Co-generation of game levels and game-playing agents

Aaron Dharna
Dept. of Computer and Information Science
Fordham University
aadharna@gmail.com

Julian Togelius
Tandon School of Engineering
New York University
julian@togelius.com

L.B. Soros
Tandon School of Engineering
New York University
lsoros@nyu.edu

Abstract

Open-endedness, primarily studied in the context of artificial life, is the ability of systems to generate potentially unbounded ontologies of increasing novelty and complexity. Engineering generative systems displaying at least some degree of this ability is a goal with clear applications to procedural content generation in games. The Paired Open-Ended Trailblazer (POET) algorithm, heretofore explored only in a biped walking domain, is a coevolutionary system that simultaneously generates environments and agents that can solve them. This paper introduces a POET-Inspired Neuroevolutionary System for KreativitY (PINSKY) in games, which co-generates levels for multiple video games and agents that play them. This system leverages the General Video Game Artificial Intelligence (GVGAI) framework to enable co-generation of levels and agents for the 2D Atari-style games Zelda and Solar Fox. Results demonstrate the ability of PINSKY to generate curricula of game levels, opening up a promising new avenue for research at the intersection of procedural content generation and artificial life. At the same time, results in these challenging game domains highlight the limitations of the current algorithm and opportunities for improvement.

Introduction

Humans acquire skills incrementally, e.g. learning to crawl before learning to walk. In this way, primitive skills serve as building blocks for more complex and difficult behaviors. Curricula, which are sets of related tasks or increasingly complicated versions of the same task, can scaffold incremental acquisition of skills for hard problems that are too complex to solve from scratch. However, AI research does not often make use of curricula, instead operating on largely unrelated task domains. Yet, curriculum generation is an important problem because although deep learning algorithms and other recent innovations have achieved landmark performance on historically unsurmounted benchmark domains such as the board game Go (Silver et al. 2016) (which has long served as a grand challenge for artificial intelligence) and the video game Montezuma's Revenge (Ecoffet et al. 2019), the challenge of developing intelligent processes that perform well *in general* remains unmet.

The recent Paired Open-Ended Trailblazer (POET) algorithm (Wang et al. 2019) took an initial step towards more open-ended curriculum generation by evolving parameters for a 2D biped locomotion domain (i.e. slope of hills and placement of obstacles) while simultaneously evolving agent controllers. This coevolutionary system proved capable of generating unique adaptive curricula for learning to walk on uneven terrain. However, it is unknown what other kinds of curricula can be generated coevolutionarily. Games are rich domains for further experiments in curriculum generation because they require critical skills not necessary for bipedal walking, such as long-term planning to avoid enemies. However, many standard video-game-based reinforcement learning (RL) benchmarks are unsuitable for curriculum generation because the games cannot be modified.

This paper describes a novel system called PINSKY that co-generates gameplay agents and levels for games in the General Video Game AI (GVGAI) competition framework. First, necessary background on procedural content generation is reviewed and the POET algorithm is described in full detail. The PINSKY system is then introduced and key differences from original POET (necessary for generating and playing games) are explicitly noted. The results show that PINSKY can co-generate levels and agents for the 2D Zelda- and Solar-Fox-inspired GVGAI games, automatically evolving a diverse array of intelligent behaviors from a single simple agent and game level, but there are limitations to level complexity and agent behaviors. Our analysis suggests reasons for these limitations and directions for further research.

Background

Playing video games with deep networks

Early work on gameplay AI centered around tree search methods such as A* and minimax (Yannakakis and Togelius 2018).The successive development and wider adoption of artificial neural networks allowed further innovation for game-playing AI. Methods for optimizing neural networks generally fall into five categories: supervised learning, unsupervised learning, reinforcement learning, evolutionary approaches, and hybrid learning approaches (Justesen et al. 2020). Reinforcement learning approaches to gameplay in particular generally involve an agent interacting with an environment and repeatedly gaining some amount of reward for its actions. Learning, then, is an optimization process that maximizes long-term reward. Modern RL systems have achieved success in part by incorporating *self-play* (which can be viewed as a form of co-evolution (Arulkumaran, Cully, and Togelius 2019)) into learning schemes, at least for two-player competitive games. In this paradigm, policies being learned are played against each other, with the resulting gameplay data then affecting the trajectory of the learning algorithm. Recent examples of high-performing self-play systems include AlphaGo (Silver et al. 2016), AlphaStar (Vinyals et al. 2019), and OpenAI Five (OpenAI et al. 2019), though it is important to note that these systems all require human gameplay data for initial bootstrapping.

