

Algorithm 1: QD-PG**Given:** N , max_steps , $\text{gradient_steps_ratio}$, BD extraction function ξ , state descriptor extraction function ψ **Initialize:** MAP-Elites grid \mathbb{M} , Replay Buffer \mathbb{R} , N actors $\{\pi_{\theta_i}\}_{i=\{1,\dots,N\}}$, 2 critics Q_w^D and Q_w^Q , state descriptors archive \mathbb{A}

```

total_steps, actor_steps = 0, 0 // Step counters

// Parallel evaluation of the initial population
for j ← 1 to N do
  Play one episode with actor  $\pi_{\theta_j}$  and store all transitions in  $\mathbb{R}$ 
  Get episode length  $T$ , discounted return  $R$  and state descriptors  $\{\psi(s_1), \dots, \psi(s_T)\}$ 
  Store state descriptors  $\{\psi(s_1), \dots, \psi(s_T)\}$  in  $\mathbb{A}$ 
  Compute  $\xi(\theta_j)$  and add the tuple  $(R, \xi(\theta_j), \theta_j)$  in the MAP-Elites grid  $\mathbb{M}$ 
  actor_steps ← actor_steps + T
end

// Main loop
while total_steps < max_steps do
  // Select new generation
  Get  $N$  actors  $\pi_{\theta_i}, i \in \{1, \dots, N\}$  from  $\mathbb{M}$ 
  gradient_steps = int(actor_steps × gradient_steps_ratio)
  actor_steps = 0

  // Perform in parallel population update and evaluation
  for j ← 1 to N do
    // Update the population
    for i ← 1 to gradient_steps do
      Sample batch of  $(s_t, a_t, r_t, s_{t+1}, \psi(s_t))$  from  $\mathbb{R}$ 

      // First half is updated to maximise diversity
      if j ≤ N//2 then
        Compute novelty reward as  $r_t^D$  from  $\psi(s_t)$  and  $\mathbb{A}$ 
        Update  $\pi_{\theta_j}$  for diversity
        Compute the novelty critic gradient locally
        Average novelty critic gradients between threads
        Update novelty critic  $Q_w^D$ 
      end

      // Second half is updated to maximise quality
      else
        Update  $\pi_{\theta_j}$  for quality
        Compute the quality critic gradient locally
        Average quality critic gradients between threads
        Update quality critic  $Q_w^Q$ 
      end
    end
  end

  // Evaluate the updated actors
  Play one episode with actor  $\pi_{\theta_j}$  and store all transitions in  $\mathbb{R}$ 
  Get episode length  $T$ , discounted return  $R$  and state descriptors  $\{\psi(s_1), \dots, \psi(s_T)\}$ 
  Store state descriptors  $\{\psi(s_1), \dots, \psi(s_T)\}$  in  $\mathbb{A}$ 
  Compute  $\xi(\theta_j)$  and add the tuple  $(R, \xi(\theta_j), \theta_j)$  in the MAP-Elites grid  $\mathbb{M}$ 
  actor_steps ← actor_steps + T
end

total_steps ← total_steps + actor_steps // Update total time steps
end

```

520 B The TD3 Agent

521 The Twin Delayed Deep Deterministic (TD3) agent Fujimoto et al. (2018) builds upon the Deep
522 Deterministic Policy Gradient (DDPG) agent Lillicrap et al. (2015). It trains a deterministic actor
523 $\pi_\phi : \mathcal{S} \rightarrow \mathcal{A}$ directly mapping observations to continuous actions and a critic $Q_\theta : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ which
524 takes a state s and an action a and estimates the average return from selecting action a in state s and
525 then following policy π_ϕ . As DDPG, TD3 alternates between policy evaluation and policy improvement
526 so as to maximise the average discounted return. In DDPG, the critic is updated to minimize a temporal
527 difference error during the policy evaluation step which induces an overestimation bias. TD3 corrects
528 for this bias by introducing two critics Q_{θ_1} and Q_{θ_2} . TD3 plays one step in the environment using its
529 deterministic policy and then stores the observed transition (s_t, a_t, r_t, s_{t+1}) into a replay buffer \mathcal{M} .
530 Then, it samples a batch of transitions from \mathcal{M} and updates the critic networks. Half the time it also
531 samples another batch of transitions to update the actor network.

532 Both critics are updated so as to minimize a loss function which is expressed as a mean squared error
533 between their predictions and a target:

$$L^{critic}(\theta_1, \theta_2) = \sum_{\text{batch } i=1,2} \sum (Q_{\theta_i}(s_t, a_t) - y_t)^2, \quad (6)$$

534 where the common target y_t is computed as:

$$y_t = r_t + \gamma \min_{i=1,2} Q_{\theta_i}(s_{t+1}, \pi_\phi(s_{t+1})) + \epsilon, \quad (7)$$

535 where $\epsilon \sim \mathcal{N}(0, I)$.

