thus establishing an evolutionary arms race. While current multiagent and multipopulation systems still largely focus on solving a single task, evolution in such domains has already been shown to lead to specialization and discovery of cooperation, which could lead to major transitions. Multitask and multiobjective evolution is already known to result in more robust solutions, and in such environments could lead to progressive development of general intelligence. Perhaps most promising avenue is to have the agents themselves modify the environment, building artifacts and complexity into it that persists (Lehman et al. 2022). In this manner, the environment and the agents in it can complexify indefinitely.

What goals might such experiments be set to achieve? An important one is a better understanding of biological evolution, i.e. the origins of major transitions and intelligence. Another one is to construct better artificial systems, i.e. systems that can be deployed in natural environments and social environments where they adapt to existing challenges and changes to them indefinitely–much like people do. Such ability is one essential ingredient in artificial general intelligence.

To make these ideas concrete, the next two subsections review two concrete experiments on changing environments. In the first one, the body of the agent is co-evolved with the brain in a cooperative manner. In the second, the environment is coevolved to provide more difficult challenges in a competitive manner.

9.2 Cooperative Coevolution of Body and Brain

In Section 3.2, we explored the simple idea of evolving the weights of a neural network to control a bipedal walker, and showed how to use evolution algorithms in the context of neural networks to learn a set of weights that can allow a neural network with such weights to perform a given task. If such a network is an agent operating in an environment, we have so been evolving a policy to manipulate an agent, whose design is fixed, to maximize some notion of cumulative reward. The design of the agent's physical structure is rarely optimized for the task at hand. Unlike gradient-based methods, evolution algorithms are more flexible and thus can optimize parameters beyond the weights of a neural network agent. In principle, we can even optimize parameters governing the agent as well. Why constraint ourselves to weights, when we can also optimize other important design choices governing our agents?

In this section, we explore the possibility of learning a version of the agent's design that is better suited for its task, jointly with the policy. We look at a minor modification to the environment, where we parameterize parts of an environment, and allow an agent to jointly learn to modify these environment parameters along with its policy. We show that an agent can learn a better structure of its body that is not only better suited for the task, but also facilitates policy learning. Joint learning of policy and structure may even uncover design principles that are useful for assisted-design applications.

In addition to the weight parameters of our agent's policy network, the agent's environment is also parameterized, which includes the specification of the agent's body structure. This extra parameter vector, which may govern the properties of items such as width, length, radius, mass, and orientation of an agent's body parts and their joints, will also be treated as a learnable parameter. Hence the weights w we need to learn will be the parameters of

the agent's policy network combined with the environment's parameterization vector. During a rollout, an agent initialized with w will be deployed in an environment that is also parameterized with the same parameter vector w.

```
def rollout(agent, env):
  obs = env.reset()
  done = False
  cumulative_reward = 0
  while not done:
    a = agent.action(obs)
    obs, r, done = env.step(a)
    cumulative_reward += r
  return cumulative_reward
```

```
def rollout(agent, env_params, env):
  env.augment(env_params)
  obs = env.reset()
  done = False
  cumulative_reward = 0
  while not done:
    a = agent.action(obs)
    obs, r, done = env.step(a)
    r = augment_reward(r, env_params)
    cumulative_reward += r
  return cumulative_reward
```

Figure 9.5: Gym framework for rolling out an agent in an environment (left; (Brockman et al. 2016)). We propose an alteration where we parameterize parts of an environment, and allow an agent to modify its environment before a rollout, and also augment its reward based on these parameters (right).

The goal is to learn w to maximize the expected cumulative reward, $E[R(w)]$, of an agent acting on a policy with parameters w in an environment governed by the same w. In our approach, we search for w using PGPE discussed earlier.

Armed with the ability to change the design configuration of an agent's own body, we also wish to encourage the agent to challenge itself by rewarding it for trying more difficult designs. For instance, carrying the same payload using smaller legs may result in a higher reward than using larger legs. Hence the reward given to the agent may also be augmented according to its parameterized environment vector.

