Deep Reinforcement Learning Agents are not even close to Human Intelligence

Quentin Delfosse[1,2,†], **Jannis Blüml**[1,3,†], **Fabian Tatai**[4,5], **Théo Vincent**[1,6], **Bjarne Gregori**[1], **Elisabeth Dillies**[7], **Jan Peters**[1,3,4,6], **Constantin Rothkopf** [1,3,4,5], **Kristian Kersting**[1,3,4,6]
{quentin.delfosse,jannis.blueml}@tu-darmstadt.de

[1]**Department of Computer Science, Technical University Darmstadt, Germany**
[2]**National Research Center for Applied Cybersecurity (ATHENE), Germany**
[3]**Hessian Center for Artificial Intelligence (hessian.AI)**
[4]**Centre for Cognitive Science, Darmstadt**
[5]**Institute of Psychology, Technical University Darmstadt, Germany**
[6]**German Research Center for Artificial Intelligence (DFKI)**
[7]**Sorbonne Université, Paris, France**

[†] equal contribution

Abstract

Deep reinforcement learning agents achieve impressive results in a wide variety of tasks, but they lack zero-shot adaptation capabilities. While most robustness evaluations focus on tasks complexifications, for which human also struggle to maintain performances, no evaluation has been performed on tasks simplifications. To tackle this issue, we introduce HackAtari, a set of task variations of the Arcade Learning Environments. We use it to demonstrate that, contrary to humans, RL agents systematically exhibit huge performance drops on simpler versions of their training tasks, uncovering agents' consistent reliance on shortcuts. Our analysis across multiple algorithms and architectures highlights the persistent gap between RL agents and human behavioral intelligence, underscoring the need for new benchmarks and methodologies that enforce systematic generalization testing. It demonstrates the need to integrate more human inductive bias to achieve truly intelligent agents.

1 Introduction

Deep reinforcement learning (RL) has become a key technique for training agents to solve relational reasoning tasks directly from high-dimensional sensory inputs (Zambaldi et al., 2018). In these tasks, agents must identify distinct entities, infer their relationships, and model their dynamics to derive effective decision-making policies. The Arcade Learning Environment (ALE) (Bellemare et al., 2013) is the most widely used benchmark for evaluating RL algorithms in this setting, offering a diverse collection of Atari 2600 games that span a broad range of perceptual and strategic challenges, including spatial reasoning, long-term planning, and real-time reaction. In their seminal work, Mnih et al. (2015) introduced the first deep RL method able to solve many ALE games and claimed that when "*agents are confronted with a difficult task: they must derive efficient representations of the environment from high-dimensional sensory inputs, and use these to generalize past experience to new situations*". Inspired by the brain's visual processing mechanisms, they incorporated convolutional neural networks to extract spatial features from raw pixels. Deep RL algorithms have then been improved and finally achieved *superhuman performance* in all Atari games (Badia et al., 2020a;b).

However, the capacity of these agents to "*generalize experiences from past situations to novel scenarios*", core to relational reasoning, is rarely assessed by RL practitioners. Traditional benchmarks typically involve training and evaluating agents within identical environments, thereby masking their reliance on shortcuts and spurious correlations. This is a particularly troubling issue, given recent evidence that RL agents often exploit superficial shortcuts rather than learning robust, causal strategies (Ilyas et al., 2019; Geirhos et al., 2020; Chan et al., 2020; Koch et al., 2021; Delfosse et al., 2024b).

This reliance on shortcuts has lately been uncovered in the simplest Atari Pong game (depicted in Figure 4). In this game, the agent's enemy follows a deterministic behavior based on the position of the ball. Delfosse et al. (2024b) have thus exposed that deep and symbolic RL agents learn to rely on the enemy's position to catch and return the ball optimally. Hiding the enemy or altering its behavior thus leads to a performance drop. While the inability of DQN agents to zero-shot generalize to complexified versions of their training environment has been showcased (Farebrother et al., 2018), there has not yet been any systematic evaluation of the potential misalignments of RL agents. To assess whether RL agents learn the right behaviors, we evaluate them on simplified task variations that humans can easily adapt to - such as color changes or gameplay simplifications - ensuring that performance drops reveal flawed reasoning rather than increased task difficulty.

In this article, we demonstrate that current deep and symbolic RL agents systematically fail to solve simplified variants of their training tasks, supporting that **human-level performances in training settings does not imply human-like reasoning capabilities**. We present a systematic investigation of generalization failures in RL using controlled environment modifications, most of which simplify the original tasks. We release HackAtari[1], a suite of environment variations for the most widely used RL benchmark: the Arcade Learning Environments (Bellemare et al., 2013). We argue that RL agents should be evaluated on held-out task variations – common practice in machine learning – to enable meaningful comparisons with human intelligence, particularly in relational reasoning domains. While our current focus is empirical, our findings align closely with established literature on shortcut learning and spurious correlations in representation learning (e.g., Geirhos et al. (2020); Ilyas et al. (2019). Future work may build on this to develop formal models of such behavior.

Our main contributions can be summarized as follows:

(i) We show that deep RL agents fail to generalize to task simplifications and consistently rely on shortcut learning – selecting the right actions for the wrong reasons, regardless of the algorithm.
(ii) We show that symbolic/object-centric agents exhibit better adaptation capabilities, but are not yet able to match humans' one.
(iii) We conduct a user study to assert users' abilities to maintain their performances within many of these variations, and thus provide performance references for these simplifications.
(iv) We open-source a wide range of variations for the Arcade Learning Environments, opening up new possibilities for designing and testing RL agents with supposed human-like intelligence.

In the following, we present the different RL agents' architectures and algorithms used in this article. We then introduce our HackAtari framework and use it to evaluate RL agents' generalization ability.

2 The Quest for General RL algorithms

The objective of relational reasoning RL is to develop general agents with human-comparable problem-solving skills. This goal encompasses two key subgoals: (1) designing a single, versatile algorithm that can be applied across a wide range of tasks, and (2) enabling trained agents to generalize effectively to variations of their training tasks. This work focuses on the second aspect of generality. We here evaluate two prominent classes of agents: deep RL agents and neurosymbolic agents. Task agnostic deep agents often struggle with overfitting and generalization to task variations (Farebrother et al., 2018). In contrast, neurosymbolic agents introduce inductive biases by representing environments in terms of objects and their interactions, supporting abstract and transferable representations.

[1]HackAtari available at `https://github.com/k4ntz/HackAtari`.

Task agnostic deep RL agents have been introduced with the Deep Q-Networks (DQN) (Mnih et al., 2015) algorithm, demonstrating that a single convolutional architecture could learn to play a variety of Atari games directly from pixel inputs, establishing a foundation for general-purpose deep RL. Subsequent improvements extended DQN along several dimensions: C51 (Bellemare et al., 2017) returns distributions to capture uncertainty better. M-DQN (Vieillard et al., 2020) introduces entropy-regularized rewards to improve training stability. Rainbow (Hessel et al., 2018) combines several of these enhancements into a widely used baseline. i-DQN (Vincent et al., 2025) enables multiple consecutive Bellman updates through a sequence of learned action-value functions. On-policy methods like PPO (Schulman et al., 2017) and distributed frameworks such as IMPALA (Espeholt et al., 2018) offer scalable alternatives with strong empirical performance. More recently, model-based agents like Dreamer (Hafner et al., 2020) learn latent dynamics models and perform planning in imagination, yielding greater sample efficiency and improved extrapolation in high-dimensional visual environments. The ease of adaptation of deep RL algorithms to novel tasks, requiring no expert knowledge injection, explains their widespread adoption in relational RL (Shaheen et al., 2025).

Object-centric and neurosymbolic agents incorporate stronger inductive biases through structured representations but require specific domain adaption. Early work on object-oriented MDPs (Diuk et al., 2008) introduced state decompositions based on object attributes and relations, enabling policies to reason abstractly over entities rather than raw pixel arrays. Recent methods build on this idea to improve both interpretability and generalization. Unsupervised object extraction methods (Lin et al., 2020; Delfosse et al., 2023b) have allowed RL practitioners to develop symbolic policies. Jiang & Luo (2019) and Delfosse et al. (2023a) represents policies as sets of logic rules over symbolic object states, showing transfer to environments with varying object types and counts. Shindo et al. (2025) leverages large language models to autonomously generate predicates from visual inputs and integrate symbolic reasoning with deep learning in a hybrid policy architecture. Other approaches constrain object-centric policy representations to interpretable structures such as polynomials (Luo et al., 2024), decision trees (Bastani et al., 2018; Delfosse et al., 2024b; Marton et al., 2025), or programs (Cao et al., 2022; Liu et al., 2023; Kohler et al., 2024). These methods are usually also tested in tasks that require extrapolating to unseen object configurations or recombining learned behaviors across novel layouts. Finally, object-centric deep agents have been developed. They make use of object-centric masking (Davidson & Lake, 2020; Blüml et al., 2025), removing background information to force attention on dynamic entities without relying on explicit symbolic conversion.

3 Creating ALE variations

Our main objective is to show that relational reasoning RL agents consistently learn misaligned policies, *i.e.* that they rely on shortcuts. To explicitly exhibit opaque deep RL agents' reliance on shortcuts, we need to evaluate them on altered version of the environment, that allow to test that they select the right actions for the right reason. Evaluations on tasks variations have been conducted in the past on more complex environments (Farebrother et al., 2018). However, the eventual performance drops do not imply the agent's misalignment, as it could stem from bad adaptation capabilities. To assert that relational reasoning agents learn incorrect policies, and have not mastered the desired set of skills necessary to solve the game, we need to test them on tasks simplifications, *i.e.* tasks that will lead humans to *increase or maintain* their overall performances.

To do so, we introduce HackAtari, a set of variations (mostly simplifications) of ALE. A complete description of HackAtari and many examples are available in Appendix A and H. In the next section, we use HackAtari to show that RL agents learn policies incapable of adapting tasks simplifications.

4 Experimental Evaluation

Let us now empirically evaluate several aspects of deep and object-centric RL agents' generalization capabilities, focusing specifically on their ability to handle simplified variations of their original training tasks. We here aim to answer the following research questions:

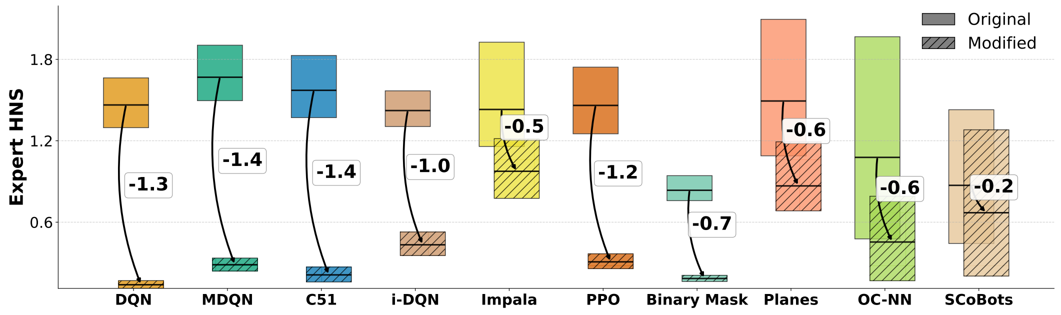


Figure 1: **Deep and symbolic RL agents performances drop on HackAtari variations**, illustrated by the IQM (following reliable (Agarwal et al., 2021)) over the human normalized scores (HNS) of various RL agents on a total set of 32 task variations (over 17 games). IQMs are computed over 3 seeded trained agents (30 evaluations each). Expert-human scores are borrowed from Badia et al. (2020a). Performance in the original environment is plotted filled, while the performance in the modified environment is plotted hatched. Raw IQM scores (with CIs) for each agent on each game (original and variations) and extended results are provided in Appendix E.1 and E.2.

(Q1) Do RL agents' performance drop on HackAtari tasks variations?
(Q2) Can human easily adapt to such tasks variations?
(Q3) Are deep agents systematically learning shortcuts on relational reasoning tasks?
(Q4) Do human inductive biases help aligning RL agents?

Experimental Setup. We evaluate a diverse set of RL agents, including standard value-based methods: DQN (Mnih et al., 2015), C51 (Bellemare et al., 2017) and i-DQN (Vincent et al., 2025), policy-gradient methods: PPO (Schulman et al., 2017) and IMPALA (Espeholt et al., 2018)). All agents use the nature-CNN neural network, except for IMPALA, whose authors have come up with a more complex neural network using neural architecture search. We also evaluate several object-centric variants: OCCAM (Blüml et al., 2025), Object-centric agents using neural networks (Delfosse et al., 2024a), SCoBots (Delfosse et al., 2024b). Most tested agents have been collected from their official repository, with only deep PPO and OCNN variants trained in-house using the CleanRL framework (Huang et al., 2022). Detailed training settings, architectures, and source references for each agent are provided in Appendix C. As commonly done, all agents are trained for 200 million frames on the Atari environments (v5) using a fixed frameskip of 4 and a repeat action probability of 0.25, following established evaluation protocols (Machado et al., 2018). A complete hyperparameters list can be found in the extended experimental setup (*cf.* Appendix E). To assess generalization, we evaluate each agent HackAtari controlled environment variations, with some described in Section 3. The evaluation is conducted over 30 episodes per game per agent, using at least 3 different evaluation seeds per agent. This resource-limited evaluation suffices to assert the consistent performance drops of the agents (as shown by confidence intervals (CI) in Appendix E.1). For metrics, we report Expert-Humans Normalized Score (E-HNS), with expert scores borrowed from Badia et al. (2020a), and the random scores evaluated in house both on the original and the task variations (*cf.* Table E.3), following Agarwal et al. (2021), as well as Performance Drop, aggregated using the inter-quartile Mean (IQM) and 95% CI. The Performance Drop is positive if the agent's performance increases, null if the performance does not change, and negative if the performance drops. Note that in the computation of these metrics, the average score, obtained by random agents (on the original or on the tasks variation) is ablated. These metrics enable robust comparisons of the ability of agents to generalize to task variations. For further details on the metrics, *cf.* Appendix B.

To evaluate the effect of the game variations on human performance, we conducted an online study with 135 subjects on 15 games and a variation for each (on the Prolific platform). We included English speaking participants from around the world and all age groups (Mean Age: 33, Range:[18-73]).

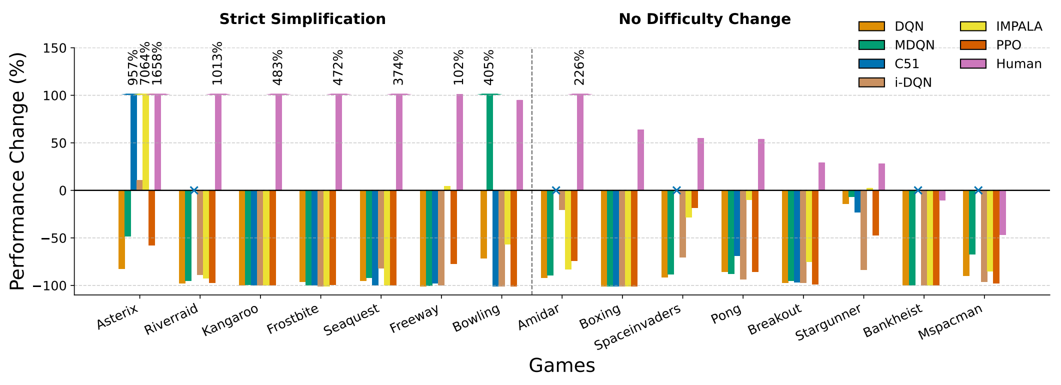


Figure 2: **While humans easily adapt to task simplifications, deep agents' performances drop,** illustrated on 15 ALE games. Non-expert users and deep RL agents are trained and evaluated on the original ALE environment, then presented with a variation of the task. Left: Variations considered as task simplifications by design. Right: Variations for which little or no performance increase is expected. Games for which no C51 agent is publicly available are marked with ×. For the exact performances of humans and deep agents, *cf.* Appendix E.1 and E.3.

The subjects were allowed to train for 10 to 15 minutes on the original game. They were then evaluated for 15 minutes on the original task, and finally for 15 minutes on a HackAtari variation (for the game and variation list, *cf.* Appendix E.3). Each subject was only trained and evaluated on a single game and its variation. Unlike to Badia et al. (2020a), we have not selected professional gamers, to avoid biased evaluations, potentially stemming from such users' ability to perform well on *any* video game task (thus not directly measuring their ability to adapt to the variations), but there overall zero-shot (or no training) performances. For more details on the evaluation protocol and demographics studies, *cf.* Appendix G. Let us now demonstrate that current RL agents consistently learn shortcuts, and fail to adapt to task simplifications.

RL agents' performance drop on HackAtari most tasks variations (Q1). We first evaluate all the available RL algorithms on a subset of 17 ALE games (the ones with publicly available agents; C51 was only available for 12 games), with 1 to 4 variations per game. Figure 1 depicts the expert-human normalized scores, averaged using IQMs, for each algorithm, on the 17 original games and on the 30 modifications. Every RL agent exhibits a significant performance drop over the complete set of tested variations. Remarkably, SCoBots agents exhibit the lowest E-HNS performance drop (20% overall), and IMPALA maintains an average superhuman performance in the game variations. This outlines that changing the architecture or inference paradigm has a bigger impact on the ability of agents to adapt to task variations than varying the algorithm. However, we still need to demonstrate that these agents learn misaligned policies that prevent them from adapting to simplifications, and do not fail on the variations because of potential increased game complexities.

Human adaptation to simplification far exceeds RL agents' ones (Q2). Our claim that RL agents learn misaligned policies is grounded in evaluating them on task *simplifications*—variations where human performance is expected to remain stable or improve. To validate this, we conducted a user study on 15 games, each with one simplified variation where most agents exhibit performance drops. We recruited 134 participants (at least 8 per game), who were trained and evaluated on both the original ALE task and its variation. We measured the change in performance for each participant and agent separately. As shown in Figure 2, human performance improves notably on 11 games, slightly increases on 2, and only drops on Bankheist and MsPacman. In Bankheist, new chasing police cars introduce risk but higher rewards, leading to mixed outcomes depending on player strategy (*cf.* Appendix G.6). For MsPacman, performance drops may stem from participant fatigue or reliance on overfit strategies that don't transfer across altered mazes. Still, human drops remain well below those of all evaluated agents. Together with Section 3 and Appendix H, these results confirm that the 15 variations used are valid simplifications from a human perspective.

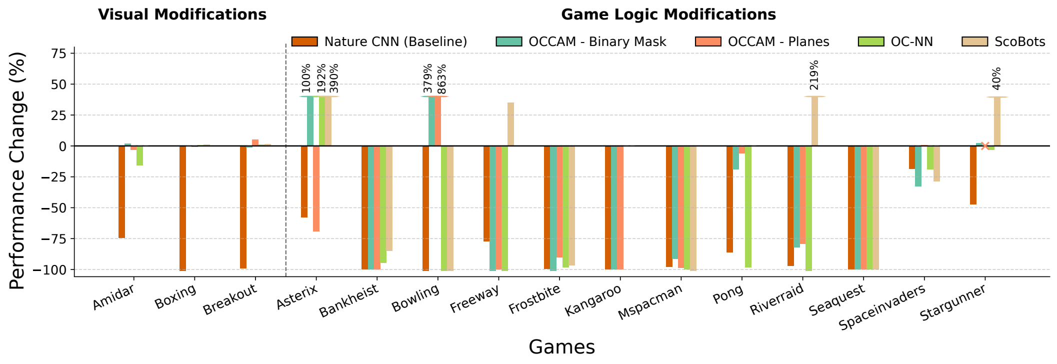


Figure 3: **Object-centric RL agents also fail to adapt to simplified environments.** Different object-centric approaches (all using PPO) are here compared to the classical CNN baseline on the same 15 variations (as Figure 2). Visual perturbations (*e.g.* color change, left) have a very limited impact on the symbolic agents, while most gameplay modifications (left) still cannot be solved by these agents. Extended results are available in Appendix E.2.

