corner, the agent chooses the NOOP actions until one of the cars leaves the screen.

[3] Videos of multiple gameplay episodes, along with RISE and Occlusion saliency maps and code are available in the online appendix of the paper: https://github.com/camilodlt/MAGE-for-Atari-Gecco-2025.

```
ACTIONS = [0,1,2]
function evolved_freeway_policy(frame1, frame2, frame3, frame4)
    # output NO OP
    car_in_the_middle = sobel(frame2, border = 50.)
    x_y_car = argmax_position(car_in_the_middle) # the car in the
      middle is usually highlighted because of the road lines
    blurry_frame3 = dilation(frame3, k = vertical_argmax(frame1))
    blurry_frame3 = gaussian_blur(blurry_frame3)
    x_y_center = center_of_mass(blurry_frame3) # ~ midpoint of the
      screen
    dir_car_to_center = direction_from_to(x_y_car, x_y_center)
    extract_chicken = tophat(frame3,  k = 100)
    x_y_chicken = argmax_position(extract_chicken)
    d_chicken_to_dir_car_to_center = direction_from_to(x_y_chicken,
      dir_car_to_center)
    x_y_chicken_second_frame = argmax_position(frame2)
    x_y_chicken_first_frame = argmax_position(frame1)
    d = direction_from_to(x_y_chicken_second_frame,
      x_y_chicken_first_frame) # has the chicken move up/down or not
      moved
    output1 = true_gt_or_eq(d, d_chicken_to_dir_car_to_center) #
      checks if the middle car has crossed the x coordinate of the
      chicken

    # Output UP
    from = exp(vertical_relative_argmax(sobely(sobelx(frame4))))
    k = vertical_argmax(frame4)
    dilated_once = dilation(frame3, k)
    dilated_twice = dilation(dilated_once, k)
    edges = sobely(dilated_twice)
    upper_edges = notmaskfromto_vertical(edges, from, 10) # gets the
       top part of the screen who can have some pixels if the top car
       is there
    y_pos_edge = vertical_argmax(upper_edges)
    output_2 = y_pos_edge * 0.5 # if the top car "disappears" from
      the top, this value will be low, high otherwise (unless the
      chicken is also there).

    # Output DOWN
    edges = sobelm(frame2, 50)
    opened = opening(edges, 50)
    edges_2 = sobelm(opened, 40)
    output3 = horizontal_relative_argmax(edges_2) # usually a very
      low value

    outputs = (output1, output_2, output_3)
    return ACTIONS[argmax(outputs)]
end
```

Figure 5: Code for the Freeway player.

While the saliency maps provided related insights, they offered only a partial view, ignoring for example the elements that condition the NOOP action. This detailed analysis shows that the policy is relatively simple and unsophisticated, as it tracks only two cars instead of all of them. The key point, however, is that the policy's existence in code format enabled us to identify its limitations. This verification can offer a valuable perspective, allowing us to go beyond solely considering the player's final score.

5.3 Bowling

Bowling is a game where the player rolls a ball down a lane to knock over pins at the end. The player has ten frames to make an action, with two chances per frame to knock down all ten pins. The more pins you hit, the higher your score, with strikes (all pins in one roll) and spares (all in two rolls) giving bonus points. The average human score in Bowling is 160.7 [44] and state-of-the-art deep RL methods score over 200 points [5].

For the game Bowling, our results were particularly promising, with the best-performing policy surpassing human performance. The best policy achieved a high performance on seeds used for training (μ_{train_seeds} = 211., σ_{train_seeds} = 11.5), similar to state-of-the-art deep RL methods [5], and outperformed the human average on unseen scenarios (μ_{100_seeds} = 172.2, σ_{100_seeds} = 37.4, max_{100_seeds} = 223.).

The saliency maps, generated using both RISE and the Occlusion method Figures 2c and 3c, indicate that the agent places significant importance on the player's position in its decision-making process. However, beyond this observation, the saliency maps provide limited insight into the specific mechanisms driving the agent's behavior. In this game, the player can still take actions once the ball is released, so attention that follows the ball or focuses on the pins would be expected. However, we find that the saliency is diffuse, apart from a small focus on the player sprite.

A deeper understanding is obtained through an analysis of the evolved program's underlying code. The program in Figure 6 reveals that the agent never selects the DOWN action, as it is consistently assigned a low support value. The support for the UP action is strongly correlated with the distance between the darkest pixel on the screen—usually corresponding to the player or the ball—and the screen's bottom-right corner. Consequently, when the agent is in the bottom-left portion of the screen, the UP action is preferred. While the player moves upward, the support for the FIRE action is triggered once the player's "head" enters the top portion of the screen, overwhelming all other action supports. This moment notably coincides with the player being perfectly aligned at the center of the bowling pins.

If the initial throw does not result in a strike, the agent remains stationary, continuing to FIRE as the ball returns. The environment's mechanics eventually reposition the player to the initial starting point. Since the agent continuously fired during the previous phase, it automatically launches the ball upon receiving it. The agent then repeats its earlier behavior: selecting the UP action to move the ball diagonally toward the remaining pins.

In the case of a strike on the first attempt, the agent continues to FIRE as before, but ceases to do so once the ball is halfway through its return trajectory. This cessation coincides with a visual "blinking" effect introduced by the environment which makes the player disappear from the top portion of the screen. In this scenario, the NOOP action dominates, as it responds to the number of distinct gray pixel intensities present on the screen.

6 DISCUSSION

This study demonstrates the potential of GP to evolve inherently interpretable control policies for Atari games, offering a promising approach to applications that require transparency. The inherent interpretability of the evolved programs represents a significant advantage, particularly for critical domains where understanding and auditing decision-making processes is highly important [1, 26].

We employed common post-hoc explainability methods as an initial approach to gain insight into the logic behind the evolved policies. These methods provided a preliminary understanding of how the policies process states and make decisions. However, the insights gained from saliency maps are often insufficient [56], functioning more as an exploratory tool rather than providing a comprehensive explanation of the policies' behavior [3].