Despite these advances, automated gameplay *in general* is far from solved. Notably, even deep RL systems are still prone to overfitting, meaning trained agents perform poorly on unseen game levels. Justesen et al. (2018) validate this claim empirically and demonstrate that generating levels at an agent-appropriate difficulty level dramatically improves performance on 2D games. However, the level generators for the four games in their study (adapted versions of Zelda, Solar Fox, Frogger, and Boulder Dash) incorporated human-designed elements specific to each game, opening the door for a truly human-free algorithm.

Search-based procedural content generation

Procedural Content Generation (PCG) refers to a variety of methods for algorithmically creating novel artifacts, from static assets such as art and music to game levels and mechanics. Much research is devoted to creating levels that provide adequate challenge and could have plausibly been created by a human level designer. Importantly, the work described in this paper is not focused on creating levels plausibly created by human designers. Instead, it creates game levels that a) satisfy specific playability constraints, b) increase in complexity over time, and c) co-evolve alongside algorithmically-controlled game-playing agents.

Search-based PCG in particular has been theorized to potentially lead to truly endless games (Togelius et al. 2011). The search-based approach requires three primary components: 1) a search algorithm, 2) a content representation, and 3) an evaluation function (Shaker, Togelius, and Nelson 2016). The search algorithm component of such systems frequently (but not always) takes the form of an evolutionary algorithm, wherein a population of content artifacts is created and gradually varied in order to maximize an evaluation function. For the purpose of creating games, the evaluation function can incorporate information from automated gameplay(Togelius et al. 2011). For example, playtraces can be examined for lead changes (Browne and Maire 2010), the capacity of an agent to learn to play the game can be measured (Togelius and Schmidhuber 2008), or the performance of several agents on a game can be compared (Nielsen et al. 2015). In any case, methods designed *only* to optimize objective-oriented fitness metrics can result in incomplete or stagnated search (Lehman and Stanley 2011). In contrast, the work reported in this paper sees one realization of endlessly creating diverse levels for game-playing agents to learn from where the levels grow more complex over time.

The POET Algorithm

The Paired Open-Ended Trailblazer (POET) algorithm (Wang et al. 2019) is a coevolutionary system for concurrently generating and solving new environments. The approach first explored the OpenAI Gym's Hardcore Bipedal Walker domain, wherein environments consist of obstacle-laden hills. Given rangefinder sensors and joint angle information, agents must learn gaits that allow them to walk far over difficult terrain. POET co-evolves agents and terrains through three main processes: 1) periodically generating new environments by mutating existing parents, 2) incrementally optimizing agents paired with environments, and 3) occasionally attempting to transfer optimized agents into new environments. An overview is given in Algorithm 1:

Algorithm 1: POET Algorithm

```
Pair initial environment with unoptimized agent
while not done do
    if counter % mutationTimer == 0 then
        Generate offspring environment-agent pairs
        Remove too-easy and too-difficult offspring
        if population size exceeded then
            Remove oldest environment-agent pairs
        end
    end
    Perform a fixed number of optimization steps
    Re-evaluate all optimized individuals
    if counter % transferTimer == 0 then
        Evaluate all agents on all environments
        Replace incumbent agents with more
          successful agents, if any exist
    end
    counter += 1
end
```

Importantly, generated environments must satisfy a minimal criterion (for viability) that the level is neither too easy nor too hard. The reward function for biped walkers is continuous, allowing "neither too easy nor to hard" to be defined by minimal and maximal acceptable reward values. This binary approach to fitness, explored recently in the context of artificial life and evolutionary robotics (Lehman and Stanley 2010), presents a potentially more open-ended alternative to traditional gradient-based evolution. After an environment satisfies the difficulty criteria, it inherits a copy of its parent's neural controller. Another important and unusual feature of POET is that it periodically evaluates all possible pairs of agents and environments in the population, thereby revealing behaviors that can be easily adapted to multiple environments. Experiments showed that such *transfers* are necessary for solving difficult walking problems. Through incremental optimization and regular transfer of agents, POET generates viable curricula for biped walking.

It should be noted upfront that POET's dynamics are still largely unknown, as few experiments have actually ever been performed. In fact, Wang et al. (2019) reported results from *only three runs* because of the high computational cost.