Deep agents fail to adapt to task simplifications (Q3). Let us now investigate whether deep agents can adapt to task simplifications. We evaluate several widely-used algorithms, including DQN, PPO, C51, IMPALA, i-DQN, and MQN, across many original Atari environments and some simplified variations included in HackAtari. Figure 2 shows that, contrary to humans, all agents exhibit severe performance drops in most of the task variations. Even IMPALA's performances drop by more than 50% on 10 out of 15 games. All other algorithms exhibit even higher performance drops (also confirmed by Figure 1). Deep RL agents can thus not maintain their performances on most task simplifications, demonstrating their inability to learn meaningful, aligned policies.

The object-centric inductive bias is not enough (Q4). RL agents relying on more human inductive biases could help narrow the gap between deep agents and humans. The most common bias is object-centricity, with agents that first extract the depicted objects and their properties (such as position, color, size) from the pixel states. Such representations have been shown to improve transferability and interpretability in structured environments (Shindo et al., 2025). We thus evaluate whether introducing object-centric representations enhance the agents' ability to generalize to simplifications. Figure 3 shows Performance Change for object-centric agents. As these agents are all trained using PPO, we also provide also include deep (*i.e.* using Nature-CNN) PPO baseline. As expected, these agents are insensitive to visual perturbations (*e.g.* color changes). Notably, decision tree based SCoBots agents have limited performance drops (<30%), on 9 out of 15 games. However, even on most tasks with gameplay alterations, all object-centric agents exhibit high performance drops. Overall, even if these symbolic architectures generally adapt better than the deep pixel-based PPO baseline, they still exhibit notable performance drops. These findings suggest that while object-centricity helps, it alone is insufficient for robust generalization to simpler tasks.

Overall, our analysis demonstrates the global incapacity of both deep and symbolic agents to adapt to task simplifications, as they often completely overfit to their training tasks. **RL agents do not** "*derive efficient representations allowing to generalize past experience to new situations*" (Mnih et al., 2015).

5 Related Work

Generalization Benchmarks and Failures. Traditional generalization evaluations often focus on performance in fixed environments, leading to overfitting and a poor understanding of agents' robustness. Extensions to the Arcade Learning Environment (ALE) such as sticky actions (Machado et al., 2018), unseen modes (Cobbe et al., 2019) and action noise (Koch et al., 2021) introduced limited variability to probe robustness. Zhang et al. (2018) also focus on perturbations on the observation space of the ALE by evaluating trained agents against abrupt background changes.

More ambitious frameworks like CoinRun (Cobbe et al., 2019) and Procgen (Cobbe et al., 2020) procedurally generate diverse levels, but this does not prevent misalignment di Langosco et al. (2022). Crafter (Hafner, 2022) and MineRL (Guss et al., 2019; Milani et al., 2020) further emphasize compositionality and long-term credit assignment. However, these benchmarks increase perceptual or structural complexity, making it difficult to disentangle poor adaptation from underlying misalignment. In contrast, our work mainly introduces *simplifications* that preserve task semantics while reducing difficulty, revealing performance failures even in settings where humans readily adapt.

Misalignment and Shortcut Learning. We show that RL agents often succeed by exploiting spurious correlations or shallow heuristics, rather than learning task-aligned policies. This phenomenon, known as shortcut learning (Geirhos et al., 2020), has been documented in supervised vision (Ilyas et al., 2019; Stammer et al., 2021) and increasingly in RL (Zhang et al., 2018; Cobbe et al., 2020; Koch et al., 2021; Delfosse et al., 2024b). Recent interpretability efforts, such as saliency maps and attention overlays, suggest that agents attend to task-relevant regions (Greydanus et al., 2018), yet fail to capture the latent reasoning process. In Pong, for instance, agents may appear to track the ball but instead use opponent behavior as a proxy (Delfosse et al., 2024b). While robustness techniques such as adversarial regularization (Pinto et al., 2017) and domain randomization (Tobin et al., 2017) offer partial defenses, they do not prevent agents from failing when irrelevant cues are removed. By evaluating agents on simplified tasks, where human performance is stable or improves, we isolate misalignment not as failure to adapt, but as failure to *learn to select actions for the right reasons*.

6 Towards RL policies aligned with the true task goals

Evaluating agents' robustness through simplifications to better reflect human-like reasoning. Robust reinforcement learning is often framed as worst-case optimization (Tamar et al., 2014; Moos et al., 2022) or adversarial training (Pinto et al., 2017), with recent work addressing observation perturbations through network sparsity (Grooten et al., 2023) or Lipschitz constraints (Barbara et al., 2024). However, RL agents often fail on simplified versions of their training tasks – settings where humans adapt effortlessly. This reveals a reliance on superficial cues rather than robust relational reasoning. We argue that testing agents on logically simpler variations is essential for evaluating true generalization. HackAtari allows for testing whether agents truly understand the task, challenging the common assumption — reflected in metrics like the Human Normalized Score — that achieving human-level performance implies human-like reasoning. We advocate for benchmarks that explicitly test relational understanding, as the ones appearing in supervised learning setting (Helff et al., 2025; Wüst et al., 2025) and for integrating human inductive biases to align RL agents with task structure.

Incorporating human inductive bias is crucial for the rise of aligned RL agents. Our work demonstrates that relying on extensive compute, large number of parameters and long training does not guarantee that deep RL agents learn coherent representations of the tasks, aligned with the intended task goals. While the dopaminergic system in humans partly resembles model-free reinforcement learning (Schultz et al., 1997), humans make use of use rich model-based representations of their environment (Tenenbaum et al., 2011; Daw et al., 2011). Several studies provide guidance how these findings from cognitive science about human intelligence should be incorporated in intelligent agents (Lake et al., 2017; Zhu et al., 2020). Besides using structured representation and planning in navigation (Kessler et al., 2024) humans have also been shown to have coherent but noisy models of their physical environment (Kubricht et al., 2017; Battaglia et al., 2013), aiding with inference over object properties (Neupärtl et al., 2020), learning tool use (Allen et al., 2020), and choosing actions (Tatai et al., 2025). Further, humans also decompose complex tasks into a sequence of high-level actions, and learn skills that correspond to the different sub-goals of such tasks. Hierarchical RL has been shown to improve adaptation to task variations by learning reusable sub-policies or skill hierarchies (Bacon et al., 2017; Hausman et al., 2018; Li et al., 2020; Cannon & Şimşek, 2025). Humans also model causal object interaction (Michotte, 1963) and can employ abstract and qualitative reasoning about physical situations (Forbus, 1988).

Latest approaches try to incorporate causal world models within RL agents (Yang et al., 2025; Dillies et al., 2025) applicable to multi-objective tasks (Bhatija et al., 2025). Additionally, humans also represent the beliefs and desires of other agents (Dennett, 1989), mathematically formalizable as inverse reinforcement learning (IRL) (Baker et al., 2009). IRL can infer uncertainties about others' behaviors and the consequential reward signals (Dimitrakakis & Rothkopf, 2011; Straub et al., 2023), relevant in *e.g.* Kangaroo, where agents prioritize enemy-killing over child-rescue, showcasing misalignment between RL agents' and human goals. Incorporating the common reasoning abilities of language models to extract reward signals from object-centric states could help tackle this problem (Kaufmann et al., 2024).

7 Limitations and Future Work

While HackAtari offers valuable insights into RL generalization, it is limited by its reliance on ALE, which restricts evaluation to visually stylized environments that may not transfer to more complex or realistic domains. As the benchmark gains popularity, there is also a risk that the agents will overfit to its specific perturbations, undermining its diagnostic value, even if we believe that with enough variations for a given tasks, agents able to solve all variations while learning on the original tasks are likely to be aligned with the primary desired task objectives. Finally, the absence of standardized difficulty scaling across variations can make it difficult to interpret the significance of observed performance drops. We hope that overall, this analysis and the release of the HackAtari variations will motivate RL practitioners to develop agents can maintain performance on task simplifications, thus learning correctly aligned policies.

We believe that systematic evaluations should also be run on held-out test sets also for other, potentially more complex RL benchmarks. To obtain correctly aligned agents, IMPALA stands out for maintaining superhuman performance under task variations, likely due to its off-policy V-trace correction, distributed data collection, and smoother policy updates. Understanding how these elements support generalization may guide the development of more task-aligned deep RL agents. While object-centricity offers a practical inductive bias to narrow the gap between human and artificial agents, it captures only a subset of human adaptability. Broader inductive mechanisms – such as causal reasoning, goal-directed behavior, and abstract task decomposition – play a key role in human generalization. Incorporating these additional biases, for instance via causal modeling or symbolic planning, may further enhance agents' ability to learn aligned, human-like policies.

8 Conclusion

While Mnih et al. (2015) emphasized that "*reinforcement learning agents must learn efficient representations from high-dimensional sensory inputs and generalize from past experience to new situations*", our results demonstrate that current deep and symbolic RL agents largely fail to meet this objective. To investigate this issue systematically, we introduce HackAtari, an open-source benchmark comprising over 200 task variations based on the Arcade Learning Environment, the most widely used evaluation suite in deep RL. HackAtari is designed to go beyond in-distribution evaluation by allowing researchers to test agents on slight but targeted modifications of familiar tasks. These variations, such as changes in color schemes or simplified game dynamics, are typically trivial for humans to adapt to but often reveal the brittleness of learned policies. We hope that HackAtari will serve as a diagnostic tool for evaluating whether RL agents have truly internalized the relational structure of their environments and whether they select actions for the right, generalizable reasons.

Finally, Silver et al. (2021) have hypothesized that *reward is enough*, but our results suggest that without the right biases and evaluations, reward alone is not enough to build agents that truly understand.

Acknowledgments

We would like to express our sincere gratitude to our students for their dedication and feedback as well as all people participating in our study.

This research work has been primarily funded by the German Federal Ministry of Education and Research, the Hessian Ministry of Higher Education, Research, Science and the Arts (HMWK) within their joint support of the National Research Center for Applied Cybersecurity ATHENE as well as the Excellence Program of the Hessian Ministry of Higher Education through their cluster projects within the Hessian Center for AI (hessian.AI) "The Third Wave of Artificial Intelligence - 3AI" and "The Adaptive Mind". Further, we acknowledge support and funding by the German Research Center for AI (DFKI).

References

Agarwal, R., Schwarzer, M., Castro, P. S., Courville, A. C., and Bellemare, M. Deep reinforcement learning at the edge of the statistical precipice. *Advances in Neural Information Processing Systems*, 2021.

Allen, K. R., Smith, K. A., and Tenenbaum, J. B. Rapid trial-and-error learning with simulation supports flexible tool use and physical reasoning. *Proceedings of the National Academy of Sciences*, 2020.

Bacon, P., Harb, J., and Precup, D. The option-critic architecture. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

Badia, A. P., Piot, B., Kapturowski, S., Sprechmann, P., Vitvitskyi, A., Guo, Z. D., and Blundell, C. Agent57: Outperforming the atari human benchmark. In *Proceedings of the 37th International Conference on Machine Learning,*, 2020a.

Badia, A. P., Sprechmann, P., Vitvitskyi, A., Guo, Z. D., Piot, B., Kapturowski, S., Tieleman, O., Arjovsky, M., Pritzel, A., Bolt, A., and Blundell, C. Never give up: Learning directed exploration strategies. In *8th International Conference on Learning Representations*, 2020b.

Baker, C. L., Saxe, R., and Tenenbaum, J. B. Action understanding as inverse planning. *Cognition*, 2009.

Barbara, N. H., Wang, R., and Manchester, I. On robust reinforcement learning with lipschitz-bounded policy networks. In *ICML Workshop: Foundations of Reinforcement Learning and Control–Connections and Perspectives*, 2024.

Bastani, O., Pu, Y., and Solar-Lezama, A. Verifiable reinforcement learning via policy extraction. In *Advances in Neural Information Processing Systems*, 2018.

Battaglia, P. W., Hamrick, J. B., and Tenenbaum, J. B. Simulation as an engine of physical scene understanding. *Proceedings of the National Academy of Sciences*, 2013.

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 2013.

Bellemare, M. G., Dabney, W., and Munos, R. A distributional perspective on reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning*, 2017.

Bhatija, S., Zuercher, P.-D., Thumm, J., and Bohné, T. Multi-objective causal bayesian optimization. *arXiv*, 2025.

Blüml, J., Derstroff, C., Gregori, B., Dillies, E., Delfosse, Q., and Kersting, K. Deep reinforcement learning via object-centric attention. *arXiv*, 2025.

Cannon, T. P. and Şimşek, Ö. Accelerating task generalisation with multi-level hierarchical options. In *The Thirteenth International Conference on Learning Representations*, 2025.

Cao, Y., Li, Z., Yang, T., Zhang, H., Zheng, Y., Li, Y., Hao, J., and Liu, Y. Galois: boosting deep reinforcement learning via generalizable logic synthesis. *Advances in Neural Information Processing Systems*, 2022.

Chan, S. C. Y., Fishman, S., Korattikara, A., Canny, J. F., and Guadarrama, S. Measuring the reliability of reinforcement learning algorithms. In *8th International Conference on Learning Representations*, 2020.

Cobbe, K., Klimov, O., Hesse, C., Kim, T., and Schulman, J. Quantifying generalization in reinforcement learning. In *Proceedings of the 36th International Conference on Machine Learning*, 2019.

Cobbe, K., Hesse, C., Hilton, J., and Schulman, J. Leveraging procedural generation to benchmark reinforcement learning. In *International conference on machine learning*, 2020.

Davidson, G. and Lake, B. M. Investigating simple object representations in model-free deep reinforcement learning. In *Proceedings of the 42th Annual Meeting of the Cognitive Science Society*, 2020.

Daw, N. D., Gershman, S. J., Seymour, B., Dayan, P., and Dolan, R. J. Model-based influences on humans' choices and striatal prediction errors. *Neuron*, 2011.

Delfosse, Q., Shindo, H., Dhami, D. S., and Kersting, K. Interpretable and explainable logical policies via neurally guided symbolic abstraction. *Advances in Neural Information Processing (NeurIPS)*, 2023a.

Delfosse, Q., Stammer, W., Rothenbacher, T., Vittal, D., and Kersting, K. Boosting object representation learning via motion and object continuity. In *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML)*, 2023b.

Delfosse, Q., Blüml, J., Gregori, B., Sztwiertnia, S., and Kersting, K. OCAtari: Object-centric Atari 2600 reinforcement learning environments. *Reinforcement Learning Journal*, 2024a.

Delfosse, Q., Sztwiertnia, S., Stammer, W., Rothermel, M., and Kersting, K. Interpretable concept bottlenecks to align reinforcement learning agents. *Advances in Neural Information Processing Systems*, 2024b.

Dennett, D. C. *The intentional stance*. 1989.

di Langosco, L. L., Koch, J., Sharkey, L. D., Pfau, J., and Krueger, D. Goal misgeneralization in deep reinforcement learning. In *International Conference on Machine Learning*, 2022.

Dillies, E., Delfosse, Q., Blüml, J., Emunds, R., Busch, F. P., and Kersting, K. Better decisions through the right causal world model. *arXiv*, 2025.

Dimitrakakis, C. and Rothkopf, C. A. Bayesian multitask inverse reinforcement learning. In *European workshop on reinforcement learning*, 2011.

Diuk, C., Cohen, A., and Littman, M. L. An object-oriented representation for efficient reinforcement learning. In *Proceedings of the 25th international conference on Machine learning*, 2008.

Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., Doron, Y., Firoiu, V., Harley, T., Dunning, I., Legg, S., and Kavukcuoglu, K. IMPALA: scalable distributed deep-rl with importance weighted actor-learner architectures. In *Proceedings of the 35th International Conference on Machine Learning,*, 2018.

Farebrother, J., Machado, M. C., and Bowling, M. Generalization and regularization in DQN. 2018.

Forbus, K. D. Qualitative physics: Past, present, and future. In *Exploring artificial intelligence*. 1988.

Geirhos, R., Jacobsen, J., Michaelis, C., Zemel, R. S., Brendel, W., Bethge, M., and Wichmann, F. A. Shortcut learning in deep neural networks. *Nat. Mach. Intell.*, 2020.

Gogianu, F., Berariu, T., Bușoniu, L., and Burceanu, E. Atari agents, 2022.

Greydanus, S., Koul, A., Dodge, J., and Fern, A. Visualizing and understanding atari agents, 2018.

Grooten, B., Sokar, G., Dohare, S., Mocanu, E., Taylor, M. E., Pechenizkiy, M., and Mocanu, D. C. Automatic noise filtering with dynamic sparse training in deep reinforcement learning. In *International Conference on Autonomous Agents and Multiagent Systems*, 2023.

Guss, W. H., Codel, C., Hofmann, K., Houghton, B., Kuno, N., Milani, S., Mohanty, S., Liebana, D. P., Salakhutdinov, R., Topin, N., et al. The minerl competition on sample efficient reinforcement learning using human priors. *arXiv*, 2019.

Hafner, D. Benchmarking the spectrum of agent capabilities. In *International Conference on Learning Representations*, 2022.

Hafner, D., Lillicrap, T. P., Ba, J., and Norouzi, M. Dream to control: Learning behaviors by latent imagination. In *8th International Conference on Learning Representations*, 2020.

Hausman, K., Springenberg, J. T., Wang, Z., Heess, N., and Riedmiller, M. Learning an embedding space for transferable robot skills. In *International Conference on Learning Representations*, 2018.

Helff, L., Stammer, W., Shindo, H., Dhami, D. S., and Kersting, K. V-lol: A diagnostic dataset for visual logical learning. *Journal of Data-centric Machine Learning Research*, 2025.

Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., and Silver, D. Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the AAAI conference on artificial intelligence*, 2018.

Huang, S., Dossa, R. F. J., Ye, C., Braga, J., Chakraborty, D., Mehta, K., and Araújo, J. G. Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research*, 2022.

Ilyas, A., Santurkar, S., Tsipras, D., Engstrom, L., Tran, B., and Madry, A. Adversarial examples are not bugs, they are features. *Advances in neural information processing systems*, 2019.

Jiang, Z. and Luo, S. Neural logic reinforcement learning. In *International Conference on Machine Learning*, 2019.

Kaufmann, T., Blüml, J., Wüst, A., Delfosse, Q., Kersting, K., and Hüllermeier, E. Ocalm: Object-centric assessment with language models. *arXiv*, 2024.

Kessler, F., Frankenstein, J., and Rothkopf, C. A. Human navigation strategies and their errors result from dynamic interactions of spatial uncertainties. *Nature Communications*, 2024.

Koch, J., Langosco, L., Pfau, J., Le, J., and Sharkey, L. Objective robustness in deep reinforcement learning. *arXiv*, 2021.