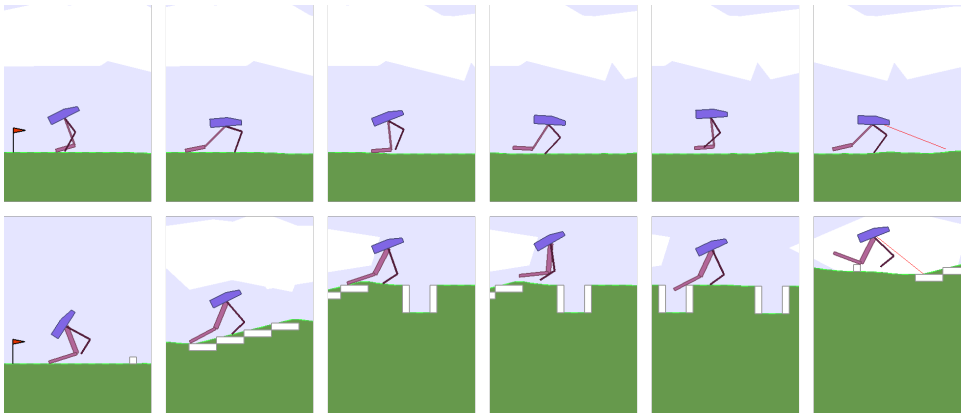


Figure 9.6: **Examples of evolved morphology.** Biped + Morphology (top) develops a thicker but short rear lower limb for the easy flat ground version of the environment. Biped Hardcore + Morphology (bottom) develops a larger rear leg. Here, its thigh is larger than the lower limb.

Learning a better version of an agent's body not only helps achieve better performance, but also enables the agent to jointly learn policies more efficiently. In this environment,

the agent generally learns to develop longer, thinner legs, with the exception in the rear leg where it developed a thicker lower limb to serve as a useful stability function for navigation. Its front legs, which are smaller and more maneuverable, also act as a sensor for dangerous obstacles ahead that complement its LIDAR sensors. While learning to develop this newer structure, it jointly learns a policy to solve the task in 30% of the time it took the original, static version of the environment.

Allowing an agent to learn a better version of its body obviously enables it to achieve better performance. But what if we want to give back some of the additional performance gains, and also optimize also for desirable design properties that might not generally be beneficial for performance? For instance, we may want our agent to learn a design that utilizes the least amount of materials while still achieving satisfactory performance on the task. Here, we reward an agent for developing legs that are smaller in area, and augment its reward signal during training by scaling the rewards by a utility factor of $1 + \log(\frac{\text{orig leg area}}{\text{new leg area}})$. This encourages development of smaller legs.

When rewarded for small leg size, the agent learned a lean minimal design where every inch matters. It also learned movements that appear more insect-like. Here, the agent learns the smallest pair of legs that still can solve BipedalWalkerHardcore-v2 (Figure 9.7).

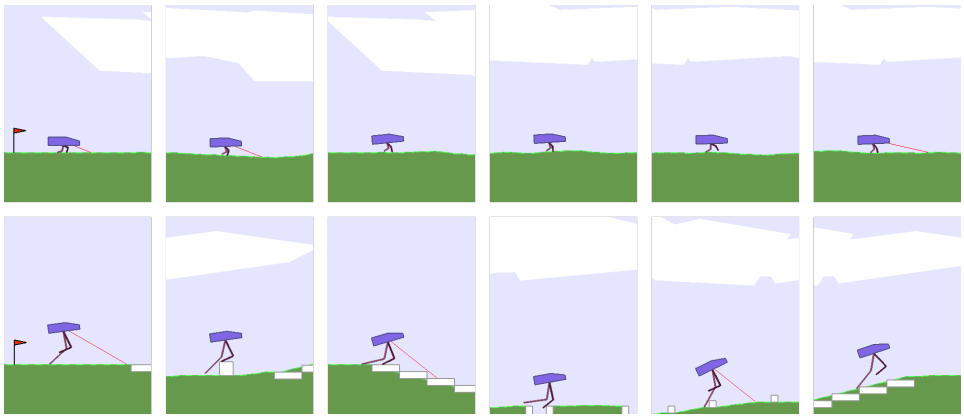


Figure 9.7: Agent rewarded for smaller legs for the easier flat version of the task (top). Agent learns the smallest pair of legs that still can solve `BipedalWalkerHardcore` (bottom).