Methodology

This section primarily describes the novel PINSKY system[1], which adapts the POET algorithm to generating game levels and gameplay agents instead of biped walkers and terrains. PINSKY is composed of three interacting subsystems: 1) the GVGAI game framework, 2) an evolutionary level generator, and 3) an incremental game-playing agent optimizer.

- **GVGAI Framework:** The General Video Game Artificial Intelligence (GVGAI) framework (Perez-Liebana et al. 2018) provides an interface for defining and playing games written in Video Game Description Language (VGDL), which is a text language for 2D games and levels ranging from dungeon crawlers and RPGs to platformers. Two GVGAI games, Zelda and Solar Fox, are explored in this paper. The GVGAI framework affords multiple tracks of interaction with the games including automated level generation and gameplaying. PINSKY uses both capabilities in tandem to build a population of agent-environment pairs that co-evolve over time such that the game levels become more complex while the agents become more proficient (i.e. solving these increasingly complex tasks).
- **Evolutionary Level Generator:** Environment evolution in PINSKY begins with a seed level from which all future levels descend. New offspring levels are generated by mutating tiles on a copy of the parent map. There are three types of possible map mutations, each with separate probabilities: 1) removing a non-player sprite, 2) adding a new sprite, or 3) moving an existing sprite. After each mutation is performed, there is a 50% chance of another mutation occurring. Ultimately, the new map is deemed viable if it can pass a minimal playability criterion check (described later in this section) whereupon the new agent-map pair inherits its parent's neural network and joins the population of actively optimizing environments. However, the new agent-map pair does not replace its parent in the population.
- **Incremental Gameplay Agent Optimizer:** Gameplay agents are controlled by fixed-topology convolutional neural networks, depicted in Figure 2, reducing the problem of finding good agents to a search through connection parameter space. When an agent-offspring pair is initially created via mutation, the offspring agent is an exact copy of the parent agent. Note that, as in the original POET algorithm, optimization occurs incrementally with a fixed number of optimization steps being executed during each main algorithm loop to adapt offspring networks to their new environments. Preliminary experiments investigated a variety of optimizers, including REINFORCE, PPO (Schulman et al. 2017), a simple ES, CMA-ES (Hansen 2007), OpenAI's ES (Salimans et al. 2017), PEPG (Sehnke et al. 2010), and Differential Evolution (DE) (Storn and Price 1995). DE, a population-based optimizer, was selected because of its good convergence properties, ease of implementation, parallelizability, and scalability to high-dimensional problems.

[1]code: tinyurl.com/ydgf64wa

Differences from POET

Games add complexity and diversity The range of game types that even a single game domain can encompass is immense. For example, in vanilla dZelda the fundamental task is to pick up a key and take it to the exit while staying alive. However, given a flexible representation (such as VGDL), the game can trivially be changed into a path-building "connect the dots" game wherein the agent must pick up a key and then find a path connecting all doors. The win conditions of these two possible dZelda varieties are vastly different, highlighting the future potential of the PINSKY system for generating arbitrary games. While diversity of environments is an important goal in its own right, games inherently create the possibility for more complex behaviors than traditional evolutionary robotics domains because winning frequently involves interacting nontrivially with other agents.

Input Difference The POET agent had access to rangefinder readings and information about its own joint angles. In this agent-centric paradigm, each action results only from local state information, allowing no possibility of long-term planning. In contrast, PINSKY agents are given a tile map of the environment as input to their neural networks (Figures 1 and 2) in addition to the agent's orientation. Giving the agent access to global game state and local agent state information allows for more complicated behaviors to emerge. Furthermore, moving away from purely agent-centric network inputs enables the potential generalization of PINSKY to arbitrary games, as most 2D Atari-style games can arguably be represented as some sort of tile map. One of the main side effects of this new tile input is that it reduces the parameter space that the policy network has to work with. Having fewer parameters makes available evolutionary optimization methods that previously were incapable of training policy networks due to not scaling well.