Kohler, H., Delfosse, Q., Akrour, R., Kersting, K., and Preux, P. Interpretable and editable programmatic tree policies for reinforcement learning. *arXiv*, 2024.

Kubricht, J. R., Holyoak, K. J., and Lu, H. Intuitive physics: Current research and controversies. *Trends in cognitive sciences*, 2017.

Lake, B. M., Ullman, T. D., Tenenbaum, J. B., and Gershman, S. J. Building machines that learn and think like people. *Behavioral and brain sciences*, 2017.

Li, A., Florensa, C., Clavera, I., and Abbeel, P. Sub-policy adaptation for hierarchical reinforcement learning. In *International Conference on Learning Representations*, 2020.

Lin, Z., Wu, Y., Peri, S. V., Sun, W., Singh, G., Deng, F., Jiang, J., and Ahn, S. SPACE: unsupervised object-oriented scene representation via spatial attention and decomposition. In *International Conference on Learning Representations*, 2020.

Liu, G.-T., Hu, E.-P., Cheng, P.-J., Lee, H.-Y., and Sun, S.-H. Hierarchical programmatic reinforcement learning via learning to compose programs. In *International Conference on Machine Learning*, 2023.

Luo, L., Zhang, G., Xu, H., Yang, Y., Fang, C., and Li, Q. Insight: End-to-end neuro-symbolic visual reinforcement learning with language explanations. *arXiv*, 2024.

Machado, M. C., Bellemare, M. G., Talvitie, E., Veness, J., Hausknecht, M., and Bowling, M. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, 2018.

Marton, S., Grams, T., Vogt, F., Lüdtke, S., Bartelt, C., and Stuckenschmidt, H. Sympol: Symbolic tree-based on-policy reinforcement learning. In *International Conference on Learning Representations*, 2025.

Michotte, A. The perception of causality. 1963.

Milani, S., Topin, N., Houghton, B., Guss, W. H., Mohanty, S. P., Nakata, K., Vinyals, O., and Kuno, N. S. Retrospective analysis of the 2019 minerl competition on sample efficient reinforcement learning. In *NeurIPS 2019 competition and demonstration track*, 2020.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M. A., Fidjeland, A., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. Human-level control through deep reinforcement learning. *Nature*, 2015.

Moos, J., Hansel, K., Abdulsamad, H., Stark, S., Clever, D., and Peters, J. Robust reinforcement learning: A review of foundations and recent advances. *Machine Learning and Knowledge Extraction*, 2022.

Neupärtl, N., Tatai, F., and Rothkopf, C. A. Intuitive physical reasoning about objects' masses transfers to a visuomotor decision task consistent with newtonian physics. *PLoS Computational Biology*, 2020.

Pinto, L., Davidson, J., Sukthankar, R., and Gupta, A. Robust adversarial reinforcement learning. In *International conference on machine learning*, 2017.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. 2017.

Schultz, W., Dayan, P., and Montague, P. R. A neural substrate of prediction and reward. *Science*, 1997.

Shaheen, A., Badr, A., Abohendy, A., Alsaadawy, H., and Alsayad, N. Reinforcement learning in strategy-based and atari games: A review of google deepminds innovations. *arXiv*, 2025.

Shindo, H., Delfosse, Q., Dhami, D. S., and Kersting, K. Blendrl: A framework for merging symbolic and neural policy learning. In *International Conference on Learning Representations*, 2025.

Silver, D., Singh, S., Precup, D., and Sutton, R. S. Reward is enough. *Artificial intelligence*, 2021.

Stammer, W., Schramowski, P., and Kersting, K. Right for the right concept: Revising neuro-symbolic concepts by interacting with their explanations. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2021.

Straub, D., Schultheis, M., Koeppl, H., and Rothkopf, C. A. Probabilistic inverse optimal control for non-linear partially observable systems disentangles perceptual uncertainty and behavioral costs. *Advances in Neural Information Processing Systems*, 2023.

Tamar, A., Mannor, S., and Xu, H. Scaling up robust mdps using function approximation. In *Proceedings of the 31st International Conference on Machine Learning*, 2014.

Tatai, F., Straub, D., and Rothkopf, C. A. Intuitive sensorimotor decisions under risk take newtonian physics into account. 2025.

Tenenbaum, J. B., Kemp, C., Griffiths, T. L., and Goodman, N. D. How to grow a mind: Statistics, structure, and abstraction. *Science*, 2011.

Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., and Abbeel, P. Domain randomization for transferring deep neural networks from simulation to the real world. In *international conference on intelligent robots and systems*, 2017.

Vieillard, N., Pietquin, O., and Geist, M. Munchausen reinforcement learning. *Advances in Neural Information Processing Systems*, 2020.

Vincent, T., Palenicek, D., Belousov, B., Peters, J., and D'Eramo, C. Iterated q-network: Beyond one-step bellman updates in deep reinforcement learning. *Transactions on Machine Learning Research*, 2025.

Wüst, A., Tobiasch, T., Helff, L., Ibs, I., Stammer, W., Dhami, D. S., Rothkopf, C. A., and Kersting, K. Bongard in wonderland: Visual puzzles that still make ai go mad? *arXiv*, 2025.

Yang, Y., Huang, B., Feng, F., Wang, X., Tu, S., and Xu, L. Towards generalizable reinforcement learning via causality-guided self-adaptive representations. In *International Conference on Learning Representations*, 2025.

Zambaldi, V. F., Raposo, D., Santoro, A., Bapst, V., Li, Y., Babuschkin, I., Tuyls, K., Reichert, D. P., Lillicrap, T. P., Lockhart, E., Shanahan, M., Langston, V., Pascanu, R., Botvinick, M. M., Vinyals, O., and Battaglia, P. W. Relational deep reinforcement learning. *arXiv*, 2018.

Zhang, A., Wu, Y., and Pineau, J. Natural environment benchmarks for reinforcement learning. *arXiv*, 2018.

Zhu, Y., Gao, T., Fan, L., Huang, S., Edmonds, M., Liu, H., Gao, F., Zhang, C., Qi, S., Wu, Y. N., et al. Dark, beyond deep: A paradigm shift to cognitive ai with humanlike common sense. *Engineering*, 2020.

Supplementary Materials

The following content was not necessarily subject to peer review.

Appendix Overview

A HackAtari complete description

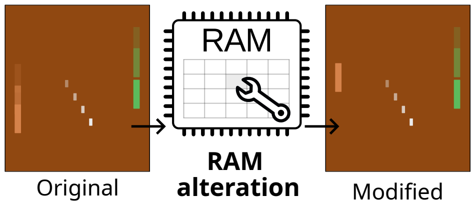


Figure 4: **RAM alteration allows for modified environments**, here exemplified on Pong. Altering specific RAM cells leads to an enemy remaining static after it returned the ball.

We here introduce HackAtari, an extension of the Arcade Learning Environment (ALE), that provides different tasks modifications (mainly simplifications) to evaluate RL agents' *true* relational reasoning abilities. Since the source code of the Atari games used in the ALE is proprietary and not publicly available, direct modifications to game logic or mechanics at the code level are not feasible. As a result, the only practical method to alter or simplify these environments for experimental purposes is through direct manipulation of their Random Access Memory (RAM) states. Thus, modifications are implemented through direct alterations of the game's RAM, allowing diverse controlled perturbations. Specifically, we created visual alterations, such as changing or obscuring object colors, forcing agents to rely less on superficial visual features. Additionally, we adjusted gameplay dynamics, including modifying the speed or presence of objects and enemies, or introducing new mechanics such as gravity effects. This is exemplified in Figure 4 on Pong. In the default game, the brown enemy is programmed to go down if the ball is bellow its paddle and up if the ball is above. There is thus a high correlation between the enemy's and the ball's vertical position, hence, a misalignment opportunity. To create the LazyEnemy variation, we first identify the RAM cell that controls the ball's horizontal speed. When this value is positive — indicating that the ball is moving toward the green agent — we overwrite the enemy's vertical position with its previous value. This makes the enemy remain static whenever the ball

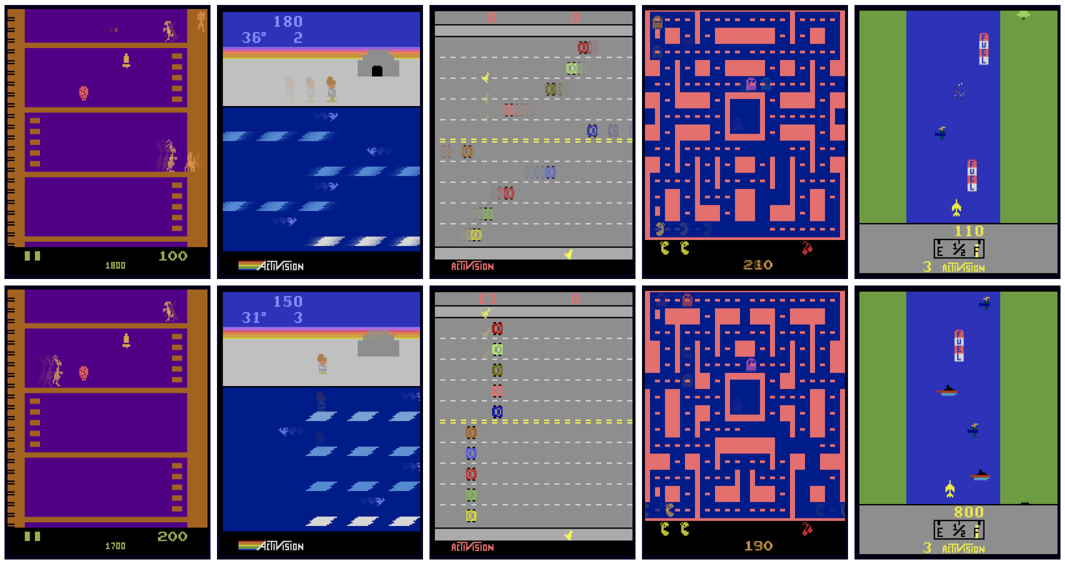


Figure 5: **Examples of HackAtari simple tasks variations.** Top: the original Atari games used to trained RL agents. Bottom: simplifications (*i.e.* variations for which human performances do not drop). These include color changes and gameplay shifts. Superposed frames show the game dynamics. Descriptions of more environments and their variations are provided in Appendix H.

approaches the agent. We can thus evaluate potential RL agents misalignments. Let us now provide further examples of tasks variations included in HackAtari, most of which are illustrated in Figure 5.

NoDanger (Kangaroo). In Kangaroo, the player controls a mother Kangaroo that needs to reach her joey (top left). Monkeys will come to punch her and throw coconuts. To collect points, mother Kangaroo can punch the enemies, collect fruits and reach her baby. In the NoDanger variations, deadly monkeys and coconuts are deactivated. The player can safely navigate to the joey.

StableBlocks (Frostbite). In Frostbite, the player builds an igloo by jumping on moving floating ice blocks while avoiding deadly predators. In the modified task, the ice blocks are aligned and static.

StoppedCars (Freeway). In Freeway, the agent's goal is to have the (left) controlled chicken cross the highway. If the chicken collides with a car, it is sent down and immobilized for a few seconds. In this variation, cars are completely stopped, making crossing trivial.

Maze2 (MsPacman). In MsPacman, Pacman needs to navigate within a maze to consuming all the dots while avoiding the four pursuing ghosts. In this variation, the maze layout is modified.

RestrictedFire (RiverRaid). Players here pilot a fighter jet over a river, aiming to destroy enemy targets while avoiding collisions with riverbanks and obstacles. In this variation, the agent can only shoot in case of unavoidable obstacles. It can collects points by dodging and overcoming the enemies.

ShiftedShields (SpaceInvaders). In SpaceInvaders, we shift the protective shields horizontally by 1 or by 3 pixels to the right. This type of change adjusts the environment in a very minimal way, which is often not even noticeable by humans. It does not significantly alter the core gameplay or difficulty.

A.1 Manipulating the RAM

All environment modifications in HackAtari are implemented by directly altering the Atari emulator's internal Random Access Memory (RAM) values—not the agent's replay buffer or the computer's system memory. The RAM of Atari games encodes key game state variables such as object positions, velocities, scores, and other logic-critical elements. By identifying and modifying specific RAM addresses, we can precisely control various aspects of gameplay—such as freezing enemy movement, shifting object colors, or disabling obstacles—without modifying the underlying source code of the game, which is not publicly available.

This RAM-based manipulation approach enables fine-grained, controlled, and reproducible task variations while preserving the original game logic and dynamics. It introduces negligible computational overhead and scales efficiently across large batches of episodes and variations. To ensure transparency and reproducibility, we publicly release the full implementation of these RAM modifications, along with comprehensive documentation and evaluation scripts, at
https://github.com/k4ntz/HackAtari.

At the time of writing, HackAtari incorporates more than 224 variations in total, spanning over more than 33 games of the original Arcade Learning Environments. We provide illustrations and descriptions of the variations in Appendix H. Most variations used in this paper simplify (*i.e.* should lead humans to at least maintain their performances) the original tasks, which is necessary for showing that RL agents learn misaligned behaviors (using shortcuts). Finally, ALE already incorporates some variations that augment some games' difficulty levels, which are of course integrated in HackAtari. Overall, these capabilities collectively position HackAtari as a valuable resource for developing relational reasoning RL agents that are robust, adaptable, and capable of generalizing their policies beyond their initial training conditions.

B Metrics

Throughout this article, we have used two complementary metrics to evaluate agents in Atari environments: the Interquartile Mean (IQM) over *Human-Normalized Scores (HNS)*, which measures absolute performance of an agent, normalized by the performance of a human. Metrics are aggregated using the `rliable` library (Agarwal et al., 2021) to provide statistically reliable evaluations across diverse environments.

(Expert) Human-Normalized Score (HNS)

To assess agent performance relative to both naive and expert baselines, we report the Human-Normalized Score (HNS), a standard evaluation metric in Atari benchmarks (Mnih et al., 2015; Machado et al., 2018). Let A denote the average score achieved by the agent, H the score of a human expert, and R the score of a random policy. The HNS is defined as:

$$\text{HNS} = \frac{A - R}{|H - R|} \tag{1}$$

An HNS of 1.0 (or 100%) indicates human-level performance, while values above 1.0 suggest superhuman ability. Values near 0 imply the agent performs comparably to a random policy, and negative values indicate performance worse than random. This metric normalizes across games with different score scales and dynamics, enabling fair comparison and aggregation across tasks. HNS is computed using raw episodic returns averaged over multiple seeds. We are using the human and random reference scores from Badia et al. (2020a). Since Badia et al. (2020a) were using "*professional human game testers*", we call this Expert HNS. We report HNS alongside other metrics such as interquartile mean (IQM) and performance change to provide a robust and nuanced picture of agent behavior across original and modified environments.

IQM over Multiple Games and Modifications

To evaluate the robustness of each model across multiple environments and their corresponding modifications, we compute the Expert HNS over all available games, *cf.* Figure 1. For modified environments, each game's (multiple) variants are first aggregated by computing the mean of HNS across modifications, ensuring that games with more variants do not exert disproportionate influence on the result. To aggregate the games, we follow Agarwal et al. (2021), using `rliable`. This means we calculate the IQM over all results of all seeds for all games.

Using the Interquantile Mean (IQM) + 95% Confidence Intervall (CI) over Mean + StdErr.

Following Agarwal et al. (2021), we report the interquartile mean (IQM) along with 95% stratified bootstrap confidence intervals (CIs) as our primary performance metric. Return distributions in Atari are frequently highly variable, skewed, and heavy-tailed, often due to a small number of runs achieving unusually high scores. These outliers can inflate the sample mean, resulting in a metric that overstates the agent's typical performance. Additionally, pairing the mean with standard error (SE) implicitly assumes independent, identically distributed (i.i.d.) samples from a light-tailed, symmetric distribution, assumptions that are frequently violated in RL settings.

The IQM addresses these issues by computing the mean over the interquartile range (25th to 75th percentiles), discarding both the lowest and highest quartiles. This yields a robust point estimate that emphasizes consistent behavior across seeds and reduces the influence of extreme values. To quantify uncertainty, we use non-parametric bootstrap resampling to construct 95% confidence intervals, which make no distributional assumptions and better reflect the empirical variability of the data.

This combination—IQM with bootstrap CIs—provides a statistically sound and robust summary of performance, particularly well-suited for environments like Atari where outcomes can vary substantially across random seeds. It enables more reliable comparisons between agents and guards against misleading conclusions driven by a few atypical trajectories.

Aggregating Human Study Results Using Mean with Per-Participant IQMs

To evaluate human performance in a robust and statistically sound manner, we report the mean over per-participant IQM as our primary summary statistic. Each participant completed multiple episodes per evaluation condition (e.g., original vs. modified environment), resulting in a distribution of scores that may be highly variable due to factors such as learning effects, momentary lapses in attention, or individual familiarity with similar tasks.

To reduce the impact of intra-participant variance and episodic outliers, we first compute the IQM across each participant's episodes. This yields a robust estimate of central performance for each individual, effectively filtering out unusually high or low scores that may not be representative of typical behavior.

After obtaining one IQM per participant per condition, we compute the mean across participants to summarize overall group-level performance. The goal of this is to obtain a more stable and representative estimate of average human performance, where all humans share the same influence on the results, independent of their performance or how many episodes they play.

Performance Change

To quantify the impact of environment modifications on agent performance, we compute the performance change (PC) as the relative degradation in performance between the original and modified environments. For a given metric M (e.g., HNS or raw return), let M_{original} denote the score in the original environment and M_{modif} the corresponding score in the modified one. In this work, we used the raw scores for the computations; as such, we subtract the random scores from each metric to make sure. The performance drop is defined as:

$$\text{PC} = \frac{(M_{\text{modif}} - R_{\text{modif}}) - (M_{\text{original}} - R_{\text{original}})}{|M_{\text{original}} - R_{\text{original}}|} \tag{2}$$

A value of 0 indicates no performance change, while values closer to 100% represent a user reaching twice the amount of points. A value close to -100% means the agent performs similar or worse than a random agent. This normalized form facilitates comparison across tasks with different score ranges and highlights robustness failures that may be masked by absolute performance metrics.

C Agents

We evaluate a diverse set of reinforcement learning (RL) agents, covering both standard baselines and object-centric architectures. Most agents are based on publicly available implementations or pretrained models, while **PPO** was trained by us specifically for this study. Below, we summarize the agents used, their foundational works, and where to find their implementations or pretrained models.