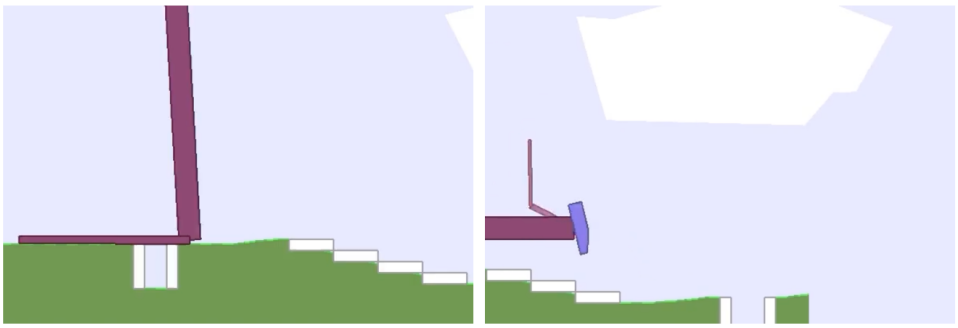


Figure 9.8: If we remove all design constraints, the optimizer came up with a really tall bipedal walker robot that "solves" the task by simply falling over and landing near the exit!

We demonstrated more interesting, life-like results, when we added additional constraints such as also optimizing for lighter and smaller agent legs. What if we do the opposite and remove these constraints altogether? Without any design constraints, it develops an extremely tall bipedal walker agent that "solves" the task by simply falling over and landing at the exit (Figure 9.8). In this manner, body-brain coevolution provides an avenue for openended discovery of better solutions. As the agent gets better at controlling the body, it can become more efficient. An alternative is to coevolve the task with the brain: as the agent gets better, it faces more difficult challenges in an open-ended manner. This is the topic of the next section.

9.3 Competitive Coevolution of Environments and Solutions

9.3.1 The Influence of Environments

Our thought processes and behaviors are significantly influenced by the specific time and place we inhabit on earth. These elements are shaped by distinct circumstances, cultural understandings, prevailing beliefs, and local customs. Together, they create a framework that both defines and restricts our experiences and the patterns of our thoughts Ryan Ruggiero 2012. Take the concept of individualism versus collectivism for example, which varies widely across cultures. In many Western societies, such as the United States, there's a strong focus on individual achievement and independence. This cultural context fosters a thought pattern that emphasizes personal goals and self-reliance. In contrast, many Eastern societies, like Japan, emphasize collectivism, where the focus is on group harmony and community. In such cultures, thought patterns and behaviors are more aligned with group goals and the collective well-being. Inhabiting a different era or being part of a distinct culture would fundamentally transform who we are, reshaping our identity in profound ways.

This principle that humans are shaped by their environments applies similarly to AI and ML systems. For example, large language models (LLMs) such as BERT, GPT-4, and Llama2 are deeply influenced by their training data. If trained on scientific literature, the model will excel in technical explanations, whereas training on conversational texts results in more colloquial responses. This extends to the biases and perspectives inherent in the data. Similarly, in image generation, diffusion models produce different outputs based on their training datasets: models trained on classical art will generate different images than those trained on modern digital art. In the realm of reinforcement learning (RL), the training environment crucially defines an agent's skills. For instance, an agent trained in a simulated urban setting will develop different capabilities and strategies compared to one trained in a virtual natural landscape.

Just as human experiences are shaped by our environments and cultures, AI and ML agents are similarly molded by their training contexts and data environments. The quality and diversity of their training inputs are crucial, emphasizing the importance of co-evolving AI systems with their environments to enhance their capabilities, behaviors, and ethical alignment. This approach mirrors human development and extends it into the digital realm.

9.3.2 Co-Evolving Agents and Their Environments

In exploring the co-evolution of agents and their environments, it is pivotal to differentiate between static and dynamic environmental settings. Static environments, such as the terrain a bipedal walker traverses, provide consistent, unchanging challenges that evolve gradually with the agent's abilities. This approach is highlighted in our first example below, where the walking skill of a bipedal robot is simultaneously evolved with the complexity of the terrain it must navigate. Conversely, dynamic environments involve interactions with changing elements within the system, exemplified by our second example where a quadrupedal robot's locomotion controller evolves in response to the locomotion skills of a dynamically adapting opponent. These examples illustrate how AI development benefits from environments that not only challenge but evolve in response to the AI.