Reward Sparsity The RL problem of credit assignment, or determining the actions responsible for the observed outcome, are hard even when the reward function is dense. When the function is sparse (or distracting) the task becomes

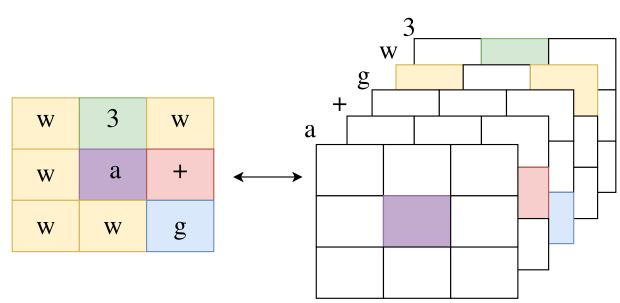


Figure 1: **One-hot encoded map input to the convolutional policy network.** Tiles in each GVGAI map (left) correspond to x,y positions in the environment. In this example from dZelda, possible tiles include (w)all, (g)oal, (a)vatar, key (+), and monster (3). These 2D maps are then extended into a tensor (right) where each slice denotes the presence (indicated by color) or absence of each tile type.

significantly more difficult. Regardless of the optimization routine, which ultimately attempt to solve this credit assignment problem, games such as dZelda in GVGAI are substantially more difficult than the biped walker domains in the OpenAI Gym, which are dense with reward (the single ingredient that differentiates successful members of the optimization population from less successful members). In dZelda the agent is rewarded for picking up a key, taking the key to the door (the win condition), and killing monsters. However, killing enough monsters can eventually provide more reward than winning the game outright. Conversely, in Solarfox the desired task of picking up the coins is precisely what gets rewarded. However in both domains the reward can be sparse because only specific game states (e.g. walking over a key for the first time) earn reward.

Minimum Playability Criteria POET prevents evolutionary search from degenerating by requiring that evolved terrains satisfy a minimal criterion (Lehman and Stanley 2010) defined *a priori*; the walker had to be able to walk at least a minimum amount (ensuring the level is not too hard) and at most a maximum amount (ensuring the level is not too easy). In PINSKY, the minimal criterion concept has been adapted into a playability criterion. Specifically, a level is too easy if a random agent *can* beat the level and too hard if a Monte Carlo Tree Search agent *cannot* beat the level.

Methodology

As a reminder, the initial POET experiments consisted of three runs in a single biped walking domain. Three PINSKY experiments are similarly performed, however each run explores a different game domain and thereby highlights unique capabilities of the novel system. Experimental parameters are in Table 1. While such a small number of runs precludes statistically significant analysis, demonstrating the viability of this new approach to co-generating game levels and gameplaying agents *at all* despite significant computational limitations is worthwhile in its own right.

The first two experiments demonstrate PINSKY performance on two dZelda variants. The first dZelda experiment type (singleDoor) permits only single-door environments, wherein game complexity is increased by adding and rearranging enemies, walls, and keys. The second dZelda experiment type (multiDoor) additionally permits multiple doors in each level, subtly transforming the game from a relatively simple dungeon crawler into a more complex game requiring planning to take one key to *all* doors within the time constraints. All dZelda experiments start with the same seed level (Figure 3a, left).

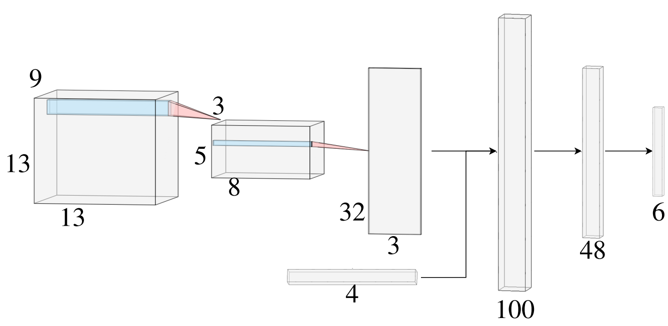


Figure 2: **Dual-input convolutional policy network for dZelda.** As input, the network takes both the one-hot encoded GVGAI-maps (Figure 1) and agent orientation information, then produces an action. The Solarfox network structure is minimally different because Solarfox's map is a different shape and the agent can select from fewer actions.