- **DQN, MDQN** – Standard value-based deep RL models. DQN is the canonical deep Q-learning agent introduced by Mnih et al. (2015), while MDQN is its Munchausen variant (Vieillard et al., 2020) that incorporates entropy regularization into Q-learning. We use the pretrained models from Gogianu et al. (2022)[2], originally built for analyzing agent robustness.
- **C51** – A distributional RL method that models return distributions instead of expected values (Bellemare et al., 2017). C51 models were also taken from Gogianu et al. (2022), though pretrained models are not available for all games included in our benchmark.
- **i-DQN** – Another strong and relatively new baseline, based on the idea of iterated Q-Networks (i-QN) (Vincent et al., 2025), enabling multiple consecutive Bellman updates by learning a tailored sequence of action-value functions where each serves as the target for the next one. We used the models by Vincent et al. (2025), which can be found on Huggingface[3].
- **IMPALA** – A scalable actor-critic architecture designed for distributed training, introduced by Espeholt et al. (2018). We evaluate pretrained models by the cleanRL team, available on Huggingface[4]
.
- **PPO** – A widely used policy-gradient baseline introduced by Schulman et al. (2017). We trained our own PPO models using the CleanRL framework (Huang et al., 2022), following their recommended hyperparameters and reproducibility practices. Training details and hyperparameters are provided in Table 1. Models will be made available after acceptance.
- **Binary Mask PPO, Planes PPO** – Object-centric PPO variants introduced by Blüml et al. (2025), incorporating structured visual representations (e.g., binary object masks, spatial planes) instead of raw pixels. These models were trained by us using modified observation wrappers. Code and trained models can be found on GitHub[5] or were provided by the authors. Note, these models were trained only for 40 million frames instead of 200 million.
- **Semantic Vector PPO, ScoBots Vector** – Agents that operate on symbolic or vector-based representations of objects in the environment, pretrained using the OCAtari and ScoBots frameworks (Delfosse et al., 2024a;b). Both Semantic Vector PPO and ScoBots Vector were provided by the authors. Final model checkpoints will be made available after acceptance.

[2]https://github.com/floringogianu/atari-agents
[3]https://huggingface.co/TheoVincent/Atari_i-QN
[4]https://huggingface.co/cleanrl
[5]https://github.com/VanillaWhey/OCAtariWrappers

C.1 Training PPO Agents

We train PPO agents for 200 million environment frames using the `v5` version of the ALE, following the evaluation protocol established by Machado et al. (2018). A fixed frameskip of 4 and a repeat action probability of 0.25 introduce environment stochasticity and align with established Atari benchmarks. Observations are preprocessed by converting RGB frames to grayscale, resizing to 84×84 pixels, and stacking the last four frames to capture short-term temporal dynamics. Each agent is trained using 3 random seeds to ensure robustness across initialization.

Our PPO policy uses an IMPALA-style convolutional encoder (Espeholt et al., 2018) with separate heads for the policy and value function. Training is conducted with CleanRL (Huang et al., 2022), with full training settings provided in Table 1. The agent is trained to maximize the sum of undiscounted episodic returns.

In addition to pixel-based PPO, we train a Semantic Vector agent using object-centric observations derived from OCAtari (Delfosse et al., 2024a). Instead of raw images, this agent receives a structured vector representation extracted from two consecutive frames. Each observation encodes object-level information, including positions, bounding boxes, and class labels for entities present in the scene. These vector inputs are passed through a multilayer perceptron encoder before being processed by a PPO policy. Training follows the same protocol as our pixel-based PPO agents, using CleanRL with adjusted hyperparameters for vector inputs.

This setup enables direct comparison between pixel-level and object-centric agents under identical training conditions and stochastic environment settings, allowing us to assess how input structure affects generalization and robustness.

Table 1: Hyperparameters used in our PPO training to ensure reproducibility and consistency.

Hyperparameter	**Value**	**Hyperparameter**	**Value**
Seeds	{0,1,2}	Learning Rate (α)	2.5×10^{-4}
Total Timesteps	2×10^{8}	Number of Environments	10
Batch Size (B)	1280	Minibatch Size (b)	320
Update Epochs	4	GAE Lambda (λ)	0.95
Discount Factor (γ)	0.99	Value Loss Coefficient (c_v)	0.5
Entropy Coefficient (c_e)	0.01	Clipping Coefficient (ϵ)	0.1
Clip Value Loss	`True`	Max Gradient Norm ($\|\|g\|\|_{\text{max}}$)	0.5

D Computational Resources

Although many of the agents used in this study were obtained from publicly available repositories and did not require retraining (cf. Appendix C), substantial computing resources were required to conduct large-scale evaluations across our proposed HackAtari benchmark.

Evaluation Scale. Each of our $11 - 12$ agents was evaluated on 17 games with $1 - 4$ simplified or modified variations per game (in total 50 game configurations), across 3 random seeds and 10 episodes per configuration. This resulted in around $1,500$ gameplay rollouts per agent. While these are not expensive, the amount still leads to non-trivial inference costs. The total evaluation workload included both deep RL baselines (e.g., DQN, PPO, IMPALA, C51, i-DQN,...) and object-centric agents (e.g., binary masks, planes, OC-NN, ScoBots,...), for a total of 12 unique agent configurations. This results in around $18,000$ evaluation runs. The results can be seen in Appendix E. Evaluation used deterministic seeds and fixed emulator settings (e.g., sticky actions) to ensure reproducibility (also *cf.* Appendix E).

Training Resources. A subset of agents, namely, PPO (pixel-based) were trained in-house using the CleanRL framework (cf. Appendix C.1). Most PPO agents took around $4 - 6$h per instance In total, we trained around 120 PPO models. Total training time for these agents amounted to approximately $500 - 700$ GPU hours.

Hardware. All training was conducted on a cluster of NVIDIA DGX systems, each equipped with A100 GPUs and sufficient RAM (greater than 40 GB per process) to support parallelized Atari rollouts, see Table 2. All evaluations, except IMPALA, were conducted using Python 3.11, Gym 0.26.2, and the ALE-py Atari interface (v0.8.1), on macOS 15.3.1. IMPALA, due to their requirement of the envpool package, was evaluated on a separate device, using Python 3.9 and Arch 2025.02.01.

Hardware/Software	**Description**
GPU	$8 \times$ NVIDIA® Tesla A100
NGC Container	`nvcr.io/nvidia/pytorch:23.05-py3`
GPU-Driver	CUDA 12.2
CPU	Dual Intel Xeon Platinum 8168
Operating System	Ubuntu 23.02 LTS

Table 2: Hard- and software configuration for our experimental section.

E Extended Results

Evaluation Setup

Our evaluation benchmarks RL agents across *17* Atari environments, each tested under 1 to 4 environment modifications, depending on the environment. All agents are trained, not always by ourselves, exclusively on the original, unmodified versions of the games using 200 million environment frames and the hyperparameters listed in Table 1, unless otherwise specified in the figure or table captions. Evaluation is performed using our own evaluation setup, based on HackAtari.

Each experiment is conducted over 3 seeds $(0, 1, 2)$ with 10 episodes per game per seed, totaling 30 episodes per configuration. We follow the evaluation guidelines by Machado et al. (2018), e.g., using a repeat action probability of 0.25 and a maximum of 30 NOOP actions after each reset, meaning initial states are sampled by taking a random number of no-ops on reset. No-op is assumed to be action 0.

The evaluated agent architectures and training settings are described in detail in Appendix C. For performance metrics, we report raw episodic returns, interquartile mean (IQM), and performance change, as defined in Section B. The selected environments and the corresponding modifications applied to each are introduced in Section H.

Table 3: Evaluation Parameters

Hyperparameter	**Value**	**Hyperparameter**	**Value**
#Episodes	30	#Seeds	3
Epsilon	0.001	Frameskip	4
Frames stacked	4	Repeat Action Probability	0.25
Full Actionspace	False	Max. NOOP Actions after Reset	30

Mapping Results to Figure

Table 4: Mapping of HackAtari game variations to figures in the main paper. Each entry indicates which figure(s) include evaluation results for the given game and modification pair. Games in pink are used in the human study (cf. Appendix G).

Game	**Variant**	**Appears In**
Amidar	paint roller player	Fig. 3, 4, 5
Amidar	pig enemies	Fig. 3
Asterix	obelix	Fig. 3, 4, 5
BankHeist	two police cars	Fig. 3, 4, 5
Bowling	shift player	Fig. 3
Bowling	top pins	Fig. 3, 4, 5
Boxing	color player red	Fig. 3, 4, 5
Boxing	switch positions	Fig. 3
Breakout	color all blocks red	Fig. 3
Breakout	color player and ball red	Fig. 3, 4, 5
FishingDerby	fish on different sides	Fig. 3
Freeway	all black cars	Fig. 3
Freeway	stop all cars edge	Fig. 3
Freeway	stop all cars	Fig. 3, 4, 5
Freeway	stop random car	Fig. 3
Frostbite	reposition floes easy	Fig. 3, 4, 5
Jamesbond	straight shots	Fig. 3
Kangaroo	no danger	Fig. 3, 4, 5
MsPacman	set level 1	Fig. 3, 4, 5
MsPacman	set level 2	Fig. 3
MsPacman	set level 3	Fig. 3
Pong	lazy enemy	Fig. 3, 4, 5
RiverRaid	exploding fuels	Fig. 3
RiverRaid	game color change01	Fig. 3
RiverRaid	restricted firing	Fig. 3, 4, 5
Seaquest	disable enemies	Fig. 3, 4, 5
SpaceInvaders	relocate shields off by one	Fig. 3
SpaceInvaders	relocate shields off by three	Fig. 3, 4, 5
StarGunner	remove mountains	Fig. 3, 4, 5
StarGunner	static bomber	Fig. 3
StarGunner	static flyers	Fig. 3
StarGunner	static mountains	Fig. 3

E.1 Can Deep Agents solve simplifications?

We present the raw scores obtained by all evaluated deep RL agents across a set of Atari games and their corresponding HackAtari variations. Each agent was trained solely on the original environment and evaluated on both the unmodified and modified versions without any fine-tuning or adaptation. Performance is measured over 30 episodes per configuration, averaged across 3 random seeds, and reported as interquartile means (IQM) with 95% confidence intervals.

The task variations are primarily designed as *simplifications*—modifications that preserve core game mechanics while reducing visual or strategic complexity or modifications that should not influence the performance of an agent, e.g., color changes. They provide a strong test for determining whether agents have learned task-aligned policies or are relying on superficial cues. A reliable agent should maintain or improve performance under these conditions.

The table also indicates which task variations were used in our human study (cf. Appendix E). These entries are highlighted in pink.

Table 5: Raw episodic scores for all evaluated deep RL agents across selected Atari games and their HackAtari task variations (cf. Figure 1). Each entry shows the interquartile mean (IQM) and 95% confidence interval across 30 episodes and 3 random seeds. Variants marked in pink were used in the human study (cf. Appendix G) and in Figure 2.

Game (Variant)	DQN	C51	PPO	IMPALA	i-DQN	MDQN
Amidar	867 [771, 956]	–	1052 [1017, 1111]	1215 [1131,1298]	666 [452,901]	1481 [1225, 1928]
paint roller player	71 [54, 94]	–	271 [189, 377]	205 [166,255]	529 [439,630]	160 [94, 300]
pig enemies	480 [316, 664]	–	1313 [1177, 1454]	1122 [890,1369]	70 [53,86]	708 [555, 1248]
Asterix	10212 [6734, 17884]	7766 [5262, 17937]	8588 [6915, 11412]	3543 [2084,7491]	4017 [3194,4888]	5116 [3046, 9275]
obelix	3812 [3062, 4593]	81812 [52250, 121593]	5594 [4687, 6750]	240218 [136250,346501]	6269 [5288,7365]	4594 [3437, 5906]
BankHeist	1064 [1011, 1113]	–	1062 [1028, 1092]	388 [222,636]	1291 [1226,1366]	1319 [1245, 1413]
two police cars	0 [0, 0]	–	0 [0, 0]	0 [0,0]	0 [0,0]	0 [0, 0]
Bowling	34 [30, 39]	46 [40, 53]	67 [65, 69]	45 [42,49]	42 [40,45]	33 [33, 35]
shift player	36 [31, 41]	45 [39, 50]	63 [54, 65]	39 [35,44]	43 [39,48]	37 [34, 40]
top pins	40.88 [9,94]	37.38 [23,56]	80.00 [60,96]	62 [38,87]	26 [21,41]	181.38 [139,213]
Boxing	92 [88, 94]	77 [68, 85]	99 [97, 99]	94 [92,97]	96 [95,98]	95 [93, 97]
color player red	−2 [−5, 0]	−7 [−12, −2]	−1 [−2, 0]	−3 [−8,0]	−2 [−6,−0]	−2 [−3, 0]
switch positions	47 [21, 68]	−16 [−29, −1]	57 [34, 68]	94 [92,97]	67 [60,72]	46 [23, 68]
Breakout	123 [85, 178]	41 [27, 64]	391 [366, 409]	303 [245,350]	259 [219,295]	284 [202, 349]
color all blocks red	144 [91, 215]	23 [12, 37]	413 [396, 424]	247 [175,322]	103 [63,173]	261 [176, 331]
color player and ball red	4 [2, 6]	2 [1, 3]	4 [3, 6]	76 [59,121]	8 [7,10]	15 [11, 19]
FishingDerby	32 [24, 40]	5 [0, 13]	41 [37, 44]	34 [31,38]	32 [28,38]	42 [32, 50]
fish on different sides	−90 [−92, −88]	−84 [−87, −82]	−98 [−98, −97]	27 [25,30]	20 [16,24]	−94 [−96, −90]
Freeway	27 [16, 33]	32 [31, 32]	31 [30, 31]	33 [33,33]	33 [33,33]	34 [33, 34]
all black cars	10 [7, 13]	18 [16, 20]	25 [23, 26]	28 [27,29]	26 [24,29]	25 [24, 26]
stop all cars edge	6 [1, 13]	34 [22, 40]	39 [37, 39]	11 [6,19]	0 [0,6]	40 [39, 40]
stop all cars	0 [0, 0]	0 [0, 0]	7.56 [0,20]	35 [29,38]	0 [0,1]	0.38 [0,1]
stop random car	16 [9, 20]	22 [20, 22]	21 [20, 21]	22 [21,22]	16 [16,18]	24 [22, 24]
Frostbite	5431 [4411, 6624]	5165 [4466, 6055]	301 [288, 311]	281 [262,299]	6198 [5196,7193]	8349 [6961, 8951]
reposition floes easy	198 [43, 461]	22 [0, 63]	9 [0, 28]	0 [0,2]	0 [0,0]	28 [12, 58]
Jamesbond	972 [887, 1046]	3056 [1099, 5134]	2216 [1850, 2596]	11778 [9775,13891]	600 [571,637]	722 [662, 778]
straight shots	22 [6, 46]	94 [53, 762]	481 [215, 987]	1772 [734,3319]	63 [46,77]	91 [65, 118]
Kangaroo	9669 [7962, 11487]	7456 [5231, 10012]	13525 [12275, 14243]	6244 [4200,8181]	14269 [13581,14550]	13588 [12581, 14012]
no danger	19 [0, 50]	6 [0, 37]	0 [0, 0]	0 [0,0]	15 [0,54]	81 [50, 100]
MsPacman	4232 [3632, 4636]	–	7225 [6684, 7600]	3652 [3089,4212]	3667 [3142,4180]	3961 [3208, 4579]
set level 1	602 [500, 718]	–	357 [291, 438]	631 [527,744]	328 [283,414]	1412 [1216, 1555]
set level 2	333 [240, 426]	–	326 [229, 428]	786 [641,979]	408 [368,465]	432 [307, 513]
set level 3	356 [306, 412]	–	347 [295, 419]	738 [477,986]	176 [139,214]	231 [158, 302]
Pong	19 [17, 19]	5 [4, 6]	18 [14, 19]	9 [7,11]	19 [19,20]	20 [18, 20]
lazy enemy	−15 [−17, −12]	−13 [−15, −8]	−15 [−17, −13]	6 [1,10]	−18 [−19,−17]	−16 [−18, −12]
Riverraid	14030 [12366, 14900]	–	16186 [15134, 16920]	17421 [14467,20599]	11251 [10166,12375]	15340 [14739, 15953]
exploding fuels	5530 [5129, 5847]	–	7972 [7710, 8233]	6756 [5249,8446]	4763 [4229,5383]	6094 [5454, 6671]
game color change01	439 [405, 475]	–	368 [256, 480]	721 [592,888]	379 [302,461]	234 [138, 427]
restricted firing	757 [576, 984]	–	933 [520, 1820]	1656 [1313,2114]	1573 [1227,1928]	1155 [883, 1553]
Seaquest	5916 [5281, 6750]	123622.50 [38904, 222808]	1836 [1826,1840]	951 [945,958]	6336 [4995,7672]	18170 [15821, 21457]
disable enemies	268 [0, 717]	0 [0, 0]	0 [0, 0]	0 [0,0]	1134 [409,2030]	1392 [676, 5015]
SpaceInvaders	4173 [2305, 7020]	–	1406 [1179, 1668]	11514 [5439,19878]	3225 [2222,4798]	6183 [3195, 10738]
relocate shields off by one	1314 [949, 2302]	–	1533 [1320, 1725]	10415 [6222,15419]	3853 [2041,6291]	2863 [1983, 6178]
relocate shields off by three	475 [322, 684]	–	1158 [919, 1385]	8252 [4306,13849]	1037 [864,1454]	820 [578, 1344]
StarGunner	59269 [55750, 61300]	31894 [22831, 42906]	29288 [15737, 42162]	166881 [159112,177275]	55411 [51977,58388]	63181 [61068, 64781]
remove mountains	50838 [43012, 56243]	24688 [15562, 35025]	15781 [8693, 23206]	171169 [162438,180062]	9726 [7177,12427]	58931 [56487, 60975]
static bomber	64506 [62362, 66925]	25250 [17175, 36750]	41475 [24381, 55043]	172731 [163138,181425]	55146 [51323,57812]	66475 [65006, 68143]
static flyers	235031 [209081, 260543]	24856 [14675, 36993]	4244 [950, 13562]	878106 [869850,884031]	68684 [40065,112265]	190569 [133318, 246019]
static mountains	46950 [37356, 54575]	32769 [23600, 42875]	19562 [10293, 29275]	159169 [152875,165269]	53965 [49888,56515]	61769 [59487, 63681]

E.2 Can Object-centricity solve misalignment?

This section presents the raw performance scores of object-centric agents evaluated on the same HackAtari task variations described in Appendix C.1. These agents incorporate structured inductive biases through symbolic, object-based, or mask-based representations. While such representations are often credited with improved interpretability and generalization in structured environments, our results assess whether they also support better alignment with simplified task variants.

All object-centric agents were trained using Proximal Policy Optimization (PPO) and receive object-based inputs—either through learned visual decompositions (e.g., binary masks or spatial planes) or symbolic representations (e.g., object class vectors from OCAtari and ScoBots). For comparison, we include the standard PPO baseline trained on raw pixel inputs.

Each entry reports the interquartile mean (IQM) and 95% confidence interval across 30 episodes and 3 random seeds. A reliable object-centric agent should demonstrate resilience to both visual and gameplay simplifications, avoiding performance drops that indicate overreliance on spurious correlations. However, as our findings show, object-centricity improves robustness primarily to visual perturbations, but fails to fully eliminate shortcut behavior in more complex dynamics-altering variations.

Table 6: Episodic scores for object-centric RL agents on HackAtari task variations. All agents use PPO and differ only in their input representation (e.g., binary masks, planes, or symbolic vectors). Models were taken by Blüml et al. (2025) and Delfosse et al. (2024a;b) or self-trained. Scores are reported as interquartile means (IQM) with 95% confidence intervals, evaluated over 30 episodes and 3 seeds. PPO with raw pixels, as done by Mnih et al. (2015), is included as a baseline. Games and Modifications in pink were used in Figure 3 and are identical to the human study.