```
ACTIONS = [0,1,2,3]
function evolved_policy_bowling(frame1, frame2, frame3, frame4)
    # OUTPUT NOOP
    output_noop = reduce_ncolors(frame2)

    # OUTPUT FIRE
    remove_top_bottom = maskfromto(frame4, exp(-1.0), 60.0)
    exponent = vertical_argmax(remove_top_bottom) # has the player
      reach the top ?
    # normally the exponent will be one if the player hasn't, 3
      otherwise
    n_diff_pixel_values = reduce_ncolors(frame1)
    details = tophat(frame4, n_diff_pixel_values,) # big Kernel =>
      makes pins salient
    whitest_pixel = argmax_position(details) # usually the pins
    whitest_pixel_orig_frame1 = argmax_position(frame1) # usually
      the head of the player
    dist_player_pins = dist(whitest_pixel_orig_frame1, whitest_pixel
      )
    output_fire = dist_player_pins^exponent

    # OUTPUT UP
    blackest_pixel = argmin_position(dilation(frame4))
    constant_coordinate = (27, 84) # last pixel
    closed_frame1 = closing(frame1)
    player = argmax_position(closed_frame1)
    player_vector = direction(player, constant_coordinate) # player
      to bottom_left
    output_up = dist_second(player_vector, blackest_pixel) # compare
       horizontal coordinate for both

    # OUTPUT DOWN
    output_down = 1.0

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

Figure 6: Code for the Bowling player.

First, the sequential nature of the game introduces a challenge, as static saliency maps reflect poorly the relationship between agent actions and its value over the long-term. For example, the complex behavior of the Bowling agent after throwing the ball is poorly captured by saliency, even in a video format. Second, saliency gives an incomplete explanation over the multitude of information contained in the four input frames. As seen in the Freeway agent, saliency explanations can given an incomplete picture of what information is being used, and how it is being used. Finally, the outputs of saliency methods are sensitive to the parameters used [25, 26] which complicates their interpretation —particularly when two saliency maps for the same state yield conflicting results.

However, certain challenges and limitations associated with the current methodology are worth of discussion, many of which can be addressed in future work. The first is the interpretability of the evolved pipelines. The graphs evolved by MAGE require expertise for effective analysis. Understanding the underlying logic and interactions within the functions demands familiarity with the components and their operations. Additionally, as the pipelines execute over time and produce multiple outputs, the process of tracking these interactions introduces complexity. Evolutionary processes can also yield solutions that, while functionally effective, may deviate from conventional human logic or exhibit redundancy, making them less immediately intuitive. One avenue for exploration is the use of LLMs to provide preliminary explanations of the full code version of evolved graphs.

The second challenge is to increase the overall performance of agents evolved with MAGE. The agents we trained demonstrated limited robustness against the stochasticity inherent in the Atari benchmark, and did not consistently match state-of-the-art deep RL. It is important to note, however, that robustness was not a specific objective during training. Furthermore, assessing the best performance for each game would require extensive hyperparameter tuning, followed by fitting sufficient models to enable a statistically sound performance estimation.

Despite these considerations, the flexibility and potential of the proposed method offer clear pathways for addressing these limitations. Robustness can be explicitly targeted in future iterations by incorporating it as an objective during the evolutionary process. For critical applications where interpretability is highly important, it is reasonable to expect dedicated expertise and resources to aid in the interpretation of evolved pipelines. Furthermore, evolving policies explicitly for interpretability by introducing complexity penalties—whether computationally defined or guided by user preferences—represents a possible avenue for future work.

We demonstrate in this work that it is possible to search for high-performing policies that are natively interpretable. We showed that MAGE—a GGP approach capable of handling diverse data types—can create human-level policies for various vision-based control tasks. Our primary objective was to synthesize interpretable control policies. Through experiments on three Atari games, *Pong*, *Freeway* and *Bowling*, we demonstrated that the evolved policies are fully transparent. The exact programs governing agent behavior are fully accessible, allowing all operations performed by the agent to be audited and examined. This ensures that the decision-making process is entirely transparent, with no hidden components. Moreover, the programs are decomposable, meaning individual operations can be analyzed in isolation to understand their contribution to the overall policy. Once the program logic is fully understood, it can often be neatly simplified, making it easier to interpret, apply, and explain.

We also demonstrate that the explanations given through interpretation of the evolved programs are more precise and rigorous than those derived through post-hoc explanation methods. We show that post-hoc explanation methods can fail to explain parts of the decision pipeline, making them unreliable for critical applications. This makes an argument, as has been proposed elsewhere [50], that research should focus on the creation of interpretable AI methods in addition to the explanation of black-box ones.

ACKNOWLEDGMENTS

This project used compute resources from the CALMIP projects P21049 and P24042.

REFERENCES

[1] Julius A Adebayo and others. 2016. *FairML: ToolBox for diagnosing bias in predictive modeling.* PhD Thesis. Massachusetts Institute of Technology.

[2] Marco Ancona, Enea Ceolini, Cengiz Öztireli, and Markus Gross. 2018. Towards better understanding of gradient-based attribution methods for Deep Neural Networks. https://doi.org/10.48550/arXiv.1711.06104 arXiv:1711.06104 [cs].

[3] Akanksha Atrey, Kaleigh Clary, and David Jensen. 2019. Exploratory not explanatory: Counterfactual analysis of saliency maps for deep reinforcement learning. *arXiv preprint arXiv:1912.05743* (2019).

[4] Adrià Puigdomènech Badia, Bilal Piot, Steven Kapturowski, Pablo Sprechmann, Alex Vitvitskyi, Daniel Guo, and Charles Blundell. 2020. Agent57: outperforming the Atari human benchmark. In *Proceedings of the 37th International Conference*

on Machine Learning (ICML'20, Vol. 119). JMLR.org, 507–517.

[5] Adrià Puigdomènech Badia, Bilal Piot, Steven Kapturowski, Pablo Sprechmann, Alex Vitvitskyi, Zhaohan Daniel Guo, and Charles Blundell. 2020. Agent57: Outperforming the atari human benchmark. In *International conference on machine learning*. PMLR, 507–517.