Argument	Default	Description
game	dZelda	Which GVGAI game to play
gameLen	500	Max agent actions per game
nGames	1500	DE evals per optimization step
popSize	50	Size of DE population
mutationTimer	25	Loops before mutation step
maxChildren	8	Max offspring per mutation step
mutationRate	0.8	Rate of mutating a parent level
transferTimer	10	Loops before transfer attempts
maxEnvs	30	Agent-env. pair population size
numPoetLoops	5000	Max PINSKY loops

Table 1: PINSKY parameters

The third experiment type demonstrates the broad generative potential of PINSKY by additionally investigating the GVGAI game Solarfox. Solarfox differs from dZelda in terms of enemy behaviors; while dZelda enemies move freely and kill on direct contact, Solarfox enemies (exactly two per level) can only move around the level's perimeter, but have projectile attacks. The generated neural networks for playing Solarfox have a slightly different topology than dZelda networks; the tile representation includes a separate sheet for the second enemy character, and the set of actions the agent can take does not include combat, therefore fewer output nodes are required. Furthermore, Solarfox operates on a different movement scheme than the tile-based movement of dZelda. Movement in Solarfox is continuous, where the agent moves in millimeters in the game. In that case, the tile-representation discretizes the space into tiles. Because the Solarfox agent requires many more moves to cross the map than the dZelda agent longer games were needed to ensure the minimum playability criterion could be met (so that MCTS reliably solves human-designed levels).

The potentially open-ended nature of POET-like systems means that each run of the algorithm could, in theory, continue forever. However, practical constraints on computational resources necessarily limit runs. In the original POET experiments, each run lasted 10 days in wall clock time (Wang et al. 2019) while harnessing 256 parallel CPU Cores (with no mention of RAM). The experiments reported herein ran on a 32-core CPU using 50 GB of RAM per experiment.

Results

Table 2 contains summary statistics for three experimental runs. While multiDoor dZelda ran for a full 5000 loops, the other initial runs were truncated to free up computational resources to exploring multiple domains. Specifically, the singleDoor dZelda run was terminated once it displayed substantial generative potential so the Solarfox run could begin. For the purpose of investigating the outputs of PINSKY on complex domains at longer timescales, multiDoor dZelda was allowed to complete its full 5000 main algorithm loops.

Figure 3 depict lineages of generated levels, demonstrating an increase in task complexity over evolutionary time. Successful PINSKY agents in dZelda tend to follow shortest distance paths measured in Manhattan distance. Of course, not all observed agent behaviors are efficient or even effective. Consider an example policy observed on a level nearly identical to the seed level (Figure 3a, left). The degenerate agent takes the key, moves one step, and starts swinging its sword to kill the monster, then keeps swinging forever.

In fact, when playing Solarfox (which has a sparse but non-distracting reward signal), PINSKY agents solve 84.8% of generated levels that passed the playability criterion. For comparison, 64% of playable singleDoor dZelda levels and only 12.7% of multiDoor dZelda levels were solved by PINSKY agents. Therefore, two additional dZelda singleDoor experiments were run for 5000 loops each using a non-distracting reward function rather than the GVGAI built-in reward function. The non-distracting, or *aligned*, reward signal, from Bontrager and Togelius (2020), is:

$$R = \begin{cases} 1 - \frac{n_{steps}}{gameLen} & \text{agent reaches goal} \\ \frac{n_{steps}}{gameLen} - 1 & \text{agent dies} \\ 0 & \text{agent neither reaches goal nor dies} \end{cases}$$

These two additional singleDoor dZelda experiments created results mirroring that of Solarfox where PINSKY generated 1512 and 1251 viable levels and concurrently solved 90% and 83% of viable levels respectively. Similarly when multiDoor dZelda uses the non-distracting reward function, 1344 viable levels were generated of which 29% (compared to 12.7% previously) were solved.

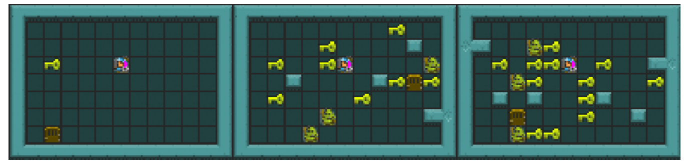
The minimal playability criterion mandates that all generated levels have a MCTS solution before an agent-level pair can be added to the PINSKY population. It is interesting then to note that most, but not all, generated multiDoor dZelda levels remain *unsolved*. The rightmost level in Figure 3b, generated relatively late in its lineage, is an example multiDoor level solved with PINSKY. In that environment, the evolved agent takes an efficient path through the level: down to a key, up to the door above its starting position, down and right to the nearby door, up to the right corner door, and then down to the bottom right door.