Game (Variant)	PPO	Object Masks	Binary Masks	Class Masks	Planes	Semantic Vector	ScoBots
Amidar	1052 [1017, 1111]	554 [493, 615]	525 [430, 605]	479 [442, 513]	527 [509, 552]	357 [325, 407]	116 [94, 128]
paint roller player	271 [189, 377]	315 [251, 369]	535 [435, 610]	492 [458, 526]	510 [491, 528]	301 [252, 350]	116 [91, 129]
pig enemies	1313 [1177, 1454]	384 [332, 426]	513 [415, 597]	516 [465, 554]	108 [91, 118]	25 [16, 34]	18 [17, 20]
Asterix	8588 [6915, 11412]	4065 [3522,4497]	171 [116,241]	4181 [3528,4900]	91753 [83844,98809]	733 [358, 1066]	883 [624, 1133]
obelix	5594 [4687, 6750]	4562 [3812,5406]	2062 [1719,2594]	3375 [2750,4469]	30343 [21312,46312]	4583 [3039, 7543]	4000 [3333, 5500]
BankHeist	1062 [1028, 1092]	781 [769, 794]	1192 [1158, 1211]	1181 [1151, 1205]	1154 [1013, 1302]	1163 [1155, 1173]	720 [670, 741]
two police cars	0 [0, 0]	0 [0, 0]	0 [0, 0]	0 [0, 0]	0 [0, 0]	61 [42, 91]	108 [80, 133]
Bowling	67 [65, 69]	65 [62, 67]	70 [67, 71]	64 [61, 67]	99 [99, 99]	99 [97, 99]	51 [41, 60]
shift player	63 [54, 65]	63 [61, 66]	67 [64, 70]	62 [60, 66]	84 [53, 85]	0 [0, 0]	51 [41, 61]
top pins	80 [60, 96]	58 [37,78]	98 [49,159]	184 [171,195]	98 [73,124]	8 [8, 8]	0 [0, 0]
Boxing	99 [97, 99]	94 [92, 96]	96 [95, 97]	94 [93, 96]	97 [96, 98]	87 [78, 96]	51 [41, 60]
color player red	−1 [−2, 0]	80 [76, 83]	96 [95, 97]	95 [93, 96]	96 [95, 98]	87 [78, 95]	51 [41, 61]
switch positions	57 [34, 68]	−53 [−77, −11]	−65 [−97, −18]	29 [−12, 65]	37 [−14, 77]	−35 [−42, −31]	63 [45, 78]
Breakout	391 [366, 409]	216 [156, 279]	259 [212, 295]	222 [171, 267]	371 [348, 390]	38 [25, 51]	21 [14, 28]
color all blocks red	413 [396, 424]	299 [247, 328]	278 [229, 312]	259 [207, 290]	372 [330, 403]	38 [25, 51]	21 [14, 28]
color player and ball red	4 [3, 6]	99 [72, 155]	256 [201, 287]	216 [165, 266]	390 [352, 407]	38 [25, 51]	21 [14, 29]
FishingDerby	41 [37, 44]	20 [13,28]	−81 [−84,−79]	−62 [−70,−56]	47 [42,52]	18 [11, 26]	23 [18, 27]
fish on different sides	−98 [−98, −97]	23 [9,30]	−86 [−89,−83]	−60 [−67,−53]	30 [27,32]	3 [−4, 21]	−4 [−12, 4]
Freeway	31 [30, 31]	33 [32, 33]	33 [33, 33]	32 [32, 32]	33 [33, 34]	31 [31, 32]	30 [28, 31]
all black cars	25 [23, 26]	23 [21, 25]	33 [32, 33]	32 [32, 33]	33 [33, 34]	31 [31, 32]	30 [28, 31]
stop all cars edge	39 [37, 39]	0 [0, 0]	0 [0, 0]	7 [0, 19]	22 [6, 37]	0 [0, 0]	41 [41, 41]
stop all cars	7 [0, 20]	0 [0,0]	0 [0,0]	32 [20,40]	19 [4,36]	0 [0, 0]	41 [41, 41]
stop random car	21 [20, 21]	18 [17, 20]	19 [18, 20]	19 [18, 20]	20 [20, 21]	15 [13, 16]	21 [20, 22]
Frostbite	301 [288, 311]	278 [270, 290]	290 [278, 304]	265 [258, 270]	278 [271, 286]	2821 [2488, 3134]	1931 [1806, 2125]
reposition floes easy	9 [0, 28]	11 [0, 32]	4 [0, 11]	1 [0, 24]	28 [0, 79]	60 [43, 75]	70 [60, 80]
Jamesbond	2216 [1850, 2596]	–	–	–	–	591 [525, 1583]	175 [125, 241]
straight shots	481 [215, 987]	–	–	–	–	108 [58, 200]	0 [0, 21]
Kangaroo	13525 [12275, 14243]	112 [0,300]	8425 [5550,10625]	112 [0,300]	1800 [1800,1800]	0 [0, 0]	0 [0, 0]
no danger	0 [0, 0]	0 [0, 0]	0 [0, 0]	0 [0, 0]	0 [0, 0]	0 [0, 0]	0 [0, 0]
MsPacman	7225 [6684, 7600]	5195 [4476, 5724]	4227 [3646, 4896]	4313 [3790, 4899]	6211 [5471, 6794]	4583 [3650, 5298]	3620 [3086, 3620]
set level 1	357 [291, 438]	281 [236, 319]	548 [400, 778]	378 [256, 553]	294 [234, 366]	210 [210, 210]	90 [90, 90]
set level 2	326 [229, 428]	285 [258,317]	376 [323,439]	695 [446,969]	126 [104,161]	170 [170, 170]	90 [90, 90]
set level 3	347 [295, 419]	259 [222,301]	628 [512,775]	228 [110,497]	124 [98,172]	1610 [1610, 1610]	90 [90, 90]
Pong	18 [14, 19]	18 [18, 19]	19 [18, 20]	19 [19, 20]	19 [19, 20]	19 [19, 20]	16 [15, 19]
lazy enemy	−15 [−17, −13]	12 [10, 15]	12 [5, 16]	3 [−4, 9]	17 [16, 18]	−20 [−20, −19]	17 [15, −19]
Riverraid	16186 [15134, 16920]	7948 [7814, 8092]	7958 [7739, 8204]	7803 [7552, 7993]	7990 [7824, 8198]	2175 [2086, 2350]	2230 [2074, 2368]
exploding fuels	7972 [7710, 8233]	4111 [3886, 4201]	3651 [3071, 4038]	4039 [3801, 4279]	4136 [3999, 4272]	1068 [1010, 1241]	1186 [952, 1221]
game color change01	368 [256, 480]	7638 [7474, 7811]	7850 [7745, 8011]	7867 [7567, 8105]	8042 [7841, 8242]	2175 [2091, 2353]	2230 [2075, 2353]
restricted firing	933 [520, 1820]	523 [511, 539]	1677 [1460, 1964]	2261 [1850, 2594]	1864 [1524, 2359]	200 [200, 200]	2741 [2216, 3058]
Seaquest	1836 [1826,1840]	1656 [1388,1820]	732 [465,1141]	1832 [1815,1840]	2013 [1840,2308]	293 [254, 353]	293 [256, 360]
disable enemies	0 [0, 0]	0 [0, 0]	0 [0, 0]	0 [0, 0]	0 [0, 0]	0 [0, 0]	0 [0, 0]
SpaceInvaders	1406 [1179, 1668]	688 [600, 806]	826 [763, 909]	554 [475, 634]	1557 [1247, 1851]	278 [205, 321]	335 [215, 451]
relocate shields off by one	1533 [1320, 1725]	621 [560, 693]	803 [730, 884]	483 [362, 588]	1787 [1416, 2217]	290 [256, 358]	241 [208, 318]
relocate shields off by three	1158 [919, 1385]	469 [389, 571]	589 [474, 683]	288 [245, 325]	1555 [1189, 1899]	239 [186, 347]	266 [252, 330]
StarGunner	29288 [15737, 42162]	8406 [3938,14106]	962 [900,1000]	20812 [10075,32231]	88787 [80131,94862]	10166 [5991, 18597]	11550 [7357, 15600]
remove mountains	15781 [8693, 23206]	9943 [5144,15150]	1000 [1000,1000]	23656 [13000,33488]	87287 [78219,95475]	13950 [9640, 18069]	11250 [7700, 14447]
static bomber	41475 [24381, 55043]	13375 [6512,20819]	1000 [1000,1000]	43031 [26325,54312]	83268 [68862,102025]	24766 [21231, 28208]	11550 [6815, 15821]
static flyers	4244 [950, 13562]	11625 [4881,27369]	343 [275,438]	83881 [27206,178088]	577606 [505392,646894]	23950 [7666, 59957]	11550 [7524, 15760]
static mountains	19562 [10293, 29275]	7325 [3888,11600]	1000 [1000,1000]	13718 [6506,25269]	84487 [77275,92850]	13516 [8041, 17467]	11550 [7323, 15592]

E.3 Can Humans Solve these Simplifications?

To validate that our selected HackAtari variations constitute genuine simplifications rather than increased difficulty, we conducted a user study evaluating human adaptation to a subset of modified environment (cf. Appendix G).

Table 7 presents the raw scores for each game and its corresponding variation. We report the mean over IQM (cf. Appendix B) and 95% confidence intervals across participants, alongside the random agent baseline and expert human scores (from Badia et al. (2020a)) when available. This enables robust comparison between deep RL agents and non-expert human players.

Table 7: Raw scores for humans, random agents, and expert human baselines on original and modified tasks. Variants used in the human study are marked and match those highlighted in Table 5. Scores are reported with 95% confidence intervals and were used in Figure 2.

Game (Variant)	Random	Random (Badia et al.)	Human	Human (Badia et al.)
Amidar	1.62 [0, 3]	5.8	114.58 [46, 220]	7127.7
paint roller player	1.31 [0, 2]	–	360.43 [52, 930]	–
pig enemies	1.06 [0, 3]	–	–	–
Asterix	218.75 [162, 275]	210.0	1357.54 [458, 2462]	1719.5
obelix	2062.50 [1688, 2500]	–	20384.03 [4624, 39712]	–
BankHeist	13.12 [10, 17]	14.2	268.67 [98, 506]	753.1
two police cars	0.00 [0, 29]	–	242.08 [72, 470]	–
Bowling	63.12 [54, 65]	23.1	109.74 [88, 129]	160.7
shift player	24.25 [22, 27]	–	–	–
top pins	90.44 [80, 104]	–	192.17 [157, 220]	–
Boxing	1.50 [0, 3]	0.1	$-2.75ci{-5}, 0$	12.1
color player red	1.06 [−0, 3]	–	−0.93 [−7, 6]	–
switch positions	1.06 [−0, 3]	–	–	–
Breakout	1.31 [1, 2]	1.7	8.35 [3, 14]	30.5
color all blocks red	0.94 [0, 1]	–	–	–
color player and ball red	1.00 [0, 2]	–	10.31 [4, 19]	–
Freeway	0.00 [0, 0]	0.0	17.74 [12, 22]	29.6
all black cars	0.00 [0, 0]	–	–	–
stop all cars edge	0.31 [0, 1]	–	–	–
stop all cars	0.53 [0, 1]	–	35.83 [32, 39]	–
stop random car	0.00 [0, 0]	–	–	–
Frostbite	71.25 [58, 84]	65.2	1372.81 [167, 2834]	4334.7
reposition floes easy	7.50 [1, 28]	–	7540.21 [1369, 15306]	–
Kangaroo	0.00 [0, 50]	52.0	459.97 [200, 791]	3035.0
no danger	0.00 [0, 0]	–	2431.48 [1616, 3414]	–
MsPacman	243.75 [219, 270]	307.3	996.9 [518, 1492]	6951.6
set level 1	201.88 [174, 229]	–	673.6 [305, 1206]	–
set level 2	231.88 [203, 262]	–	–	–
set level 3	187.50 [165, 219]	–	–	–
Pong	−20.62 [−21, −20]	-20.7	−15.19 [−18, −10]	14.6
lazy enemy	−20.62 [−21, −20]	–	−12.2 [−17, −6]	–
Riverraid	1533.75 [1348, 1688]	1338.5	2406.56 [1428, 3584]	17118.0
exploding fuels	860.62 [659, 946]	–	–	–
game color change01	1475.62 [1341, 1637]	–	–	–
restricted firing	520.00 [520, 579]	–	13222.81 [8391, 18407]	–
Seaquest	70.00 [54, 88]	68.4	2626.39 [331, 5397]	42054.7
disable enemies	0.00 [0, 0]	–	12191.67 [6836, 17288]	–
SpaceInvaders	137.81 [115, 169]	148.0	237.8 [148, 338]	1668.7
relocate shields off by one	117.50 [92, 148]	–	–	–
relocate shields off by three	125.62 [96, 160]	–	286.96 [175, 436]	–
StarGunner	675.00 [550, 800]	664.0	2435.83 [846, 5056]	10250.0
remove mountains	656.25 [550, 781]	–	2936.67 [1200, 5197]	–
static bomber	468.75 [356, 594]	–	–	–
static flyers	512.50 [419, 650]	–	–	–
static mountains	706.25 [588, 831]	–	–	–

F Code and Data

To support reproducibility and further research, we will release all code, task variations, evaluation scripts, and selected model checkpoints as part of the supplementary materials and the code base through an anonymized repository upon acceptance.

HackAtari Environment. The HackAtari benchmark is implemented as a lightweight extension of the Gymnasium Atari framework. All modifications are applied via deterministic RAM-based wrappers, allowing reproducible control over game dynamics, visuals, and semantics. The environment includes:

- Using any of task variants by name (*cf.* Appendix H).
- Supporting scripts to help visualize and change the RAM of the current state on the fly, as well as other tools helping to create new variants.
- Compatibility with Gym-style RL pipelines, wrappers and vectorized rollouts.
- Compatibility with StableBaselines3 and CleanRL training frameworks

Agent Models and Evaluation. We include pretrained checkpoints for selected deep and object-centric agents in the supplementary materials, covering a subset of games and variants.

Human Study Infrastructure. To facilitate future user studies and make it easy to reproduce and test it, we will release the full source code for the web-based evaluation interface used in our human experiments (see Appendix G).

Anonymized Access and Deanonymization. For the review phase, we include all evaluation results and selected models in the supplementary materials. Deanonymized versions of HackAtari, HackTari-Web, Models, and the Dataset will be provided upon paper acceptance.

G Human Study: Evaluating Human Performance of Modified Atari Environments

G.1 Motivation and Research Question

The primary motivation of this study is to understand how visual and semantic modifications to Atari game environments affect human gameplay performance. Our central research question is:

How do different types of modifications influence human players' performance in Atari games?

By comparing performance across original and modified versions of the same game, we aim to identify which transformations preserve or degrade task-relevant understanding for humans. This helps evaluate the alignment between human representation of the games and machine abstractions. It also informs us if the proposed environmental changes are doable for a human, compared to the performance of our machine learning agents.

G.2 Study Design

Participants

We recruited participants through Prolific[6]. A total of **160** participants completed the study. All participants were at least 18 years old and reported fluent English skills. Participation was voluntary, and individuals were monetarily compensated. Communication with the participants, if required, was done over Prolific.

Procedure

Each participant completed the study in three phases:

1. **Free Training (10-15 minutes):** Participants played an unmodified version of a randomly assigned Atari game. This phase allowed them to familiarize themselves with the core mechanics. Participants could proceed early by clicking an "I'm ready" button after 10 min.
2. **Evaluation Phase 1 (15 minutes):** Participants played the original version of the same game.
3. **Evaluation Phase 2 (15 minutes):** Participants then played a modified version of the game. The modification applied was determined by a token-to-variant mapping and involved visual or semantic changes.

After completing the evaluation rounds, participants answered short questions about the task (e.g., reward identification) and reported their confidence.

Each participant was assigned to a specific game and one of several predefined modification conditions. Each participant could only participate once. We used 15 games (cf. Appendix E.3) with one modification each. Assignment of participants to conditions followed a round-robin strategy over available game-modification pairs.

Tasks

Participants were asked to:

- Play the game and maximize their score across both original and modified versions, further incentivized with a bonus payment based on performance.
- Identify what they believed was the modification.
- Rate their confidence in their understanding of the task.
- Provide free-text feedback when prompted.

[6]https://www.prolific.com/

G.3 Implementation Details

The study was conducted using a custom web-based platform called **HackTari**, which was built as a demonstration tool for HackAtari modification, enabling real-time human interaction with modified Atari game environments.

The system consisted of the following components:

- A **Flask** backend that served the web interface and managed participant sessions.
- **Socket.IO** was used to support low-latency, bidirectional communication between the browser and game server, allowing responsive gameplay.
- Games were instantiated using **OpenAI Gym** environments, modified with custom wrappers to support HackAtari modifications.
- Each participant was assigned a unique **token** that mapped to a specific game-modification variant. This ensured that gameplay sessions and logging were isolated per user.
- All gameplay logs, including actions, rewards, observations, and survey responses, were stored as session-specific `.csv` files for later analysis.

The full study workflow was hosted using AWS, requiring no software installation by participants. This setup allowed scalable, remote data collection while maintaining consistency across game conditions. Screenshots of the study can be found in Figures 6 to 11.

G.4 Compensation

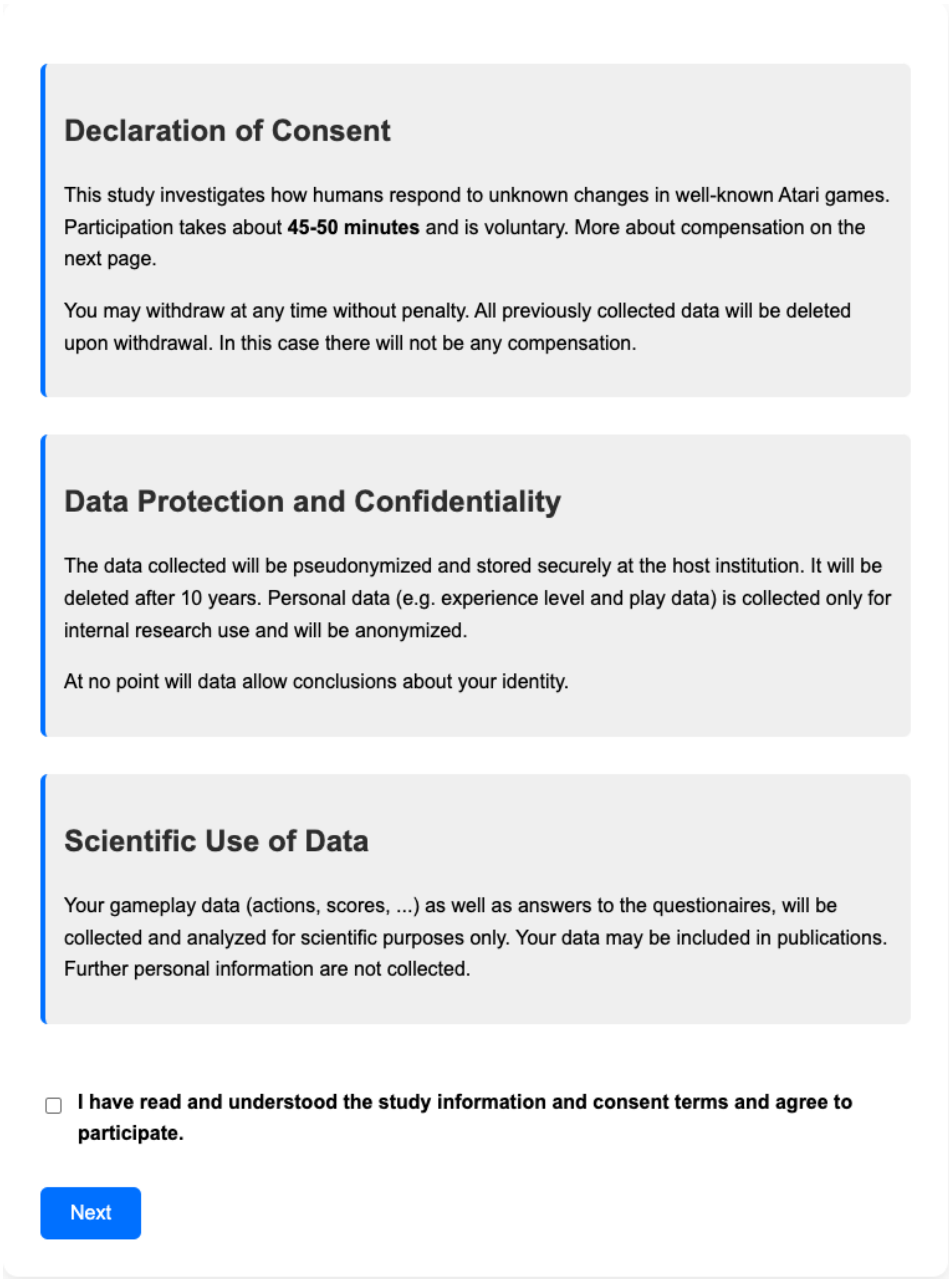
Participants were compensated with a fixed payment of £7 (equal to ~£8.50 per hour) for completing the study. Further, they were paid a bonus of up to £3 based on their performance. Importantly, this bonus was not about the performance drop but their actual performance in both phases, Eval1 and Eval2. This was done in accordance with institutional guidelines and ethical research standards.