9.3.2.1 Paired Open-Ended Trailblazer (POET)

POET R. Wang et al. 2019a is an algorithm designed to simulate an ongoing, open-ended process of discovery within a single run, by co-evolving environments and their respective agents. It begins with an initial simple environment, such as a flat-ground obstacle course, paired with a randomly initialized neural network agent. Throughout its operation, POET executes three core tasks within its main loop:

- **Environment Generation** POET generates new environments by mutating the parameters of existing ones. The parameters of the environment includes (1) stump height, (2) gap width, (3) stair height, (4) number of stairs, and (5) surface roughness. This process is selective, adding new environments to the active population only if they provide a suitable challenge and introduce novelty. For example, in their paper a minimum criterion (MC) of $S_{\min} < E^{\text{child}}(\theta^{\text{child}}) < S_{\max}$, where $S_{\min}$ and $S_{\max}$ are pre-defined scores thresholds, is used to filter out child environments that appear too challenging or too trivial, yet fostering a diverse range of challenges.
- **Agent Optimization** Each agent is continuously optimized within its environment using evolutionary strategies, though other reinforcement learning methods could also apply. The objective is to maximize performance metrics relevant to each environment, such as traversing an obstacle course efficiently. This optimization happens independently for each pair, which facilitates parallel processing and enhances computational efficiency.
- **Agent Transfer** To foster cross-environment adaptation, POET attempts to transfer agents between different environments. This strategy can help agents escape local optima by applying successful strategies from one context to another. For example, an agent performing well in a mountainous terrain might offer insights when transferred to a rocky terrain, potentially leading to breakthroughs in performance.

POET maintains a controlled number of environment-agent pairs in its active list, capped at a maximum size to manage computational resources. Environments that become obsolete or overly familiar are phased out to make room for new ones, ensuring the population remains dynamic and conducive to continuous learning.

Results

POET operates on the hypothesis that solutions to complex challenges are more likely to be found through a divergent, open-ended process rather than direct optimization. This is

illustrated by the fact that when directly optimizing agents using ES in environments created by POET, the agents often prematurely converge to suboptimal behaviors.

Experiments conducted by POET using different types of obstacles (such as gaps, rough terrain, and stumps) reveal that challenges generated and solved by POET are far too difficult for ES when tackled directly, see Figures 9.9 and 9.10. For example, agents optimized by ES in these environments tend to stop and avoid moving further to prevent penalties rather than learning to navigate obstacles effectively. This behavior contrasts starkly with the capabilities developed by agents under POET, which successfully navigate these complex environments. Additional results highlight that POET not only engineers these challenging environments but also devises innovative solutions that ES alone cannot achieve. This includes agents developed by POET that can navigate wide gaps and rugged terrains which ES agents fail to handle. In simpler environments also created by POET, ES consistently underperforms, unable to match the high standards set by POET's adaptive and dynamic approach.

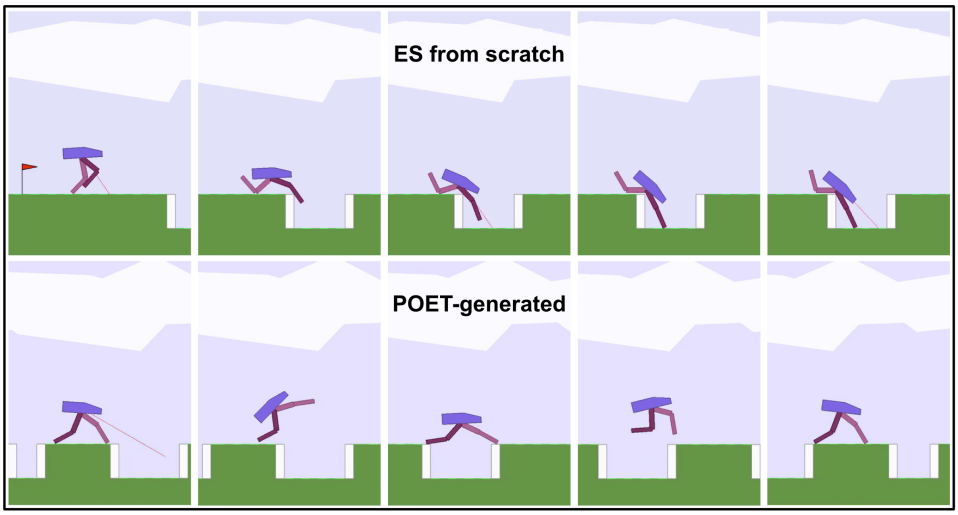
From these results, it is evident that POET exemplifies the principle of co-evolution between agents and their environments. As an automatic curriculum builder, POET continuously creates new challenges that are optimally balanced, neither too easy nor too hard, effectively teaching agents how to tackle increasingly complex problems. This co-evolutionary process fosters an environment where skills developed in one context are not only honed but also become transferable, aiding agents in solving new and more complex challenges.