In Solarfox, the evolved agent paired with rightmost environment pictured in Figure 3c immediately begins moving to the left (to avoid crashing into the wall) until it is in-between the three clustered coins and has cleared the second wall-fragment. Once there, it moves down to pick up the bottom-most coin, back up to pick up the upper coin, and then further left to pick up the third coin in the small cluster. Finally, it continues to the left until it is partially below the final coin and then moves up until it intersects the coin thereby ending the game.

To investigate whether level difficulty increases over time, PINSKY-solved easy, medium, and hard levels were selected

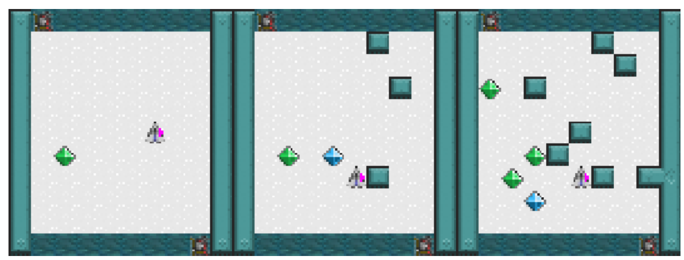
Statistic	singleDoor	multiDoor	Solarfox
Duration	8 days	15 days	7 days
Loops/Generations	2411	5000	3300
Total Levels	768	1600	1056
Viable Levels	684	1353	448
Solved Levels	443	173	380
Transfer attempts	216900	450000	297000
Transfers	3705	8560	730

Table 2: PINSKY run results across three different domains.



(a) singleDoor dZelda from seed (left) to solved environment (right)

(b) multiDoor dZelda lineage snapshots



(c) Solarfox lineage snapshots

Figure 3: Lineage Snapshots across 3 PINSKY experiments with native GVGAI reward schemes, showing increasing difficulty over evolutionary time.

from evolved lineages (e.g. Figure 3) similar to Wang et al. (2020), and concatenated into a curriculum. The agent receives the same amount of optimization time as PINSKY does in each environment. As seen in Table 3, the ability to solve difficult environments tapers off over time even with the behavioral scaffolding afforded by a curriculum. In a separate experiment, the hard levels in Table 3 are optimized with DE from scratch, and given the same number of rollouts as PINSKY, but DE fails to solve these levels. However, the hard Solarfox levels were solved by DE.

Discussion

The results in the previous section show that even with a small curriculum, agents cannot be optimized to solve harder levels independent of the larger PINSKY algorithm. This result then begs an intriguing question: which components enable solving hard levels? Inspecting data from the original

Experiment	Easy	Medium	Hard
Solarfox	✓	✓	✓
singleDoor	✓	✓	X
singleDoor aligned 1	✓	X	X
singleDoor aligned 2	✓	X	X
multiDoor	✓	X	X
multiDoor aligned	✓	X	X

Table 3: **Levels solved using an extracted curriculum**: From each sample lineage (Figure 3) a *PINSKY-solved* easy, medium, and hard level were randomly picked to form a curriculum as in Enhanced POET (Wang et al. 2020). Then optimization was performed using the extracted curriculum, validating the necessity of transfer to solve difficult levels.

PINSKY runs reveals that successful agents were frequently transferred from levels they were not paired with, highlighting the importance of periodic transfer attempts in this coevolutionary system. However, the system still cannot find agents that generalize to solve *all* generated levels, suggesting that more core algorithm innovation is needed.

Evaluating PINSKY on multiple domains with different reward schemes additionally reveals important insights for designing POET-like systems. In particular, the alignment of the agent's reward function with the task to be solved dramatically affects how well the agents solve the generated levels. In dZelda, points are earned for completing the primary goal, but also for semi-related subtasks such as killing monsters. Given enough time, the agent will maximize its score by exclusively completing distracting subtasks. Inversely, if a game has a non-distracting reward signal (like bipedal walking), then PINSKY functions more like POET.

Over time, PINSKY tends to converge with respect to level solvability. To illustrate this phenomenon, consider a multiDoor dZelda level where the agent starts next to monsters. The agent can (and does) solve such levels, but doing so requires highly specific actions (immediately facing and attacking the monster before taking the key to the door). Finding appropriate neural network weights then becomes a search for a needle in a haystack. As levels of an appropriate difficulty become rarer in the active population, the relative frequency of optimization steps increase, allowing the algorithm to create new environments sooner. However, because mutations herein add complexity more than they remove it, there is little incentive for evolving more easier levels.