G.5 Ethical Considerations

This study was conducted in accordance with ethical research standards. Participants provided informed consent prior to beginning the experiment. They were informed about the study design, the compensation, and general information about the game they would play. All data was anonymized, and no personally identifiable information was collected. Compensation was provided according to Prolific recommendations. The study was approved by the local ethics board.

G.6 Results

Participants were excluded from our data analysis if they did not complete all phases of the experiment, if it was clearly apparent that they stopped actively playing the game, or if they reported technical difficulties. Therefore, our final dataset includes **128** participants. We analyzed the quantitative performance metrics across the original and modified game conditions. Findings can be seen in Figure 12 to 14.



Declaration of Consent

This study investigates how humans respond to unknown changes in well-known Atari games. Participation takes about **45-50 minutes** and is voluntary. More about compensation on the next page.

You may withdraw at any time without penalty. All previously collected data will be deleted upon withdrawal. In this case there will not be any compensation.

Data Protection and Confidentiality

The data collected will be pseudonymized and stored securely at the host institution. It will be deleted after 10 years. Personal data (e.g. experience level and play data) is collected only for internal research use and will be anonymized.

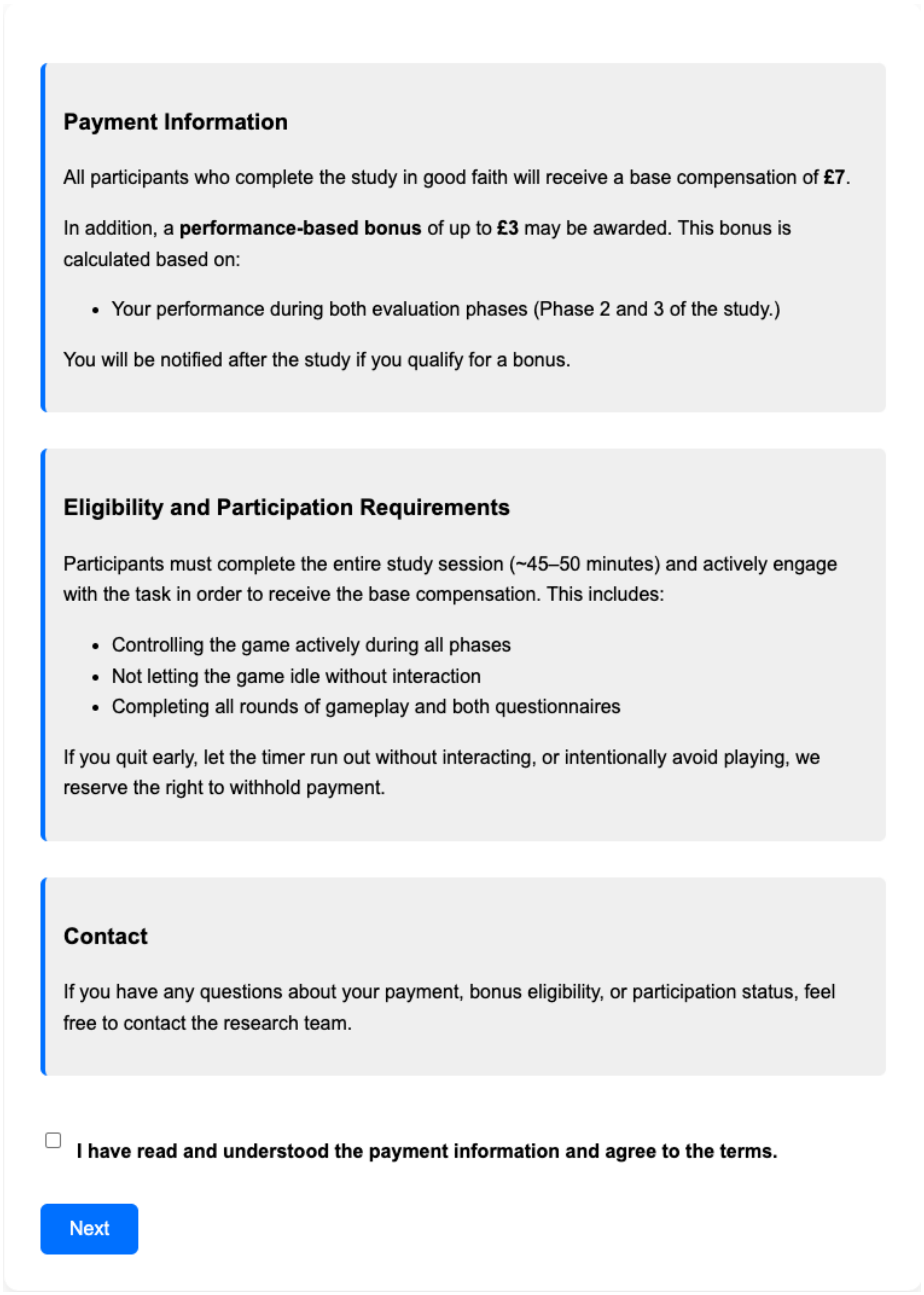
At no point will data allow conclusions about your identity.

Scientific Use of Data

Your gameplay data (actions, scores, ...) as well as answers to the questionaires, will be collected and analyzed for scientific purposes only. Your data may be included in publications. Further personal information are not collected.

☐ **I have read and understood the study information and consent terms and agree to participate.**

Figure 6: Consent form shown to participants at the beginning of the study. It includes a brief description of the study, use of data, confidentiality statement, and explicit agreement requirement before participation.



Payment Information

All participants who complete the study in good faith will receive a base compensation of **£7**.

In addition, a **performance-based bonus** of up to **£3** may be awarded. This bonus is calculated based on:

- Your performance during both evaluation phases (Phase 2 and 3 of the study.)

You will be notified after the study if you qualify for a bonus.

Eligibility and Participation Requirements

Participants must complete the entire study session (~45–50 minutes) and actively engage with the task in order to receive the base compensation. This includes:

- Controlling the game actively during all phases
- Not letting the game idle without interaction
- Completing all rounds of gameplay and both questionnaires

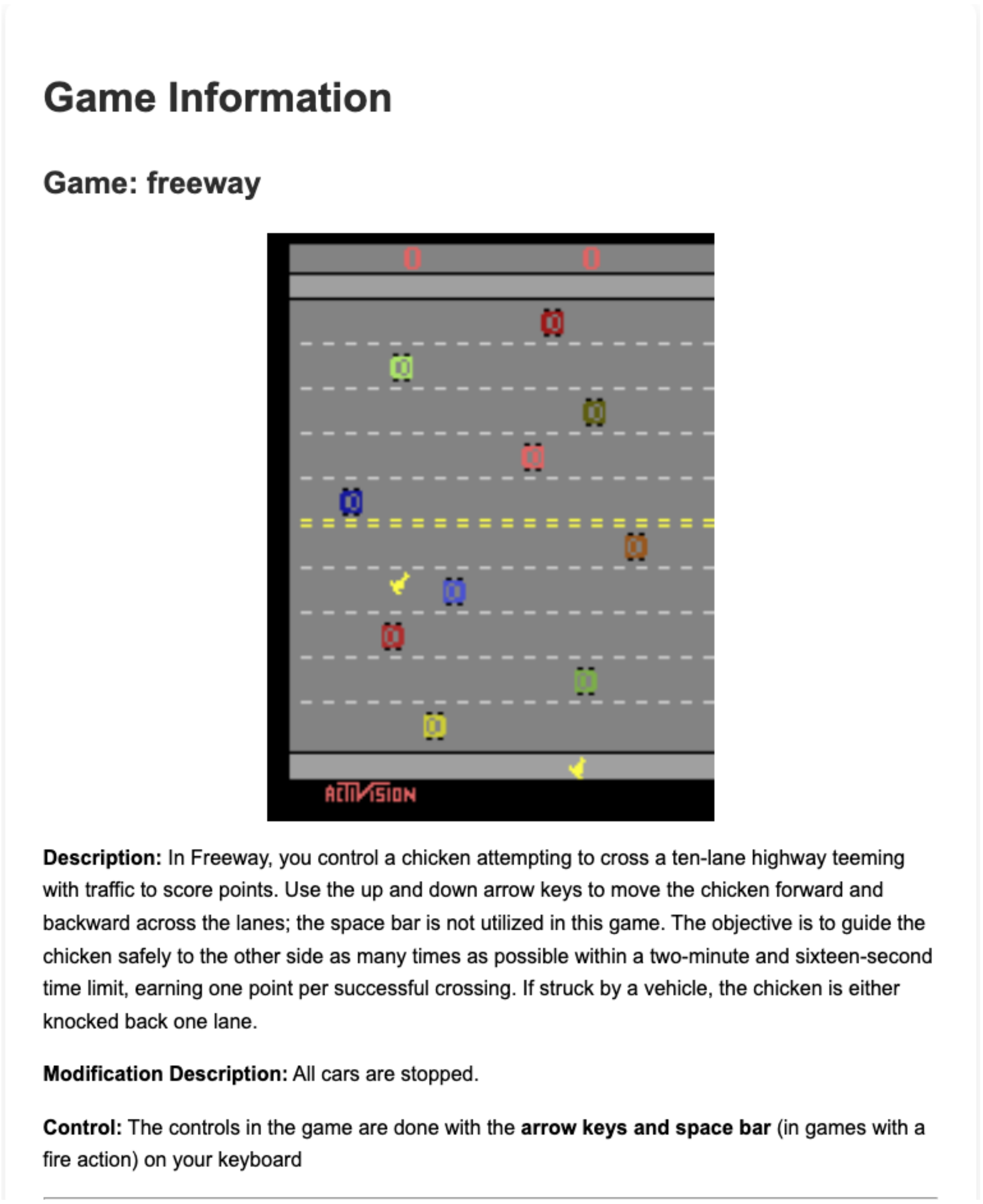
If you quit early, let the timer run out without interacting, or intentionally avoid playing, we reserve the right to withhold payment.

Contact

If you have any questions about your payment, bonus eligibility, or participation status, feel free to contact the research team.

☐ **I have read and understood the payment information and agree to the terms.**

Figure 7: Payment information screen shown before the start of the task. It outlines base compensation and performance-based bonus structure, ensuring transparency around incentives and ethical compensation. Again an explicit agreement was required before paricipation.



Game Information

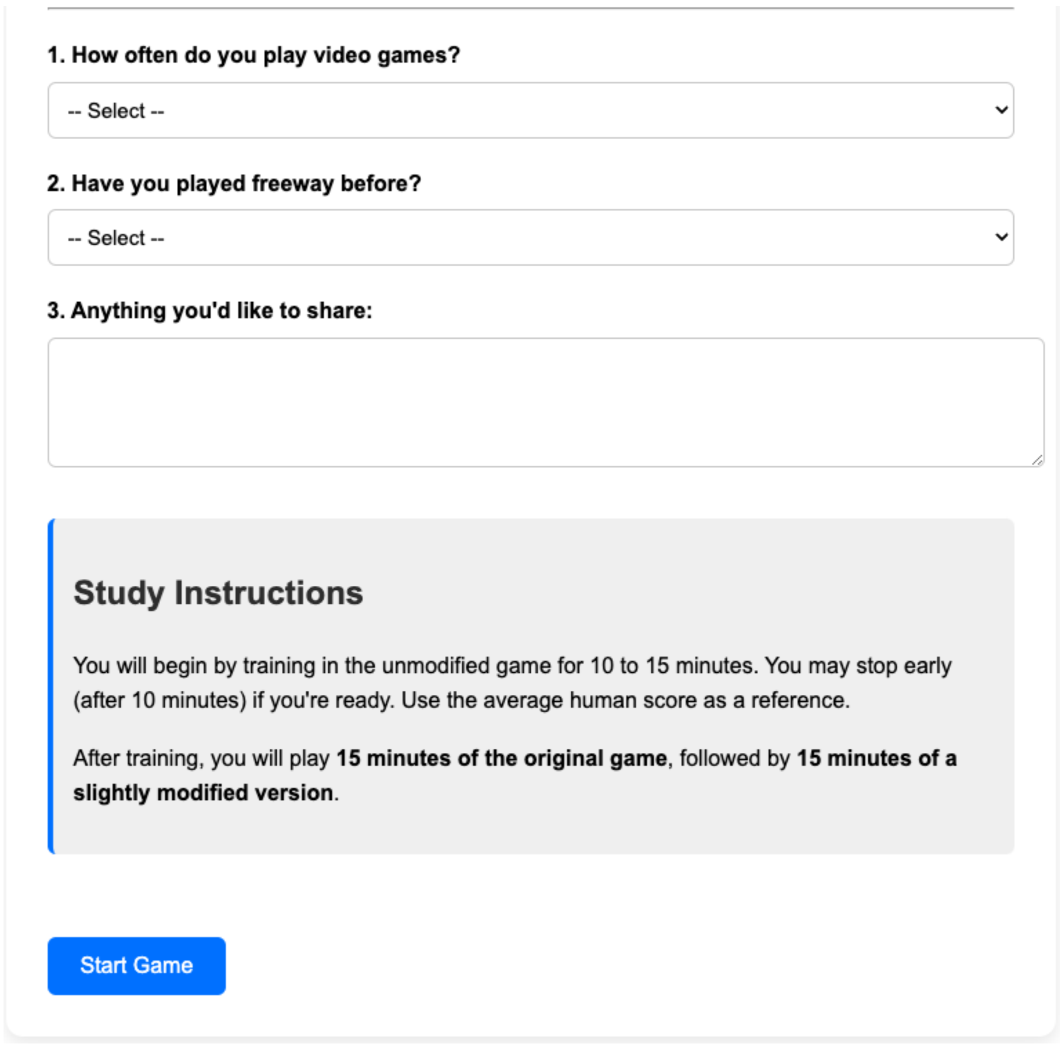
Game: freeway

Description: In Freeway, you control a chicken attempting to cross a ten-lane highway teeming with traffic to score points. Use the up and down arrow keys to move the chicken forward and backward across the lanes; the space bar is not utilized in this game. The objective is to guide the chicken safely to the other side as many times as possible within a two-minute and sixteen-second time limit, earning one point per successful crossing. If struck by a vehicle, the chicken is either knocked back one lane.

Modification Description: All cars are stopped.

Control: The controls in the game are done with the **arrow keys and space bar** (in games with a fire action) on your keyboard

Figure 8: Example game description page shown to participants before gameplay. It provides an overview of the given Atari game, including the objective, scoring mechanics, and available controls.


1. How often do you play video games?

-- Select --

2. Have you played freeway before?

-- Select --

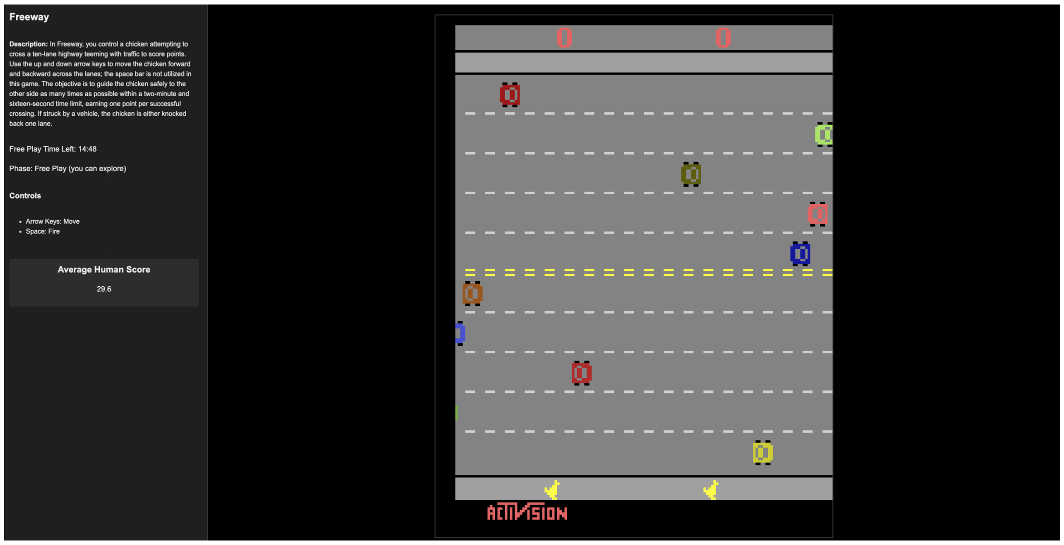
3. Anything you'd like to share:

Study Instructions

You will begin by training in the unmodified game for 10 to 15 minutes. You may stop early (after 10 minutes) if you're ready. Use the average human score as a reference.

After training, you will play **15 minutes of the original game**, followed by **15 minutes of a slightly modified version**.

Figure 9: Pre-task questionnaire to get basic information about experience. This could support the quantitative evaluation in the next step. As well as a short reminder of how the study will be run before starting it.



Freeway

Description: In Freeway, you control a chicken attempting to cross a ten-lane highway teeming with traffic to score points. Use the up and down arrow keys to move the chicken forward and backward across the lanes; the space bar is not utilized in this game. The objective is to guide the chicken safely to the other side as many times as possible within a two-minute and sixteen-second time limit, earning one point per successful crossing. If struck by a vehicle, the chicken is either knocked back one lane.