9.3.2.2 Learning to Chase-and-Escape

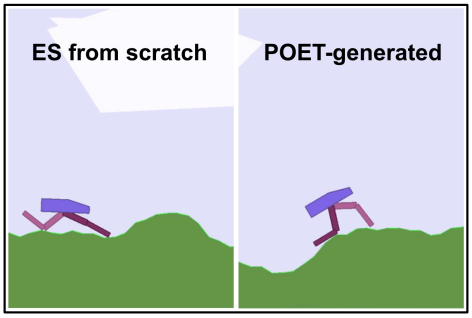
Learning to chase and escape Tang, Tan, and Harada 2020 provides an illustrative example of co-evolution in a dynamic environment. In this study, both the chaser and the escapee are robots that co-evolve: as the chasing robot develops increasingly sophisticated tactics, the robot being chased concurrently refines its skills to evade capture. This dynamic interaction exemplifies how both agents adaptively improve their strategies in response to each other's evolving capabilities.

There are two robots in the paper's settings, where the chaser is a quadrupedal robot that needs to learn low-level joint commands (i.e., desired joint angles), and the escapee is a dot robot that learns swift commands (i.e., desired velocities and directions). The escapee is said to be caught if the distance between the two robots is less than a predefined threshold $d_{\min}$. The two robots are trained in an iterative fashion:

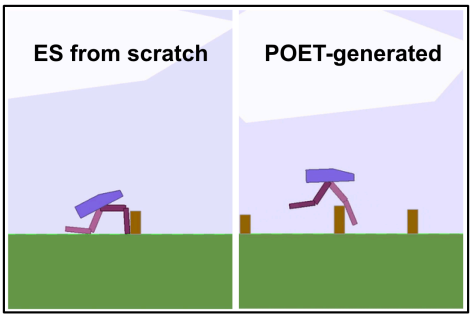
- **Learning to Chase** In each iteration, the chaser robot plays against an opponent that is randomly sampled from an adversary pool Π_a. The pool initially only contains an escapee robot that stays still, this gives the chaser robot to learn basic locomotion skills in early stages.
- **Learning to Escape** After the chaser robot's control policy is evolved, an opponent robot plays against the upgraded version of the chaser. The escapee robot has no memory of the skills it previous learned, and will devote all its energy and capacity in learning new skills that discovers and exploits the weakness of the chaser robot's locomotion capability. After learning, this escapee robot's policy is added to Π_a.



(a) Generated agents attempting gaps



(b) Generated agents on rough surfaces



(c) Generated agents attempting stumps

Figure 9.9: POET generates complex environments and effective agent solutions unachievable through standard ES. As depicted, agents optimized directly by ES (top row of panel (a) and left panels of (b) and (c)) tend to develop suboptimal behaviors, often quitting prematurely. In contrast, POET not only engineers these demanding scenarios but also successfully trains agents that adeptly navigate through them, as demonstrated in the bottom row of panel (a) and the right panels of (b) and (c)). Figure from (R. Wang et al. 2019a).

- **Encouraging Behavioral Diversity** While having the adversary pool Π_a encourages the chaser robot to play against various escapees and helps fight catastrophic forgetting, the diversity in the escapee robots' escaping maneuvers is also critical. To achieve this, the authors sampled different d_{min} when training the escapee robots. Intuitively, a small distance threshold allows the escapee to stay close to the chaser, and develop sudden quick movements to dodge, while larger values would encourage the escapee to use large circular trajectories to stay away from the chaser.