[6] Hubert Baniecki and Przemyslaw Biecek. 2024. Adversarial attacks and defenses in explainable artificial intelligence: A survey. *Information Fusion* (2024), 102303. Publisher: Elsevier.

[7] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. 2013. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research* 47 (2013), 253–279.

[8] Leonardo Lucio Custode and Giovanni Iacca. 2020. Evolutionary learning of interpretable decision trees. *arXiv preprint arXiv:2012.07723* (2020).

[9] Leonardo Lucio Custode and Giovanni Iacca. 2022. Interpretable pipelines with evolutionary optimized modules for reinforcement learning tasks with visual inputs. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. 224–227.

[10] Will Dabney, Georg Ostrovski, David Silver, and Rémi Munos. 2018. Implicit quantile networks for distributional reinforcement learning. In *International conference on machine learning*. PMLR, 1096–1105.

[11] Camilo De La Torre, Kévin Cortacero, Sylvain Cussat-Blanc, and Dennis Wilson. 2024. Multimodal Adaptive Graph Evolution. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion (GECCO '24 Companion)*. Association for Computing Machinery, New York, NY, USA, 499–502. https://doi.org/10.1145/3638530.3654347

[12] 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.

[13] M Diana and JJBJoS Marescaux. 2015. Robotic surgery. *Journal of British Surgery* 102, 2 (2015), e15–e28. Publisher: Oxford University Press.

[14] Mengnan Du, Ninghao Liu, and Xia Hu. 2019. Techniques for interpretable machine learning. *Commun. ACM* 63, 1 (2019), 68–77. Publisher: ACM New York, NY, USA.

[15] Frederik Ebert, Chelsea Finn, Sudeep Dasari, Annie Xie, Alex Lee, and Sergey Levine. 2018. Visual foresight: Model-based deep reinforcement learning for vision-based robotic control. *arXiv preprint arXiv:1812.00568* (2018).

[16] Jiajun Fan. 2023. A Review for Deep Reinforcement Learning in Atari:Benchmarks, Challenges, and Solutions. https://doi.org/10.48550/arXiv.2112.04145 arXiv:2112.04145 [cs].

[17] Thomas Fel, Lucas Hervier, David Vigouroux, Antonin Poche, Justin Plakoo, Remi Cadene, Mathieu Chalvidal, Julien Colin, Thibaut Boissin, Louis Bethune, Agustin Picard, Claire Nicodeme, Laurent Gardes, Gregory Flandin, and Thomas Serre. 2022. Xplique: A Deep Learning Explainability Toolbox. https://doi.org/10.48550/arXiv.2206.04394 arXiv:2206.04394 [cs].

[18] Aidin Ferdowsi, Ursula Challita, Walid Saad, and Narayan B Mandayam. 2018. Robust deep reinforcement learning for security and safety in autonomous vehicle systems. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 307–312.

[19] Andrea Ferigo, Leonardo Lucio Custode, and Giovanni Iacca. 2023. Quality–diversity optimization of decision trees for interpretable reinforcement learning. *Neural Computing and Applications* (2023), 1–12. Publisher: Springer.

[20] Claire Glanois, Paul Weng, Matthieu Zimmer, Dong Li, Tianpei Yang, Jianye Hao, and Wulong Liu. 2024. A survey on interpretable reinforcement learning. *Machine Learning* (2024), 1–44. Publisher: Springer.

[21] Nathan A Greenblatt. 2016. Self-driving cars and the law. *IEEE spectrum* 53, 2 (2016), 46–51. Publisher: IEEE.

[22] Samuel Greydanus, Anurag Koul, Jonathan Dodge, and Alan Fern. 2018. Visualizing and understanding atari agents. In *International conference on machine learning*. PMLR, 1792–1801.

[23] Simon Harding, Vincent Graziano, Jürgen Leitner, and Jürgen Schmidhuber. 2012. Mt-cgp: Mixed type cartesian genetic programming. In *Proceedings of the 14th annual conference on Genetic and evolutionary computation*. 751–758.

[24] Daniel Hein, Steffen Udluft, and Thomas A Runkler. 2018. Interpretable policies for reinforcement learning by genetic programming. *Engineering Applications of Artificial Intelligence* 76 (2018), 158–169. Publisher: Elsevier.

[25] Alexandre Heuillet, Fabien Couthouis, and Natalia Díaz-Rodríguez. 2021. Explainability in deep reinforcement learning. *Knowledge-Based Systems* 214 (2021), 106685. Publisher: Elsevier.

[26] Tobias Huber, Benedikt Limmer, and Elisabeth André. 2022. Benchmarking perturbation-based saliency maps for explaining Atari agents. *Frontiers in Artificial Intelligence* 5 (2022), 903875. Publisher: Frontiers Media SA.

[27] Elia Kaufmann, Leonard Bauersfeld, Antonio Loquercio, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza. 2023. Champion-level drone racing using deep reinforcement learning. *Nature* 620, 7976 (2023), 982–987. Publisher: Nature Publishing Group UK London.

[28] Stephen Kelly and Malcolm I Heywood. 2017. Emergent tangled graph representations for Atari game playing agents. In *Genetic Programming: 20th European Conference, EuroGP 2017, Amsterdam, The Netherlands, April 19-21, 2017, Proceedings 20*. Springer, 64–79.

[29] Stephen Kelly and Malcolm I Heywood. 2017. Multi-task learning in atari video games with emergent tangled program graphs. In *Proceedings of the Genetic and Evolutionary Computation Conference*. 195–202.

[30] Stephen Kelly, Daniel S Park, Xingyou Song, Mitchell McIntire, Pranav Nashikkar, Ritam Guha, Wolfgang Banzhaf, Kalyanmoy Deb, Vishnu Naresh Boddeti, Jie Tan, and others. 2023. Discovering Adaptable Symbolic Algorithms from Scratch. *arXiv preprint arXiv:2307.16890* (2023).