Free Play Time Left: 14:48

Phase: Free Play (you can explore)

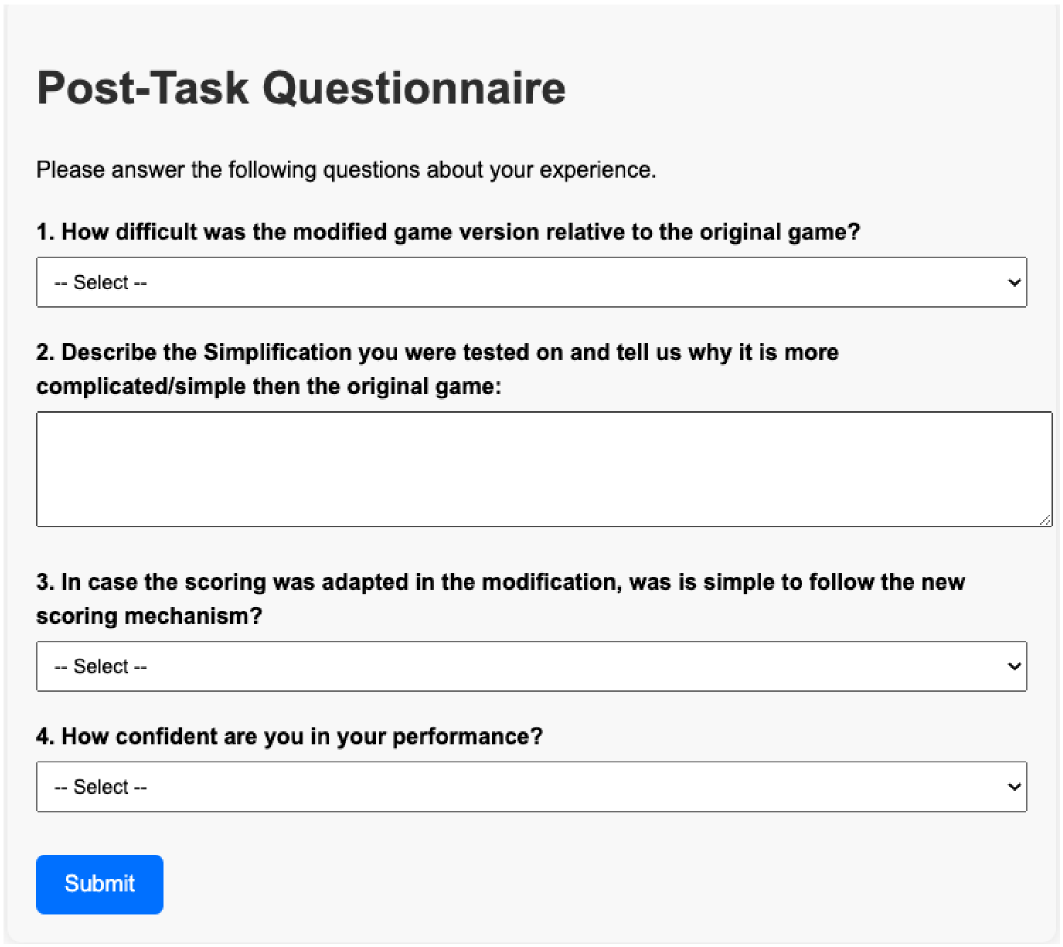
Controls

- Arrow Keys: Move
- Space: Fire

Average Human Score

29.6

Figure 10: Screenshot of the interactive HackTari gameplay interface. Participants use this interface during Free Training, Evaluation Phase 1 (original game), and Evaluation Phase 2 (modified game). The interface includes the game window, control guide, and timer. In free play the average human score, taken from Badia et al. (2020a), is shown as reference.



Post-Task Questionnaire

Please answer the following questions about your experience.

1. How difficult was the modified game version relative to the original game?

-- Select --

2. Describe the Simplification you were tested on and tell us why it is more complicated/simple then the original game:

3. In case the scoring was adapted in the modification, was is simple to follow the new scoring mechanism?

-- Select --

4. How confident are you in your performance?

-- Select --

Figure 11: Post-task questionnaire to get feedback.

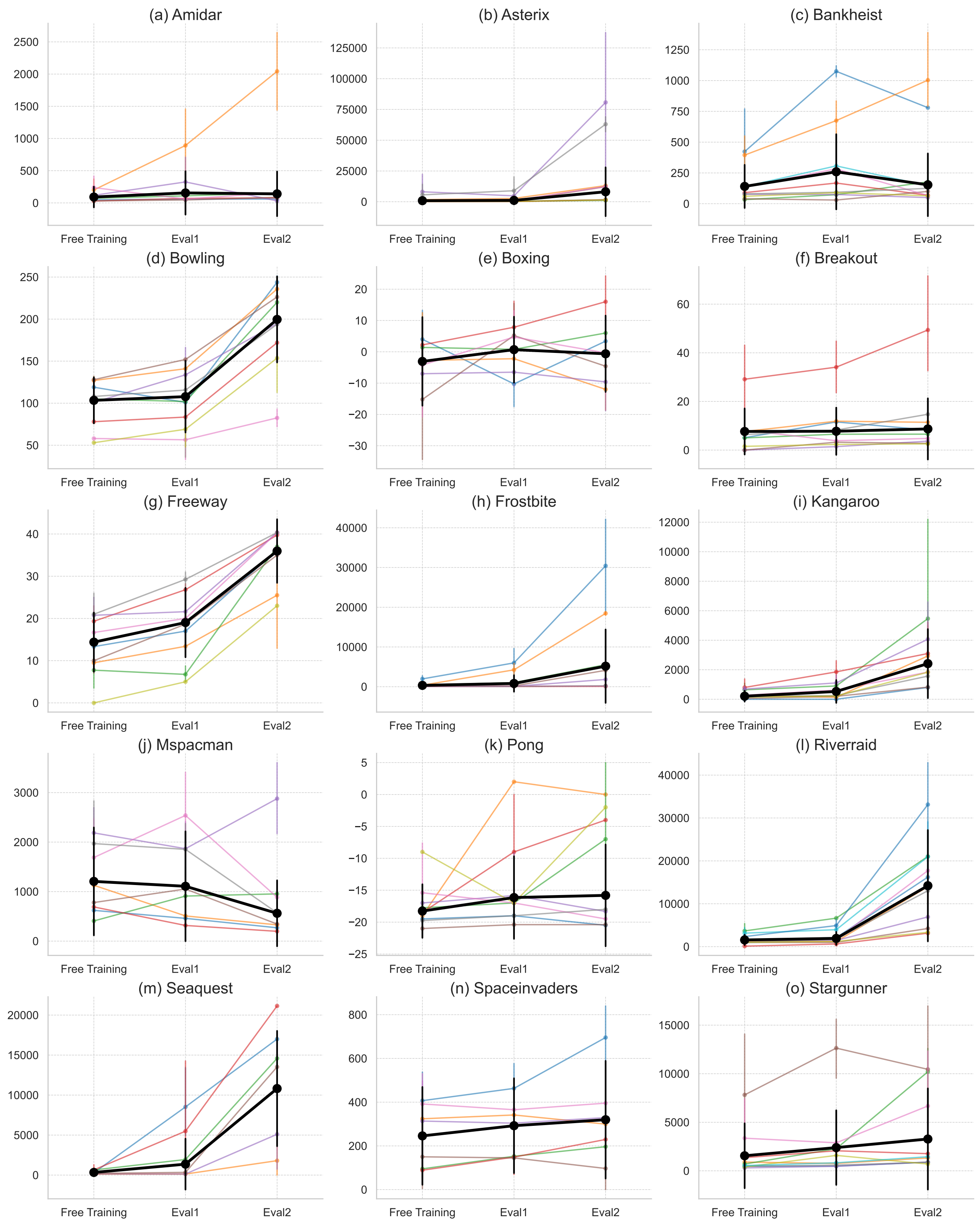


Figure 12: Raw human scores across three phases: Free Training, Evaluation on the original task (Eval1), and Evaluation on the modified task (Eval2). Each subplot shows mean scores across all participants for a specific game. Most participants improved or maintained performance on Eval2, supporting the claim that the selected HackAtari variants are true task simplifications.

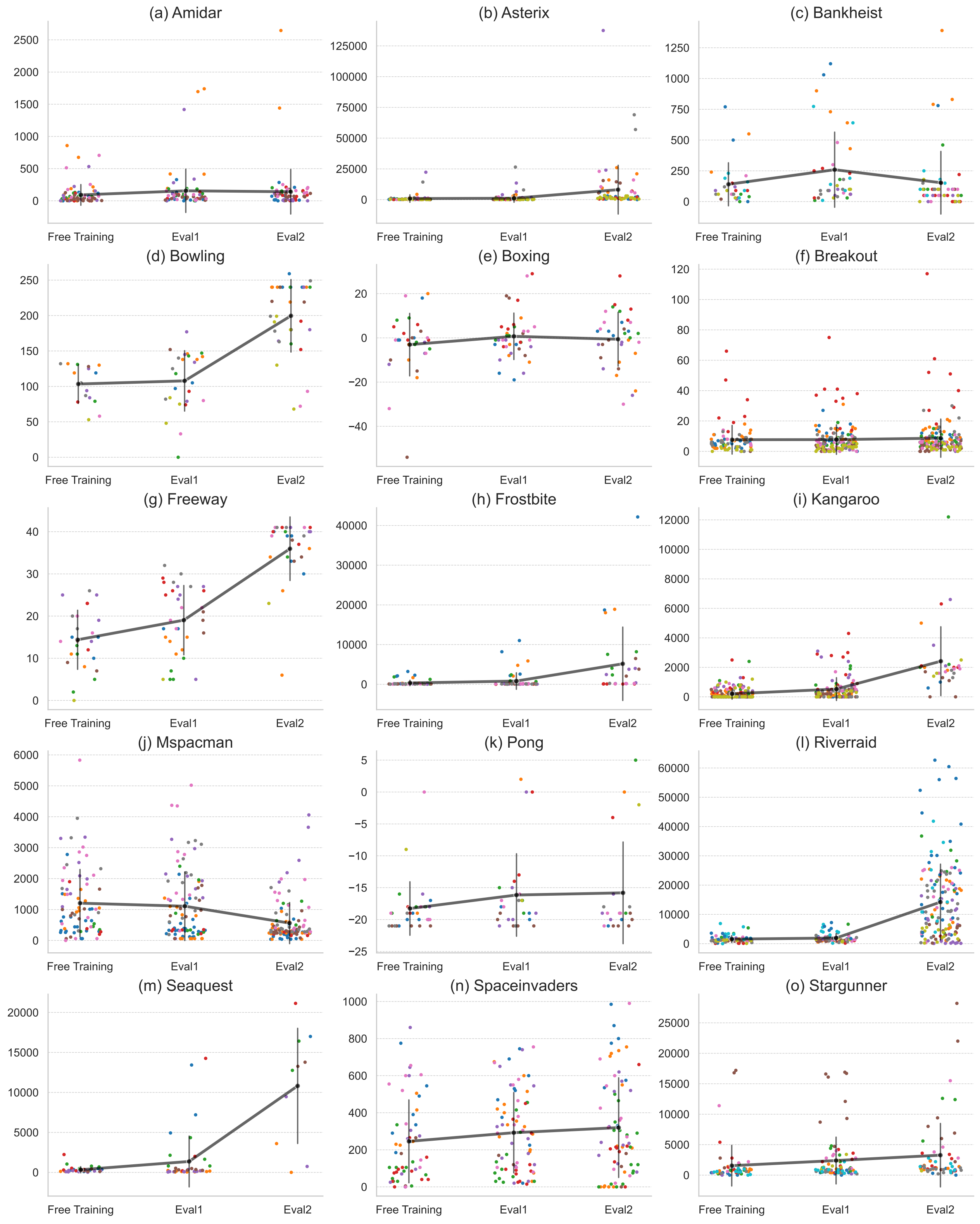


Figure 13: Visualization of all results of all users.

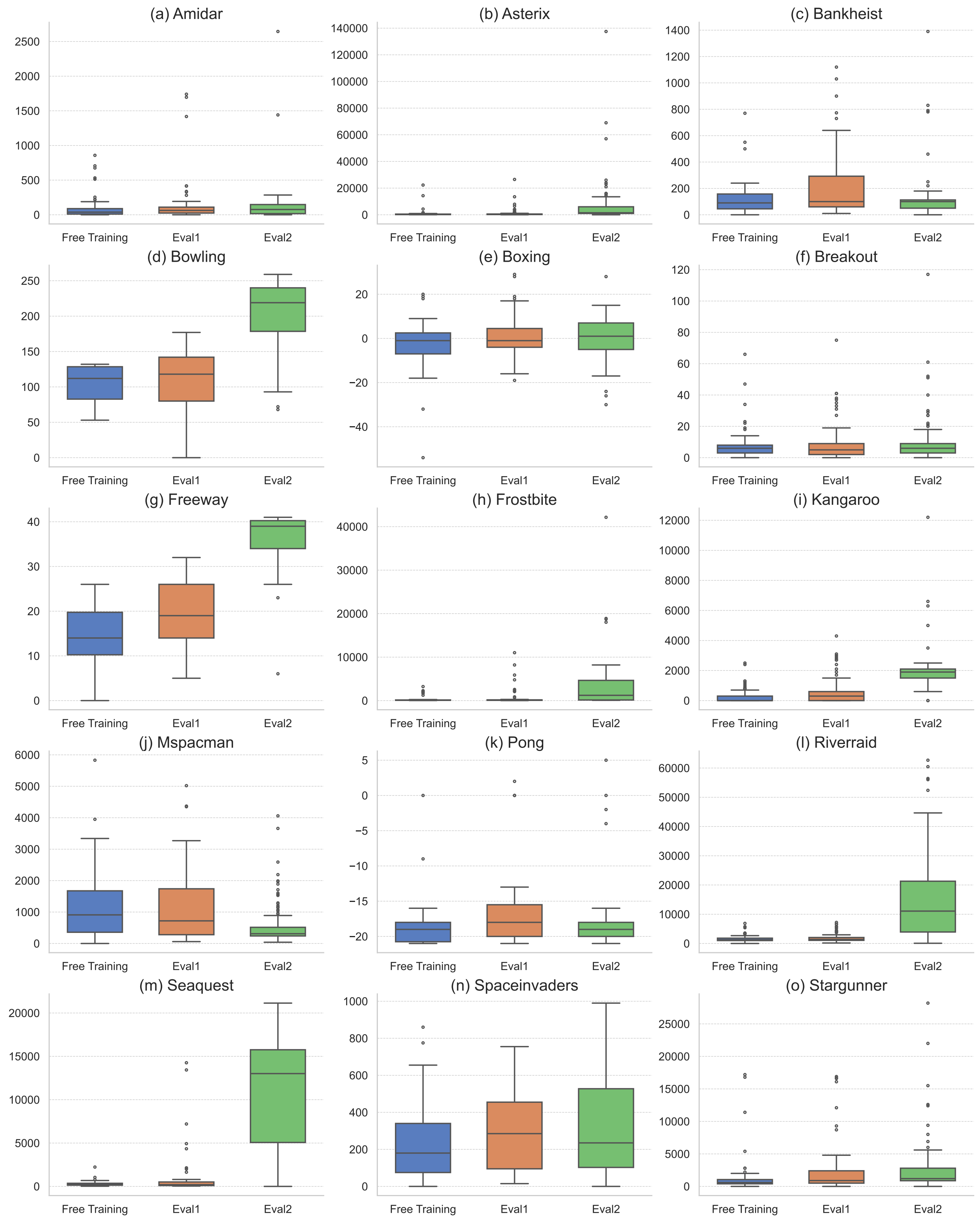
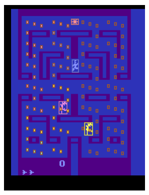


Figure 14: This presentation emphasizes the robustness of human generalization to simplified modifications.

H Game Descriptions and Modifications

H.1 Alien

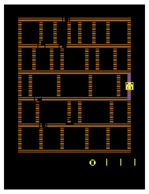
Description:
You are stuck in a maze-like space ship with three aliens. You goal is to destroy their eggs that are scattered all over the ship while simultaneously avoiding the aliens (they are trying to kill you). You have a flamethrower that can help you turn them away in tricky situations. Moreover, you can occasionally collect a power-up (pulsar) that gives you the temporary ability to kill aliens.



Modification	Effect
last_egg	Removes all eggs but one.

H.2 Amidar

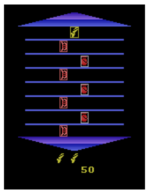
Description:
This game is similar to Pac-Man: You are trying to visit all places on a 2-dimensional grid while simultaneously avoiding your enemies. You can turn the tables at one point in the game: Your enemies turn into chickens and you can catch them.



Modification	Effect
pig_enemies	Replaces all enemies with the pig enemies.
paint_roller_player	Changes the player to the paint roller character.
unlimited_lives	Player has an unlimited amounts of lives.

H.3 Asterix

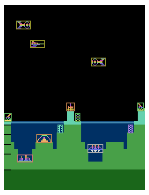
Description:
You are Asterix and can move horizontally (continuously) and vertically (discretely). Objects move horizontally across the screen: lyres and other (more useful) objects. Your goal is to guide Asterix in such a way as to avoid lyres and collect as many other objects as possible. You score points by collecting objects and lose a life whenever you collect a lyre. You have three lives available at the beginning. If you score sufficiently many points, you will be awarded additional points.



Modification	Effect
obelix	Changes the playable character to obelix.
set_consumable_1	Changes all consumables to pink consumables (100points).
set_consumable_2	Changes all consumables to shields (200points).
unlimited_lives	Player has an unlimited amounts of lives.
even_lines_free	Clears the even numbered lines (1 being the top most line).
odd_lines_free	Clears the odd numbered lines (1 being the top most line).

H.4 Atlantis

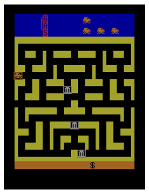
Description:
Your job is to defend the submerged city of Atlantis. Your enemies slowly descend towards the city and you must destroy them before they reach striking distance. To this end, you control turrets stationed on three defense posts. You lose if your enemies manage to destroy all seven of Atlantis' installations. You may rebuild installations after you have fought of a wave of enemies and scored a sufficient number of points.



Modification	Effect
no_last_line	Removes enemies from the lowest line.
jets_only	All enemy ships are jets.
random_enemies	Randomly assigns enemy types, instead of following the standardized pattern.
speed_mode_slow	Sets enemy speed to low.
speed_mode_medium	Sets the enemy speed to medium.
speed_mode_fast	Sets the enemy speed to fast.
speed_mode_ultrafast	Sets the enemy speed to ultrafast.

H.5 BankHeist

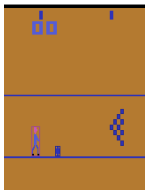
Description:
You are a bank robber and (naturally) want to rob as many banks as possible. You control your getaway car and must navigate maze-like cities. The police chases you and will appear whenever you rob a bank. You may destroy police cars by dropping sticks of dynamite. You can fill up your gas tank by entering a new city. At the beginning of the game you have four lives. Lives are lost if you run out of gas, are caught by the police, or run over some dynamite you have previously dropped.



Modification	Effect
unlimited_gas	Unlimited gas for the player.
no_police	Removes police from the game.
only_police	No banks only police.
two_police_cars	Replaces 2 banks with police cars and robbed banks give 50 points.
random_city	Randomizes which city is entered next.
revisit_city	Allows player to go back one city.