This iterative co-evolution between the chaser and escapee robots is critical in developing their agility and robustness. Each cycle of adaptation not only hones their individual strategies but also contributes to a richer, more responsive interaction between them. By

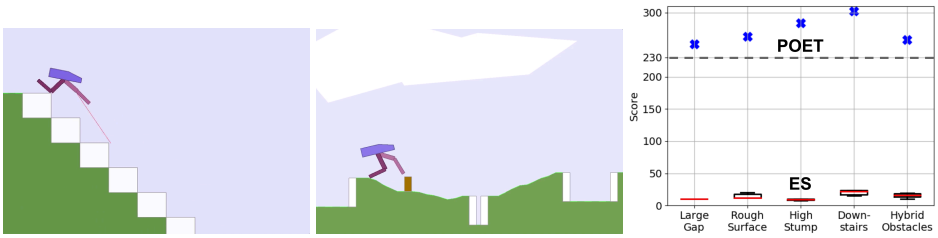


Figure 9.10: Agents demonstrate advanced navigation abilities in complex scenarios engineered by POET. Notable challenges include (Left) navigating exceptionally large steps and (Middle) mastering a rough terrain course featuring a mix of narrow and wide gaps, alongside stumps of varying heights. In addition, ES alone fails to match POET's performance in various settings. (Right) A dotted line at a score of 230 indicates the success threshold. The plots clearly show that ES consistently falls short of meeting the challenges effectively addressed by POET. Figure from (R. Wang et al. 2019a).

continuously evolving both agents and the dynamics of their environment, the study showcases how the complexity and effectiveness of autonomous systems can be significantly enhanced.

Results

After training, the quadrupedal chaser robot develops a symmetric gait that alternates between its forelimbs and hind limbs, mimicking the bounding gait commonly seen in quadruped animals at high speeds. To execute sharp turns, it extends the stance phase of one forelimb, using it as a pivot to rapidly rotate its body and change direction. Additionally, the escapee robot demonstrates sophisticated maneuvers, such as sprinting at full speed, circling to confuse the chaser, and employing sudden lateral dodges to cause the chaser to overshoot. For visual examples of these dynamic interactions, refer to Figure 9.11, which illustrates the trajectories of both the chaser and escapee robots.

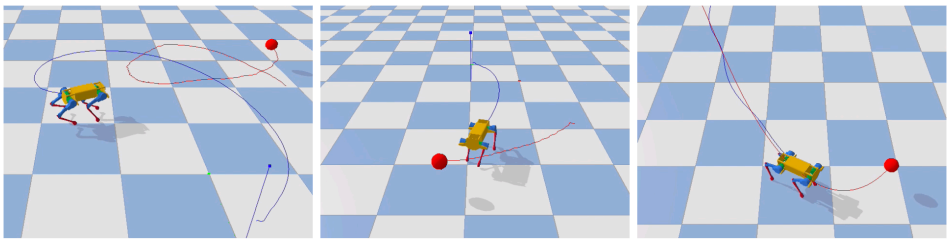


Figure 9.11: **Sample episodes of chase and escape**. The quadruped robot is the chaser and the red dot-bot is the escapee; the blue and red lines are their trajectories. In the experiments, some adversarial agents learned advanced escaping maneuvers such as induce the quadruped robot to come close, dodge and halt so that the quadruped robot runs past them. (Figures from Tang, Tan, and Harada 2020)

To illustrate the advantages of co-evolutionary methods over static training environments, three inductive bias-driven baseline methods are presented and depicted in the top row of Figure 9.12: (1) **Cone Configuration** where a target position is randomly selected within a fan-shaped area directly ahead of the chaser robot, simulating a forward-focused pursuit, denoted as π_{cone}. (2) **Circular Configuration** where the target is randomly placed anywhere within a complete circular area surrounding the chaser, promoting omnidirectional movement, denoted as π_{circle}. (3) **Zigzag Configuration** where targets are alternately placed to the left and right directly in front of the chaser, encouraging it to adopt a zigzagging movement pattern, denoted as π_{zigzag}. Additionally, to underscore the importance of diversity in training, a scenario in which the chaser robot plays against a single evolved opponent is included for comparison, denoted as π_{single}.