[31] Stephen Kelly, Tatiana Voegerl, Wolfgang Banzhaf, and Cedric Gondro. 2021. Evolving hierarchical memory-prediction machines in multi-task reinforcement learning. *Genetic Programming and Evolvable Machines* 22 (2021), 573–605. Publisher: Springer.

[32] Samantha Krening, Brent Harrison, Karen M Feigh, Charles Lee Isbell, Mark Riedl, and Andrea Thomaz. 2016. Learning from explanations using sentiment and advice in RL. *IEEE Transactions on Cognitive and Developmental Systems* 9, 1 (2016), 44–55. Publisher: IEEE.

[33] Xiaodan Liang, Tairui Wang, Luona Yang, and Eric Xing. 2018. Cirl: Controllable imitative reinforcement learning for vision-based self-driving. In *Proceedings of the European conference on computer vision (ECCV)*. 584–599.

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

[35] Romulus Lungu, Mihai Lungu, and Lucian Teodor Grigorie. 2013. Automatic control of aircraft in longitudinal plane during landing. *IEEE Trans. Aerospace Electron. Systems* 49, 2 (2013), 1338–1350. Publisher: IEEE.

[36] Marlos C Machado, Marc G Bellemare, Erik Talvitie, Joel Veness, Matthew Hausknecht, and Michael Bowling. 2018. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research* 61 (2018), 523–562.

[37] Marlos C Machado, Marc G Bellemare, Erik Talvitie, Joel Veness, Matthew Hausknecht, and Michael Bowling. 2018. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research* 61 (2018), 523–562.

[38] Paula Maddigan, Andrew Lensen, and Bing Xue. 2024. Explaining genetic programming trees using large language models. *arXiv preprint arXiv:2403.03397* (2024).

[39] Francesco Marchetti, Gloria Pietropolli, Federico Julian Camerota Verdù, Mauro Castelli, and Edmondo Minisci. 2024. Automatic design of interpretable control laws through parametrized Genetic Programming with adjoint state method gradient evaluation. *Applied Soft Computing* 159 (2024), 111654. Publisher: Elsevier.

[40] Eric Medvet and Giorgia Nadizar. 2023. GP for Continuous Control: Teacher or Learner? The Case of Simulated Modular Soft Robots. *Genetic Programming Theory and Practice XX* (2023). Publisher: Springer.

[41] Yi Mei, Qi Chen, Andrew Lensen, Bing Xue, and Mengjie Zhang. 2022. Explainable artificial intelligence by genetic programming: A survey. *IEEE Transactions on Evolutionary Computation* 27, 3 (2022), 621–641. Publisher: IEEE.

[42] 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.

[43] Tim Miller. 2019. Explanation in artificial intelligence: Insights from the social sciences. *Artificial intelligence* 267 (2019), 1–38. Publisher: Elsevier.

[44] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, and others. 2015. Human-level control through deep reinforcement learning. *nature* 518, 7540 (2015), 529–533. Publisher: Nature Publishing Group.

[45] Giorgia Nadizar, Eric Medvet, and Dennis Wilson. 2024. Searching for a Diversity of Interpretable Graph Control Policies. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '24)*. Association for Computing Machinery, New York, NY, USA, 933–941. https://doi.org/10.1145/3638529.3653987

[46] Giorgia Nadizar, Eric Medvet, and Dennis G Wilson. 2024. Naturally Interpretable Control Policies via Graph-based Genetic Programming. In *European Conference on Genetic Programming (Part of EvoStar)*. Springer.

[47] Giorgia Nadizar, Luigi Rovito, Andrea De Lorenzo, Eric Medvet, and Marco Virgolin. 2024. An Analysis of the Ingredients for Learning Interpretable Symbolic Regression Models with Human-in-the-Loop and Genetic Programming. *ACM Trans. Evol. Learn. Optim.* (2024). Publisher: ACM New York, NY.

[48] V Petsiuk. 2018. Rise: Randomized Input Sampling for Explanation of black-box models. *arXiv preprint arXiv:1806.07421* (2018).

[49] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. " Why should i trust you?" Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. 1135–1144.

[50] Cynthia Rudin. 2019. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence* 1, 5 (May 2019), 206–215. https://doi.org/10.1038/s42256-019-0048-x

Publisher: Nature Publishing Group.

[51] Christian Rupprecht, Cyril Ibrahim, and Christopher J Pal. 2019. Finding and Visualizing Weaknesses of Deep Reinforcement Learning Agents. In *International Conference on Learning Representations*.

[52] Erica Salvato, Arnob Ghosh, Gianfranco Fenu, and Thomas Parisini. 2021. Control of a mixed autonomy signalised urban intersection: An action-delayed reinforcement learning approach. In *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*. IEEE, 2042–2047.

[53] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, and others. 2020. Mastering atari, go, chess and shogi by planning with a learned model. *Nature* 588, 7839 (2020), 604–609. Publisher: Nature Publishing Group.

[54] Alexander Sieusahai and Matthew Guzdial. 2021. Explaining deep reinforcement learning agents in the atari domain through a surrogate model. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, Vol. 17. 82–90. Issue: 1.

[55] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. 2013. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034* (2013).

[56] Wolfgang Stammer, Patrick Schramowski, and Kristian Kersting. 2021. Right for the Right Concept: Revising Neuro-Symbolic Concepts by Interacting with their Explanations. https://doi.org/10.48550/arXiv.2011.12854 arXiv:2011.12854 [cs].

[57] Chen Tessler, Yonathan Efroni, and Shie Mannor. 2019. Action robust reinforcement learning and applications in continuous control. In *International Conference on Machine Learning*. PMLR, 6215–6224.

[58] Marin Toromanoff, Emilie Wirbel, and Fabien Moutarde. 2019. Is Deep Reinforcement Learning Really Superhuman on Atari? Leveling the playing field. https://doi.org/10.48550/arXiv.1908.04683 arXiv:1908.04683 [cs].

[59] Mark Towers, Ariel Kwiatkowski, Jordan Terry, John U. Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Markus Krimmel, Arjun KG, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Hannah Tan, and Omar G. Younis. 2024. Gymnasium: A Standard Interface for Reinforcement Learning Environments. https://doi.org/10.48550/arXiv.2407.17032 arXiv:2407.17032 [cs].