Despite the potential for convergence, the results in the previous section demonstrate that PINSKY is capable of co-generating lineages of increasingly complex game levels and agents that can play them. That being said, the generated levels are visibly different levels than a human designer would create. For example, PINSKY rarely builds contiguous walls. This particular idiosyncracy could easily be mitigated in an ad hoc manner by modifying the mutation operators in the evolutionary level generator. However, it is interesting for the sake of constructing open-ended generative systems to consider more bottom-up and domain-agnostic methods for incentivizing meaningful design. One possible way to rectify this situation might instead focus on increasing generalizability of agent behaviors; although agents are evaluated on multiple domains when domain transfers are attempted, the system doesn't explicitly reward learning behaviors that solve multiple levels.

It is possible that adding more demanding incentives in this way could encourage the evolution of more challenging environments. This discussion raises the question of why we should even bother generating neural networks when tree search algorithms can already solve the types of Atari-style games explored in this paper. The answer is that using tree search agents would limit the system to environments that have a fast forward model available, excluding most interesting scenarios. The pursuit of generalizable gameplay behaviors is also worthwhile in its own right, and PINSKY may prove to be a useful tool in this regard. Ideally, the key combination of incremental agent optimization with periodic transfer of agents to new environments will result in agents not having time to overfit to their respective environments, which is a phenomenon commonly observed in deep RL (Cobbe et al. 2018). However, for the current approach to be truly successful, network architectures or training methods that generalize better would likely need to be devised.

The pursuit of open-ended evolutionary and generative processes has long been a goal of artificial life research, and the experiments reported in this paper suggest that much can be learned from cross-pollination between these historically disconnected fields. For instance, experiments in a virtual evolving world show that manipulating the minimal viability criterion can speed or slow evolution (Soros, Cheney, and Stanley 2016). Similarly adjusting the viability criterion in a POET-like system would be interesting from an evolutionary dynamics perspective because of the complex interactions between the level generator and the optimizer. It should additionally be noted that the GVGAI framework explicitly makes possible the evolution of game *mechanics*, offering another promising avenue for future work with PINSKY.

Conclusion

This paper adapted the coevolutionary POET algorithm to simultaneously generating game levels and agents that can solve them. Adapting the algorithm to games from its original bipedal walker domain required innovations with respect to key differences from the original algorithm and domain, including enabling more complex environments, giving new kinds of information to gameplay agent controllers, and having an extremely sparse reward function. Results on a limited number of runs demonstrate that the system can, in fact, be adapted to co-generate game levels and game-playing agents while nonetheless illuminating future directions for making the generated levels both more difficult and more solvable. However, it appears that the failure of the trained deep networks to generalize cannot be overcome only by transferring agents from one game level to another.

Acknowledgements

This work was supported by the National Science Foundation. (Award number 1717324 - "RI: Small: General Intelligence through Algorithm Invention and Selection.").

References

Arulkumaran, K.; Cully, A.; and Togelius, J. 2019. Alphastar: An evolutionary computation perspective. *Proceedings of the Genetic and Evolutionary Computation Conference Companion on - GECCO '19*.

Bontrager, P., and Togelius, J. 2020. Fully differentiable procedural content generation through generative playing networks.

Browne, C., and Maire, F. 2010. Evolutionary game design. *IEEE Transactions on Computational Intelligence and AI in Games* 2(1):1–16.

Cobbe, K.; Klimov, O.; Hesse, C.; Kim, T.; and Schulman, J. 2018. Quantifying generalization in reinforcement learning. *arXiv preprint arXiv:1812.02341*.

Ecoffet, A.; Huizinga, J.; Lehman, J.; Stanley, K. O.; and Clune, J. 2019. Go-explore: a new approach for hard-exploration problems.

Hansen, N. 2007. The cma evolution strategy: A comparing review. *Towards a new evolutionary computation. Studies in Fuzziness and Soft Computing* 192:75–102.

Justesen, N.; Torrado, R. R.; Bontrager, P.; Khalifa, A.; Togelius, J.; and Risi, S. 2018. Illuminating generalization in deep reinforcement learning through procedural level generation.

Justesen, N.; Bontrager, P.; Togelius, J.; and Risi, S. 2020. Deep learning for video game playing. *IEEE Transactions on Games* 12(1):1–20.

Lehman, J., and Stanley, K. 2010. Revising the evolutionary computation abstraction: Minimal criteria novelty search. 103–110.