These configurations are employed to benchmark the performance of traditional methods against those that dynamically co-evolve the training environment alongside the agent. The bottom row of Figure 9.12 illustrates the trajectories of all chaser policies as they attempt to intercept a target moving along a sine-shaped route. In the first two cases, the co-evolved policy successfully intercepts the target even before it reaches the first turn. In contrast, the policies trained with the baseline configurations either fall behind or require more time to catch up. When the target maneuvers through turns (as shown in the last two plots), the co-evolved policy adeptly follows the trajectory and captures the target, whereas the baseline policies struggle, often losing balance or needing to slow down significantly to manage the turn. This stark contrast highlights that the co-evolution of the agent and the environment is crucial for achieving superior performance, as it allows the agent to adapt more effectively to complex and dynamic challenges.

9.4 Chapter Review Questions

1. **Key Ingredients:** What are the five elements of biological open-endedness that could potentially inspire open-ended neuroevolution, and how do they support continuous innovation?
2. **Neutral Mutations:** Why are neutrality and weak selection crucial for maintaining diversity in large populations, and how does such processes differ from traditional approaches in evolutionary computation?
3. **Role of Extinctions:** How can extinction events accelerate evolution and increase evolvability in computational experiments? Provide an example e.g. from the bipedal walker domain.
4. **Long-Term Effects:** Describe how repeated extinction events can lead to populations that are more evolvable and capable of filling niches more effectively.
5. **GRNs and Evolvability:** How do Genetic Regulatory Networks (GRNs) provide a substrate for evolvability, and what advantages do they offer compared to direct encodings in tasks like Nothello?
6. **Indirect Encodings:** Explain the role of indirect encodings in enhancing evolvability. How do GRNs contribute to the discovery of robust and diverse neural network motifs?
7. **Miracle Jumps:** What are "miracle jumps," and why are expressive encodings (e.g., GP or neural networks) more effective than direct encodings in achieving such jumps?

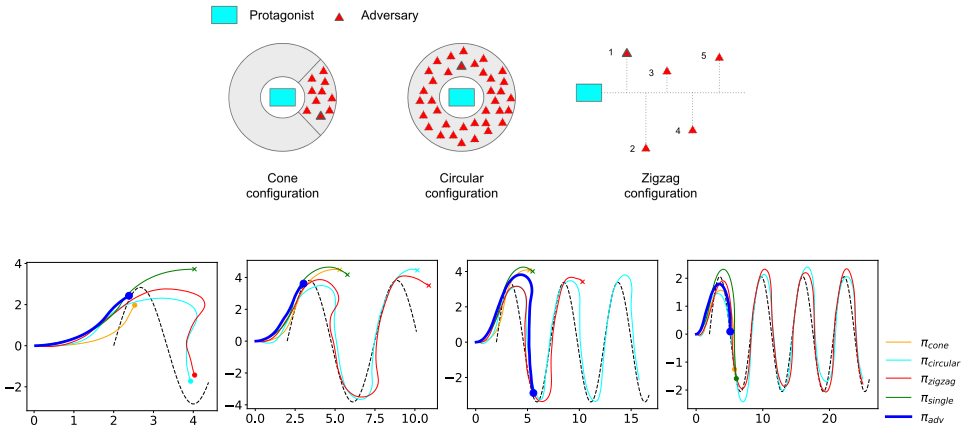


Figure 9.12: **Comparison with baseline methods.** (Top) Three configuration of initial positions for a static adversary. (Bottom) Trajectories of the methods when the chaser robot tries to catch an escapee robot that moves along a sine-wave shaped route. A cross at the end of a trajectory indicates that the chaser has fallen or the target has escaped. A dot at the end means successfully catching the target at that position. Short trajectories ending with dots indicate the chaser catches the target early. The chaser trained with dynamic adversaries (blue trajectory) is able to catch the target much earlier than other baseline policies, including the policy that plays against a single opponent (π_{single}). Figure from (Tang, Tan, and Harada 2020).

8. **Comparative Power:** Compare the benefits of expressive encodings with traditional evolutionary algorithms for solving problems with dynamically changing objectives.
9. **Body-Brain Coevolution:** How does coevolving an agent's body and brain lead to better solutions, and what principles can it reveal about designing efficient and specialized morphologies?
10. **Environment-Agent Coevolution:** Describe the core mechanisms of the POET algorithm for coevolving agents and environments. How does it differ from static training environments, and why is this approach effective for solving complex challenges?