[60] Mathurin Videau, Alessandro Leite, Olivier Teytaud, and Marc Schoenauer. 2022. Multi-objective genetic programming for explainable reinforcement learning. In *European Conference on Genetic Programming (Part of EvoStar)*. Springer, 278–293.

[61] Dennis G Wilson, Sylvain Cussat-Blanc, Hervé Luga, and Julian F Miller. 2018. Evolving simple programs for playing atari games. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '18)*. Association for Computing Machinery, New York, NY, USA, 229–236. https://doi.org/10.1145/3205455.3205578

[62] Dennis G Wilson, Julian F Miller, Sylvain Cussat-Blanc, and Hervé Luga. 2018. Positional cartesian genetic programming. *arXiv preprint arXiv:1810.04119* (2018).

[63] Matthew D. Zeiler and Rob Fergus. 2014. Visualizing and Understanding Convolutional Networks. In *Computer Vision – ECCV 2014*, David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars (Eds.). Springer International Publishing, Cham, 818–833. https://doi.org/10.1007/978-3-319-10590-1_53

[64] Quan-shi Zhang and Song-Chun Zhu. 2018. Visual interpretability for deep learning: a survey. *Frontiers of Information Technology & Electronic Engineering* 19, 1 (2018), 27–39. Publisher: Springer.

[65] Ryan Zhou, Jaume Bacardit, Alexander EI Brownlee, Stefano Cagnoni, Martin Fyvie, Giovanni Iacca, John McCall, Niki van Stein, David J Walker, and Ting Hu. 2024. Evolutionary Computation and Explainable AI: A Roadmap to Understandable Intelligent Systems. *IEEE Transactions on Evolutionary Computation* (2024). Publisher: IEEE.

A FUNCTION LIBRARIES

A.1 Image function library

Table 2: List of functions returning an image with their descriptions and arities. If a function has multiple possible arities, that means that multiple signatures exists for the same function. The dispatched function will depend on the types of the inputs. Normally a function has default values (lower arity) and a version with higher arity which accepts scalar parameters.

Function Name	Description	Arity
add	Adds two images.	2
badd	Broadcasts the addition of an image and a scalar.	2
binarize_adaptive	Adaptive threshold	1, 2, 3
binarize_manual	Parameterized Threshold by a scalar	1, 2
bmult	Broadcasts the multiplication of an image and a scalar	2
bsubtract	Broadcasts the subtraction of an image and a scalar	2
bothat	Applies the bothat filter to an image	1, 2
closing	Applies the closing operation to an image	1, 2
dilation	Applies the dilation operation to an image	1, 2
erosion	Erodes the image	1, 2
exp	Exponentiates the pixel values of an image	1
fastscanning	Applies the *fastscanning* segmentation on an image	1, 2
gaussian	Applies a gaussian filter on an image	1, 2, 3
identity	Returns the image	1
if_else_multiplexer	Returns an image *a* or an image *b* depending on the value of the first scalar parameter.	3
invert	Inverts the intensities of an image	1
log_	Applies a safe *log* to each pixel on an image	1
maskfromtoh	Covers in black (pixels with value of 0.) regions inside the *from* and *to* scalar parameters (vertically).	3
maskfromtoh_relative	Covers in black (pixels with value of 0.) regions inside the *from* and *to* scalar parameters (vertically and uses normalized parameters between 0 and 1).	3
maskfromtov	Covers in black (pixels with value of 0.) regions inside the *from* and *to* scalar parameters (horizontally).	3
maskfromtov_relative	Covers in black (pixels with value of 0.) regions inside the *from* and *to* scalar parameters (horizontally and uses normalized parameters between 0 and 1).	3
morphogradient	Applies the *morphogradient* filter to an image.	1, 2
morpholaplace	Applies the *morpholaplace* filter to an image.	1, 2
notmaskbycolor	Returns the binarized image: white where the color is the requested one, black elsewhere.	2
notmaskfromtoh	Covers in black (pixels with value of 0.) regions outside the *from* and *to* scalar parameters (vertically).	3
notmaskfromtoh_relative	Covers in black (pixels with value of 0.) regions outside the *from* and *to* scalar parameters (vertically and uses normalized parameters between 0 and 1).	3
notmaskfromtov	Covers in black (pixels with value of 0.) regions outside the *from* and *to* scalar parameters (horizontally).	3
notmaskfromtov_relative	Covers in black (pixels with value of 0.) regions outside the *from* and *to* scalar parameters (horizontally and uses normalized parameters between 0 and 1) .	3
opening	Applies the opening operation to an image	2
tophat	Applies the tophat operation on an image	1, 2
maskeqt	Indicator function, where pixel values equal to a scalar parameter are 1.	1, 2
maskgt	Indicator function, where pixel values greater than a scalar parameter are 1.	1, 2
masklt	Indicator function, where pixel values less than a scalar parameter are 1.	1, 2
mult	Multiplies and clamps between 0 and 1 two images.	1
ones	Returns an image full of 1s of the same size and the input image.	1
powerof	Elevated each pixel value to the power of a scalar parameter	2
sobelm	Applies the *sobel* filter (horizontally and vertically) to the image	1, 2
sobelx	Applies the *sobel* filter (horizontally) to the image.	1, 2
sobely	Applies the *sobel* filter (vertically) to the image.	1, 2
subtract	Subtracts two images (values clamped).	2
watershed	Applies the *watershed* segmentation algorithm to the image.	3
zeros	Returns an image full of 0s of the same size and the input image	1

A.2 Scalar (float) function library

Table 3: List of functions returning a scalar with their descriptions and arities. If a function has multiple possible arities, the same comment in Table 2 applies.