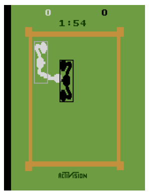
H.6 Bowling

Description:
Your goal is to score as many points as possible in the game of Bowling. A game consists of 10 frames and you have two tries per frame. Knocking down all pins on the first try is called a "strike". Knocking down all pins on the second roll is called a "spar". Otherwise, the frame is called "open".



Modification	**Effect**
shift_player	Shifts the player to the right.
horizontal_pins	Draws the pins horizontally instead of vertically.
small_pins	Decreases the pin size from 2 pixels to 1 pixel.
moving_pins	Moves the pins up and down.
top_pins	The top two pins are in-game.
middle_pins	Two middle pins are in-game.
bottom_pins	The bottom two pins are in-game.

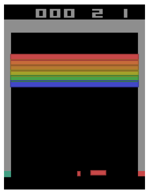
H.7 Boxing



Description:
You fight an opponent in a boxing ring. You score points for hitting the opponent. If you score 100 points, your opponent is knocked out.

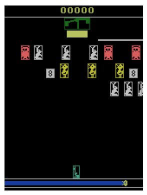
Modification	**Effect**
gravity	Adds a permanent downwards movement.
antigravity	Adds a permanent upwards movement.
offensive	Moves the player character forward in the game environment.
defensive	Moves the player character backward in the game environment.
down	Moves the player character down in the game environment.
one_armed	Disables the "hitting motion" with the right arm permanently
drunken_boxing	Applies random movements to the players input.
color_player_X	Changes the color of the player to color $X \in$ [Black, White, Red, Blue, Green]
color_enemy_X	Changes the color of the enemy to color $X \in$ [Black, White, Red, Blue, Green] by choosing a value 0-4
switch_positions	Switches the position of player and enemy
classic_colors	Using a red player and a blue enemy

H.8 Breakout



Description:
Another famous Atari game. The dynamics are similar to pong: You move a paddle and hit the ball at a brick wall at the top of the screen. Your goal is to destroy the brick wall. You can try to break through the wall and let the ball wreak havoc on the other side, all on its own! If the ball falls below your padle you lose one of your five lives.

H.9 Carnival

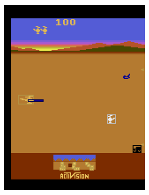


Description:
This is a “shoot ‘em up” game. Targets move horizontally across the screen and you must shoot them. You control the gun at the bottom of the screen that can be moved horizontally. The supply of ammunition is limited and chickens may steal some bullets from you if you don’t hit them in time.

Modification	**Effect**
right_drift	Applies a right drift to the ball.
left_drift	Applies a left drift to the ball.
gravity	Applies a downward force to the ball
inverse_gravity	Applies an upward force to the ball
color_player_and_ball_X	Changes the color of the player to color $X \in$ [Black, White, Red, Blue, Green].
color_all_blocks_X	Changes the color of all blocks to color $X \in$ [Black, White, Red, Blue, Green].

Modification	**Effect**
no_flying_ducks	Ducks in the last row disappear instead of turning into flying ducks.
unlimited_ammo	Ammunition doesn't decrease.
missile_speed_small_increase	The projectiles fired from the players are slightly faster.
missile_speed_medium_increase	The projectiles fired from the players are notably faster.
missile_speed_faster_increase	The projectiles fired from the players are a lot faster.

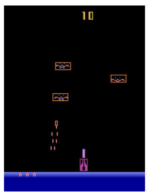
H.10 ChopperCommand



Description:
You control a helicopter and must protect truck convoys. To that end, you need to shoot down all enemy aircraft. A mini-map is displayed at the bottom of the screen that shows all truck and aircraft positions.

Modification	**Effect**
delay_shots	Puts time delay between shots
no_enemies	Removes all Enemies from the game
no_radar	Removes the radar content
invisible_player	Makes the player invisible
color_X	Changes the color of background to color $X \in$ [black, white, red, blue, green]. This also affects the enemies colors

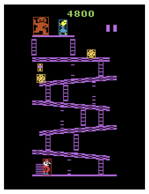
H.11 DemonAttack



Description:
You are facing waves of demons in the ice planet of Krybor. Points are accumulated by destroying demons. You begin with 3 reserve bunkers, and can increase its number (up to 6) by avoiding enemy attacks. Each attack wave you survive without any hits, grants you a new bunker. Every time an enemy hits you, a bunker is destroyed. When the last bunker falls, the next enemy hit will destroy you and the game ends.

Modification	Effect
static_enemies	Makes the enemies horizontally static (static x).
one_missile	Enemies only shoot one missile, instead of three

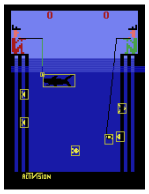
H.12 DonkeyKong



Description:
You play as Mario trying to save Pauline from the hands of Donkey Kong. Remove rivets and jump over the barrels, with a score that starts high and counts down throughout the game.

Modification	Effect
no_barrel	Remove barrels from the game.
unlimited_time	Provides unlimited time for the player.
random_start	Set the players start position to a random pre-defined start position.

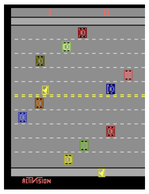
H.13 FishingDerby



Description:
Your objective is to catch more sunfish than your opponent. Watch out for the shark, as it tries to snatch the fish caught on your hook.

Modification	Effect
fish_on_player_side	Spawns all fish on the players side.
fish_in_middle	Spawns all fish in the middle.
fished_on_each_sides1	Swap fish sides, the fish that were on the player side are now on the enemy's one and vice versa.
fished_on_each_sides2	Swap fish sides, the fish that were on the player side are now on the enemy's one and vice versa.
shark_no_movement_easy	The shark will not move and stay on the opponents side.
shark_no_movement_middle	The shark will not move and is placed in the middle.
shark_teleport	The shark will teleport between the player and the enemy side at a set interval.
shark_speed_mode	Increases the sharks movement speed.

H.14 Freeway



Description:
Your objective is to guide your chicken across lane after lane of busy rush hour traffic. You receive a point for every chicken that makes it to the other side at the top of the screen.

Modification	**Effect**
stop_random_car	Stops a random car with a biased probability.
stop_all_cars	Stops all cars and repositions some to predefined positions.
align_all_cars	Aligns the x position of all cars.
reverse_car_speed_bottom	Reverses the speed order of the cars on the bottom road (the fastest becomes the slowest and vice versa).
reverse_car_speed_top	Reverses the speed order of the cars on the top road (the fastest becomes the slowest and vice versa).
speed_mode	Increases the speed of all cars.
invisible_mode	Makes the cars invisible.
phantom_mode	Each car changes color from black to invisible approximately every second.
blinking_mode	Each car changes color randomly approximately every second.
strobo_mode	Each car changes color randomly every timestep.
all_X_cars	Set color of all cars to $X \in$ [black, white, red, blue, green].

H.15 Frostbite

Description:
In Frostbite, the player controls "Frostbite Bailey" who hops back and forth across an Arctic river, changing the color of the ice shards from white to blue. Each time he does so, a block is added to his igloo. Hurry before the polar bear gets you, but watch out for birds that throw you of the shards or crabs that snap you. Look for fish jumping out of the water to receive some bonus points.

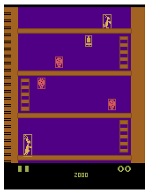
Modification	**Effect**
ui_color_X	Adjust color of the UI to color $X \in$ [black, red]
reposition_floes_easy	Adjusts the position of the ice floes to an easy layout.
reposition_floes_medium	Adjusts the position of the ice floes to a medium layout.
reposition_floes_hard	Adjusts the position of the ice floes to a hard layout.
no_birds	Removes the birds.
few_enemies	Reduces the amount of enemies spawning in.
many_enemies	Increases the amount of enemies spawning in.

H.16 Jamesbond

Description:
Your mission is to control Mr. Bond's specially designed multipurpose craft to complete a variety of missions. The craft moves forward with a right input and slightly back with a left input. An up or down input causes the craft to jump or dive. You can also fire by either lobbing a bomb to the bottom of the screen or firing a fixed angle shot to the top of the screen.

Modification	**Effect**
constant_jump	Makes the player character jump constantly.
fast_backward	Increases the reversing speed.
mobile_player	Makes the player character jump constantly and increases the reversing speed.
straight_shots	The player shots go straight up, instead of diagonal.
unlimited_lives	Player has an unlimited amounts of lives.

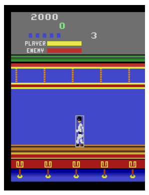
H.17 Kangaroo



Description:
The object of the game is to score as many points as you can while controlling Mother Kangaroo to rescue her precious baby. You start the game with three lives. During this rescue mission, Mother Kangaroo encounters many obstacles. You need to help her climb ladders, pick bonus fruit, and throw punches at monkeys, while avoiding the falling coconuts.

Modification	**Effect**
disable_monkeys	Disables the monkeys in the game.
disable_coconut	Disables the coconuts in the game.
disable_thrown_coconut	Disables the thrown coconut in the game.
no_danger	Combines the three modifications above, disabling all hazards in the game.
set_kangaroo_position_floor1	Sets the kangaroo's position for floor 1.
set_kangaroo_position_floor2	Sets the kangaroo's position for floor 2.
randomize_kangaroo_position	Randomize the floor on which the kangaroo starts.
change_level1	Changes the level to 1.
change_level2	Changes the level to 2.
change_level3	Changes the level to 3.
unlimited_time	Provides unlimited time to clear the level.

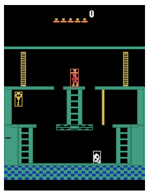
H.18 KungFuMaster



Description:
You are a Kung-Fu master on the mission to rescue your girlfriend from the evil Mr. X. find your way through the castle fighting his lackeys and dodging traps.

Modification	**Effect**
no_damage	Player does not take damage.
unlimited_time	Provides unlimited time to clear the level.
unlimited_lives	Player has an unlimited amounts of lives.

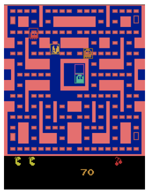
H.19 MontezumaRevenge



Description:
Your goal is to acquire Montezuma's treasure by making your way through a maze of chambers within the emperor's fortress. You must avoid deadly creatures while collecting valuables and tools which can help you escape with the treasure.

Modification	textbfEffect
random_position_start	Randomize the start position within the room
set_level_0	Sets the level to 0.
set_level_1	Sets the level to 1.
set_level_2	Sets the level to 2.
randomize_items	Randomize which item is found in which room.
full_inventory	Adds all items to inventory.

H.20 MsPacman



Description:
Your goal is to collect all of the pellets on the screen while avoiding the ghosts.

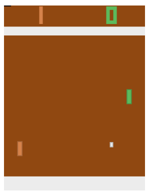
Modification	**Effect**
caged_ghosts	Fix the position of the ghost inside the square in the middle of the screen.
disable_orange	Fix the position of the orange ghost only.
disable_red	Fix the position of the red ghost only.
disable_cyan	Fix the position of the cyan ghost only.
disable_pink	Fix the position of the pink ghost only.
caged_ghosts	Fix the position of all the ghosts.
edible_ghosts	All ghost will be made edible the entire game
set_level_X	Changes the level to $X \in [0, 1, 2]$.
end_game	Simulates an almost done game, with 15 pills remaining in the level.
maze_man	Changes the game to a maze solving task. Only one pill will spawn at a time. After the player collects it, a new pill will spawn. The game is won when the player has collected 20 pills.

H.21 NameThisGame

Description:
You, the diver need to defend your treasure from the giant octopus at the top of the screen. Defend yourself by shooting his tentacles and the shark that is looking to bite you. Remember to refill your oxygen via the the boats tube, before you run out.

Modification	**Effect**
unlimited_oxygen	Provides the player with an unlimited supply of oxygen.
unlimited_lives	Player has an unlimited amounts of lives.
double_wave_length	Doubles the amount of time it takes to get into the next phase.
quick_start	Skips the intro.

H.22 Pong

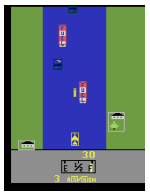


Description:
You control the right paddle and compete against the left paddle controlled by the computer. You each try to keep deflecting the ball away from your goal and into your opponent's goal.

Modification	Effect
lazy_enemy	Enemy does not move after returning the shot until player hits ball.
hidden_enemy	Removes the enemy from the OCAtari object list (Makes it invisible to object based agents).
up_drift	Makes the ball drift upwards.
down_drift	Makes the ball drift down.
left_drift	Makes the ball drift to the left.
right_drift	Makes the ball drift to the right.

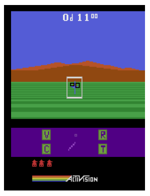
H.23 RiverRaid

Description:
You control a jet that flies over a river: you can move it sideways and fire missiles to destroy enemy objects. Each time an enemy object is destroyed you score points (i.e. rewards). Fly over a fuel depot when you begin to run low, before you lose your jet. You also lose a jet when it collides with the river bank or one of the enemy objects (except fuel depots). The game begins with a squadron of three jets in reserve and you're given an additional jet (up to 9) for each 10,000 points you score.



Modification	Effect
no_fuel	Removes the fuel deposits from the game.
red_river	Turns the river red.
linear_river	Makes the river straight, however objects still spwan at their normal position making them unreachable in the worst case.
exploding_fuels	Shooting the fuel deposits will now provides -80 points (instead of 20).
unlimited_lives	Player has an unlimited amounts of lives.
restricted_firing	The player is only able to shoot in critical situation, facing a bridge or in a corridor.
unlimited_lives	Player has an unlimited amounts of lives.
restricted_firing	The player is only able to shoot in critical situation, facing a bridge or in a corridor.
game_color_change_X	Changes the color of the player to color $X \in [01, 02, 03]$
object_color_change_X	Changes the color of the player to color $X \in [01, 02, 03]$

H.24 RoboTank



Description:
You are playing a tank looking to defeat the enemy squadrons of tanks. Each squadron consists of 12 tanks. Use the radar to find enemy tanks even in the night or during foggy weather. Being hit by the enemy requires you to switch to one of your reserve tanks. Enemy stray shots may destroy vital sensors.

Modification	Effect
fog	Weather condition is always set to fog.
snow	Weather condition is always set to snow.
rain	Weather condition is always set to rain.
tread_damage	Tread sensor is damaged.
canon_damage	Canon is damaged.
vision_damage	Vision sensor is damaged.
no_radar	Disables the radar.

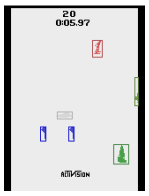
H.25 Seaquest

Description:
You control a submarine able to move in all directions and fire torpedoes. The goal is to rescue as many divers as you can, while dodging and blasting enemy subs and killer sharks; points will be awarded accordingly. The game begins with one sub and three waiting on the horizon. Each time you score 10,000 points, an extra sub will be delivered to your base. You can only have six reserve subs at one time. Your sub will explode if it collides with anything except your own divers. The sub has a limited amount of oxygen that decreases at a constant rate during the game. When the oxygen tank is almost empty, you need to surface and if you don't do it in time, your sub will blow up and you'll lose one diver. Each time you're forced to surface, with less than six divers, you lose one diver as well.

Modification	Effect
unlimited_oxygen	Changes the behavior of the oxygen bar to remain filled
gravity	Enables gravity for the player.
disable_enemies	Disables all the enemies.
random_color_enemies	The enemies have new random colors each time they go across the screen.

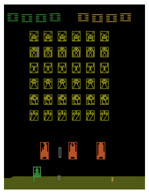
H.26 Skiing

Description:
You control a skier who can move sideways. The goal is to pass through all the gates (between the poles) in the fastest time. You are penalized five seconds for each gate you miss. If you hit a gate or a tree, your skier will jump back up and keep going.



Modification	Effect
invert_flags	Switches the flag color from blue to red.
moguls_to_trees	Replaces all moguls with trees.
moving_flags	Flags move to the left and right.

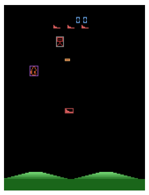
H.27 SpaceInvaders



Description:
Your objective is to destroy the space invaders by shooting your laser cannon at them before they reach the Earth. The game ends when all your lives are lost after taking enemy fire, or when they reach the earth.

Modification	**Effect**
disable_shield_left	Disables the left shield.
disable_shield_middle	Disables the middle shield.
disable_shield_right	Disables the right shield.
disable_shields	Disables all shields.
relocate_shields_right	Relocates the shields to a more right position.
relocate_shields_slight_left	Relocates the shields to a slightly more left position.
relocate_shields_off_by_one	Relocates the shields to to the right by 1 pixel.
relocate_shields_off_by_three	Relocates the shields to to the right by 3 pixels.
controlable_missile	The player can control the missile by moving left and right.
no_danger	Removes disables the enemies shots and removes the shields.

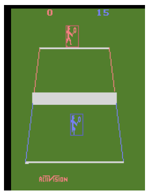
H.28 StarGunner



Description:
You are playing as a stargunner from the Yarthae Empire. The Empire is being invaded by the Sphyzygi. You, the stargunner, must destroy their invading ships while dodging the bombs dropped by the Sphyzygi droid. The droid cannot be destroyed.

Modification	**Effect**
static_bomber	Stops the bomber at the top from moving.
static_flyers	Stops the flying enemies in place.
remove_mountains	Removes the mountains from the game.
static_mountains	The mountains stay the same, even if the player moves.

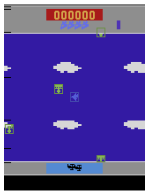
H.29 Tennis



Description:
You control the orange player playing against a computer-controlled blue player. The game follows the rules of tennis. The first player to win at least 6 games with a margin of at least two games wins the match. If the score is tied at 6-6, the first player to go 2 games up wins the match.

Modification	**Effect**
wind_effect	Sets the ball in the up and right direction by 3 pixles every single ram step to simulate the effect of wind
upper_pitches	Changes the ram so that it is always the upper persons turn to pitch.
lower_pitches	Changes the ram so that it is always the lower persons turn to pitch.
upper_player	Changes the ram so that the player is always in the upper field
lower_player	Changes the ram so that the player is always in the lower field

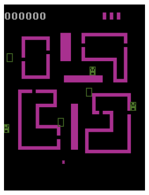
H.30 TimePilot



Description:
You control an aircraft. Destroy your enemies by shooting them down. As you progress through the game, you will encounter enemies with increasingly advanced technology.

Modification	**Effect**
level_X	Changes the level to $X \in [1, 2, 3, 4, 5]$.
random_orientation	Randomizes orientation of enemies. They are no longer aligned.

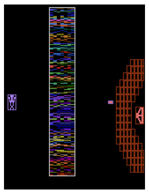
H.31 Venture



Description:
Your goal is to capture the treasure in every chamber of the dungeon while avoiding the monsters.

Modification	**Effect**
enemy_color_X	Changes the color of all enemies to the color $X \in$ [black, white, red, blue, green].
random_enemy_colors	Changes the color of all enemies to a random color.

H.32 YarsRevenge



Description:
The objective is to break a path through the shield and destroy the Qotile with a blast from the Zorlon Cannon.

Modification	**Effect**
disable_enemy_movement	Disables enemy movement.
disable_block_movement	Stops blocks in place.
static	Disables enemy movement and stops blocks in place.