Lehman, J., and Stanley, K. O. 2011. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary computation* 19(2):189–223.

Nielsen, T. S.; Barros, G. A.; Togelius, J.; and Nelson, M. J. 2015. General video game evaluation using relative algorithm performance profiles. In *European Conference on the Applications of Evolutionary Computation*, 369–380. Springer.

OpenAI; Berner, C.; Brockman, G.; Chan, B.; Cheung, V.; Debiak, P.; Dennison, C.; Farhi, D.; Fischer, Q.; Hashme, S.; Hesse, C.; Józefowicz, R.; Gray, S.; Olsson, C.; Pachocki, J.; Petrov, M.; de Oliveira Pinto, H. P.; Raiman, J.; Salimans, T.; Schlatter, J.; Schneider, J.; Sidor, S.; Sutskever, I.; Tang, J.; Wolski, F.; and Zhang, S. 2019. Dota 2 with large scale deep reinforcement learning.

Perez-Liebana, D.; Liu, J.; Khalifa, A.; Gaina, R. D.; Togelius, J.; and Lucas, S. M. 2018. General video game ai: a multi-track framework for evaluating agents, games and content generation algorithms.

Salimans, T.; Ho, J.; Chen, X.; Sidor, S.; and Sutskever, I. 2017. Evolution strategies as a scalable alternative to reinforcement learning.

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

Sehnke, F.; Osendorfer, C.; Rückstieß, T.; Graves, A.; Peters, J.; and Schmidhuber, J. 2010. Parameter-exploring policy gradients. *Neural Networks* 23:551–559.

Shaker, N.; Togelius, J.; and Nelson, M. J. 2016. *Procedural content generation in games*. Springer.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; van den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; Dieleman, S.; Grewe, D.; Nham, J.; Kalchbrenner, N.; Sutskever, I.; Lillicrap, T.; Leach, M.; Kavukcuoglu, K.; Graepel, T.; and Hassabis, D. 2016. Mastering the game of go with deep neural networks and tree search. *Nature* 529:484–503.

Soros, L. B.; Cheney, N.; and Stanley, K. O. 2016. How the strictness of the minimal criterion impacts open-ended evolution. In *Proceedings of the Artificial Life Conference 2016*. MIT Press.

Storn, R., and Price, K. 1995. Differential evolution: A simple and efficient adaptive scheme for global optimization over continuous spaces. *Journal of Global Optimization* 23.

Togelius, J., and Schmidhuber, J. 2008. An experiment in automatic game design. In *2008 IEEE Symposium On Computational Intelligence and Games*, 111–118. IEEE.

Togelius, J.; Yannakakis, G.; Stanley, K.; and Browne, C. 2011. Search-based procedural content generation: A taxonomy and survey. *IEEE Transactions on Computational Intelligence and AI in Games* 3(3):172–186.

Vinyals, O.; Babuschkin, I.; Czarnecki, W. M.; Mathieu, M.; Dudzik, A.; Chung, J.; Choi, D. H.; Powell, R.; Ewalds, T.; Georgiev, P.; Oh, J.; Horgan, D.; Kroiss, M.; Danihelka, I.; Huang, A.; Sifre, L.; Cai, T.; Agapiou, J. P.; Jaderberg, M.; Vezhnevets, A. S.; Leblond, R.; Pohlen, T.; Dalibard, V.; Budden, D.; Sulsky, Y.; Molloy, J.; Paine, T. L.; Gulcehre, C.; Wang, Z.; Pfaff, T.; Wu, Y.; Ring, R.; Yogatama, D.; Wünsch, D.; McKinney, K.; Smith, O.; Schaul, T.; Lillicrap, T.; Kavukcuoglu, K.; Hassabis, D.; Apps, C.; and Silver, D. 2019. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature* 575(7782):350–354.

Wang, R.; Lehman, J.; Clune, J.; and Stanley, K. O. 2019. POET: Open-ended coevolution of environments and their optimized solutions. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '19, 142–151. ACM.

Wang, R.; Lehman, J.; Rawal, A.; Zhi, J.; Li, Y.; Clune, J.; and Stanley, K. O. 2020. Enhanced poet: Open-ended reinforcement learning through unbounded invention of learning challenges and their solutions.

Yannakakis, G. N., and Togelius, J. 2018. *Artificial intelligence and games*. Springer.