Function Name	Description	Arity
dist	Square distance between two tuples of integers	2
dist_first	Difference between the first element of two tuples of integers	2
dist_second	Difference between the second element of two tuples of integers	2
exp_	Returns the exponent of a scalar	1
gt_on_one	Returns 1 if at least one element of the tuple a is greater than the same element of a tuple b.	2
horizontal_argmax	Returns the second element of the argmax of an image.	1
horizontal_relative_argmax	Returns the second element of the argmax of an image (coordinate is relative—divided by the size of the axis).	2
identity_float	Returns the scalar value.	1
if_else_multiplexer	Returns the scalar a or scalar b depending on the value of the first scalar parameter.	3
is_gt	Returns 1 if the first parameter is greater than the second parameter. 0 otherwise.	2
is_lt	Returns 1 if the first parameter is less than the second parameter. 0 otherwise.	2
is_eq_to	Returns 1 if the first scalar parameter is equal to the second.	2
less_on_one	Returns 1 if at least one element of the tuple a is less than the same element of a tuple b.	2
log10_	Safe *log* with base 10 of a scalar value.	1
log_	Safe natural logarithm of a scalar value	1
modulo	Unsafe modulo between two scalar values.	2
not	Returns 1 if the scalar parameter is less than 0.5	1
number_div	Divides one scalar by another	2
number_minus	Subtracts one scalar to another	2
number_mult	Multiplies one scalar by another	2
number_sum	Sums two scalars.	2
pi_	Returns π	0
power_of	Elevates a scalar value to the power of another	2
reduce_biggestAxis	Returns the size of the largest axis from an image	1
reduce_histMode	Returns the most common pixel value from an image.	1
reduce_length	Length of an image (number of pixels)	1
reduce_maximum	Returns the maximum pixel value from an image.	1
reduce_mean	Returns the average pixel value from an image.	1
reduce_median	Returns the median pixel value from an image.	1
reduce_minimum	Returns the minimum pixel value from an image.	1
reduce_nColors	Returns the number of unique pixel value from an image.	1
reduce_propBlack	Returns the proportion of pixels at a value of 1.	1
reduce_propWhite	Returns the proportion of pixels at a value of 0.	1
reduce_smallerAxis	Returns the size of the smallest axis from an image	1
reduce_std	Returns the standard deviation of pixel values from an image.	1
ret_1	Returns 1	0
safe_div	Safe division of two scalar values	2
true_gt	Returns 1 if elements of a tuple are greater than the respective elements of another tuple.	2
true_gteq	Returns 1 if both elements of a tuple are greater or equal than the respective elements of another tuple.	2
true_less	Returns 1 if elements of a tuple are less than the respective elements of another tuple.	2
true_lesseq	Returns 1 if elements of a tuple are less than or equal than the respective elements of another tuple.	2
vertical_argmax	Returns the first element of the argmax of an image.	1
vertical_relative_argmax	Returns the first element of the argmax of an image (coordinate is relative—divided by the size of the axis).	1

A.3 Tuple function library

Table 4: List of functions returning a tuple of two integers with their descriptions and arities. If a function has multiple possible arities, the same comment in Table 2 applies.

Function Name	Description	Arity
argmax_position	Coordinates of the *argmax* of an image	1
argmaxposition_from	Using relative coordinates parameters, returns the *argmax* coordinates of an image as if it was cropped from the first and second parameters to the end (coordinates are then rescaled).	3
argmaxposition_to	Using relative coordinates parameters, returns the *argmax* coordinates of an image as if it was cropped from the first and second parameters to the end.	3
argmin_position	Coordinates of the *argmin* of an image	1
center_of_mass	Coordinates of the center pixel weighted by pixel values.	1
direction	Coordinates of the vector going from the first tuple to the second tuple.	2

B PSEUDOCODE

The pseudocode presented in this section captures the core logic of the evolved agents while abstracting away low-level details. Its purpose is to provide a high-level interpretation of the agent's decision-making process, serving as a bridge between the evolved computational graph and a human-understandable strategy or policy. A fully faithful translation, including all implementation details, can be found either in the sequential code shown in Figures 4 to 6 or directly in the— equivalent—computational graphs shown in Figures 7 to 9 in the Appendix.

B.1 Pseudocode Pong player

Algorithm 1 Pseudo-code of Pong player.

Input: f_1, f_2, f_3, f_4 ▷ last 4 game frames (f_4 is current)
Output: action ∈ {NOOP,UP,DOWN}
  ▷ decision-making for NOOP
  $x_{\text{mid}} \leftarrow f_{\text{width}}/2$ ▷ horizontal midpoint of frame
  if $x_{\text{ball},f_2} < x_{\text{mid}}$ **then** ▷ ball in frame 2 did not cross midpoint
    return NOOP
  end if
  ▷ decision-making for UP
  $d_{f_2}, d_{f_4} \leftarrow \text{dilate}(f_2), \text{dilate}(f_4)$ ▷ image dilation on frames
  bias ← img_diff(d_{f_4}, d_{f_2}) ▷ left bias when ball goes down, right bias when ball goes up
  $\text{biased}_{f_2} \leftarrow \text{img\_add}(\text{bias}, d_{f_2})$ ▷ biased frame
  $x_{\text{up}} \leftarrow \text{x\_of\_brightest}(\text{biased}_{f_2})$
  ▷ decision-making for DOWN
  $\text{details}_{f_2} \leftarrow \text{top-hat}(f_2)$ ▷ extract details/small elements
  $x_{\text{down}} \leftarrow \text{x\_of\_brightest}(\text{details}_{f_2})$
  if $x_{\text{up}} \geq x_{\text{down}}$ **then**
    return UP
  else
    return DOWN
  end if

B.2 Pseudocode Freeway player

Algorithm 2 Pseudo-code of simplified Freeway player.

Input: f_1, f_2, f_3, f_4 ▷ last 4 game frames (f_4 is current)
Output: action ∈ {NOOP,UP,DOWN}
  top_car_near ← is_top_car_near(f_4) ▷ assess if the white top car is near to the left part of the screen
  ▷ check if the middle car has crossed the x coordinate of the chicken
  middle_car_crossed ← has_middle_car_crossed(f_2, 20)
  ▷ checking the top car is irrelevant if the chicken is close enough to the top
  if chicken_close_to_top(f_4) **then**
    top_car_near ← false
  end if
  if top_car_near and not middle_car_crossed **then**
    return NOOP
  else
    return UP
  end if

B.3 Pseudocode Bowling player

Algorithm 3 Pseudo-code of simplified Bowling player.

Input: f_1, f_2, f_3, f_4 ▷ last 4 game frames (f_4 is current)
Output: action ∈ {NOOP, FIRE, UP, DOWN}
  ▷ Never do DOWN
  ▷ Verify the top portion of the screen
  player_reach_top ← has_player_reach_top(f_4)
  if screen_blinks(f_4) and player_reach_top **then**
    player_reach_top ← false ▷ A strike causes a blink which causes the player to disappear momentarily
  end if
  if player_reach_top **then**
    return FIRE ▷ and keeps pressing FIRE
  end if
  nb_grays ← nb_of_distinct_gray_values(f_2) ▷ The only support for NOOP
  ▷ Usually the ball or the player
  darkest_pixel ← get_darkest_horizontal_position(f_4)
  dist_to_corner ← approx_dist_to_corner(darkest_pixel)
  if nb_grays ≥ dist_to_corner **then**
    return NOOP
  else
    return UP
  end if

C SEQUENCE OF OPERATIONS (COMPUTATIONAL GRAPH) OF THE BEST POLICY FOR EACH GAME

C.1 Pong's evolved computational graph

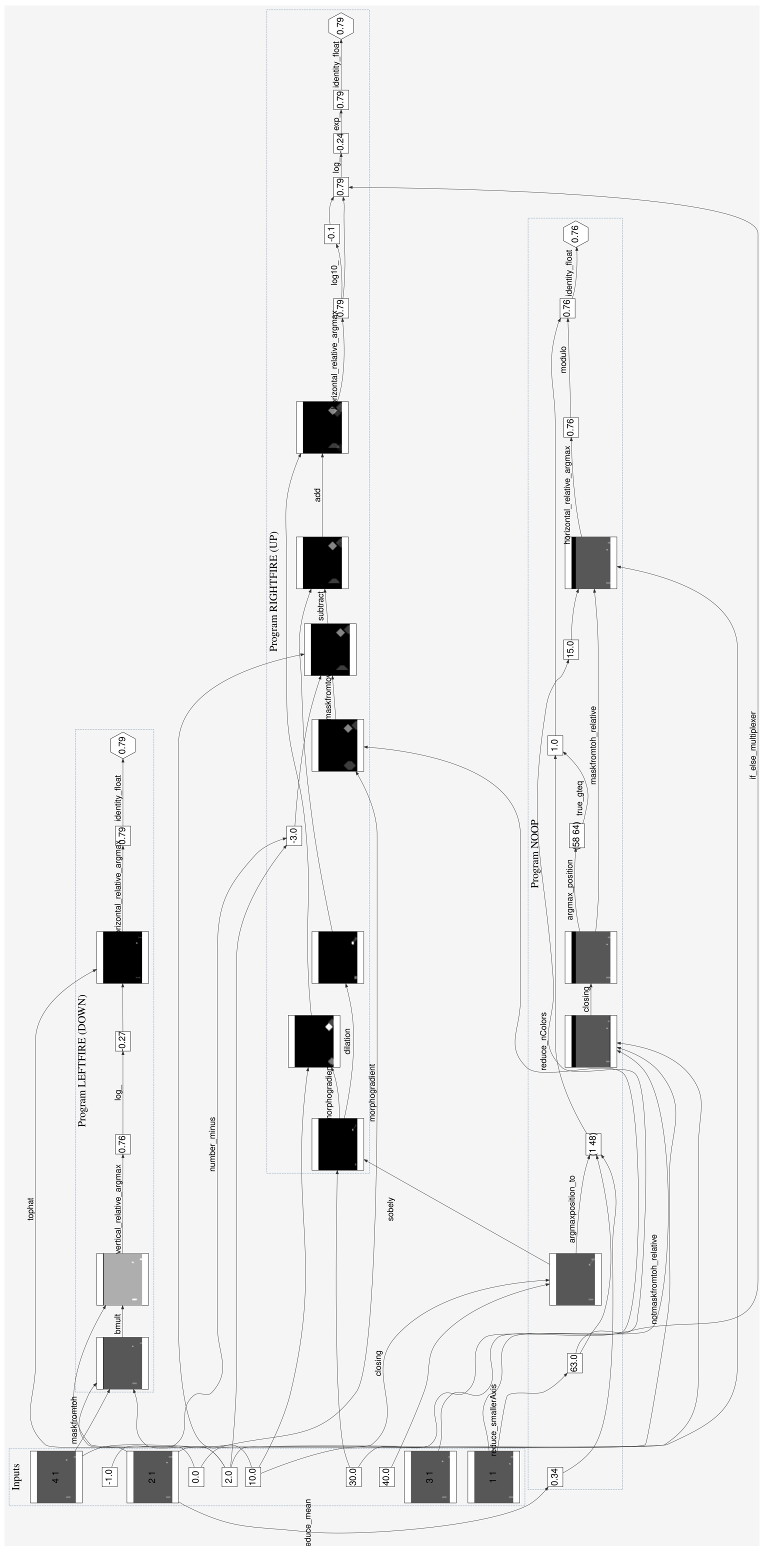


Figure 7: Action RIGHTFIRE (move the paddle up) is chosen.

C.2 Freeway's evolved computational graph

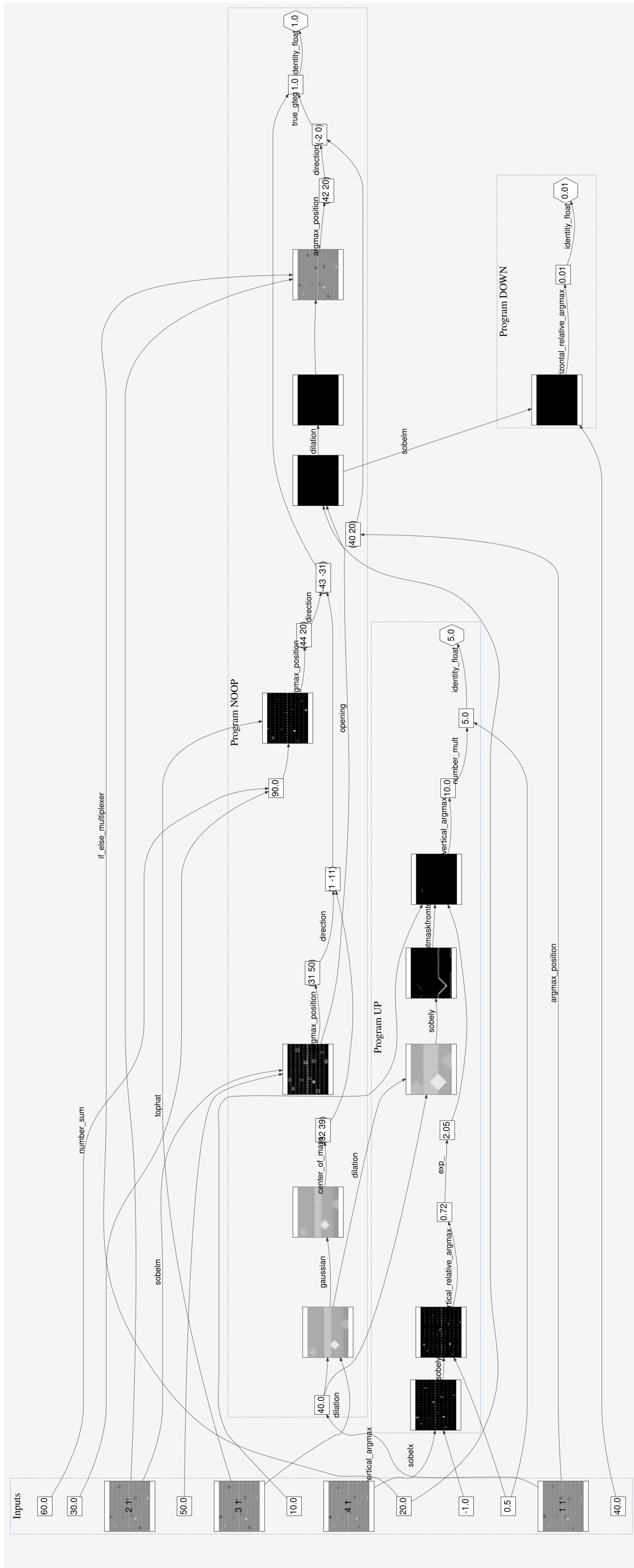


Figure 8: Action UP is chosen.

C.3 Bowling's evolved computational graph

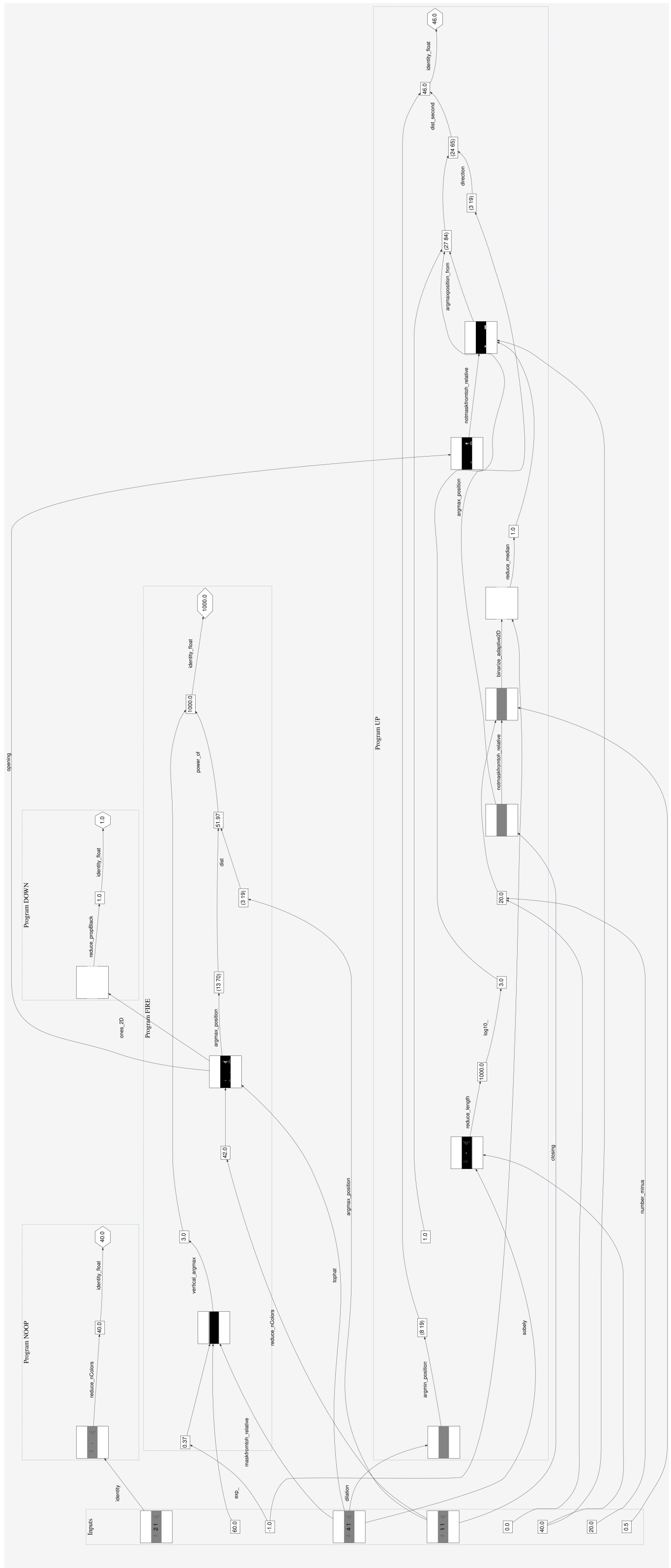


Figure 9: Action `FIRE` is chosen.