536 The Q-value estimation used to compute target y_t is taken as minimum between both critic predictions
537 thus reducing the overestimation bias. TD3 also adds a small perturbation ϵ to the action $\pi_\phi(s_{t+1})$ so
538 as to smooth the value estimate by bootstrapping similar state-action value estimates.

539 Every two critics updates, the actor π_ϕ is updated using the deterministic policy gradient also used
540 in DDPG Silver et al. (2014). For a state s , DDPG updates the actor so as to maximise the critic
541 estimation for this state s and the action $a = \pi_\phi(s)$ selected by the actor. As there are two critics
542 in TD3, the authors suggest to take the first critic as an arbitrary choice. The actor is updated by
543 minimizing the following loss function:

$$L^{actor}(\phi) = - \sum_{\text{batch}} Q_{\theta_1}(s_t, \pi_\phi(s_t)). \quad (8)$$

544 Policy evaluation and policy improvement steps are repeated until convergence. TD3 demonstrates
545 state of the art performance on several MUJoCo benchmarks.

546 C QD-PG Details

547 C.1 Computational details

548 We consider populations of $N = 4$ actors for the POINT-MAZE environment and $N = 10$ actors for
549 ANT-MAZE and ANT-TRAP. We use 1 CPU thread per actor and parallelization is implemented with
550 the Message Passing Interface (MPI) library. Our experiments are run on a standard computer with
551 10 CPU cores and 100 GB of RAM, although the maximum RAM consumption per experiment at
552 any time never exceeds 10GB due to an efficient and centralized management of the MAP-Elites grid
553 which stores all solutions. An experiment on POINT-MAZE typically takes between 2 and 3 hours
554 while an experiment on ANT-MAZE or ANT-TRAP takes about 2 days. Note that these durations can
555 vary significantly depending on the type of CPU used. We did not use any GPU.

556 Computational costs of QD-PG mainly come from backpropagation during the update of each agent,
557 and to the interaction between agents and the environment. These costs scale linearly with the
558 population size but, as many other population-based methods, the structure of QD-PG lends itself
559 very well to parallelization. We leverage this property and parallelize our implementation to assign

560 one agent per CPU thread. Memory consumption also scales linearly with the number of agents.
 561 To reduce this consumption, we centralize the MAP-Elites grid on a master worker and distribute
 562 data among workers when needed. With these implementation choices, QD-PG only needs a very
 563 accessible computational budget for all experiments.

564 C.2 MAP-Elites Implementation details

565 QD-PG uses a MAP-Elites grid as archive of solutions. We assume that the BD space is bounded and
 566 can be discretized into an Cartesian grid. We discretize each dimension into m meshes, see Table 2
 567 for the value of m depending on the environment. Hence, the number of cells in the MAP-Elites
 568 grid equals m times the number of dimensions of the BD space. When a new solution θ is obtained
 569 after the mutation phase, we look for the cell corresponding to its BD, $\xi(\theta)$. If the cell is empty, the
 570 solution is added, otherwise the new solution replaces the solution already contained in the cell if its
 571 score $F(\theta)$ is better than the score of the already contained solution. During selection, we sample
 572 solutions uniformly from the MAP-Elites grid.

573 C.3 Diversity reward computation

574 QD-PG optimizes solutions for quality but also for diversity at the state level. The diversity policy
 575 gradient updates the solutions so as to encourage them to visit states with novel state descriptors. The
 576 novelty of a state descriptor $\psi(s_t)$ is expressed through a diversity reward r_t^D . In practice, we maintain
 577 a FIFO archive \mathbb{A} of the state descriptors encountered so far. When a transition $(s_t, a_t, r_t, s_{t+1}, \psi(s_t))$
 578 is stored in the replay buffer, we also add $\psi(s_t)$ to \mathbb{A} . We only add a state descriptor in \mathbb{A} if its mean
 579 Euclidean distance to its K nearest neighbors is greater than an acceptance threshold. This filtering
 580 step enables to keep the archive size reasonable and to facilitate the computation of the K nearest
 581 neighbors. The values of K and of the threshold are given in Table 2. When a batch of transitions is
 582 collected during the update phase, we recompute fresh diversity rewards r_t^D as the mean Euclidean
 583 distance between the sampled state descriptors $\psi(s_t)$ and their K nearest neighbors in \mathbb{A} . These
 584 diversity rewards are used instead of standard rewards in sampled transitions $(s_t, a_t, r_t^D, s_{t+1}, \psi(s_t))$
 585 to compute the diversity policy gradient.

586 C.4 QD-PG Hyper-parameters

587 Table 2 summarizes hyper-parameters used in experiments. Most of these hyper-parameter values are
 588 taken from TD3.

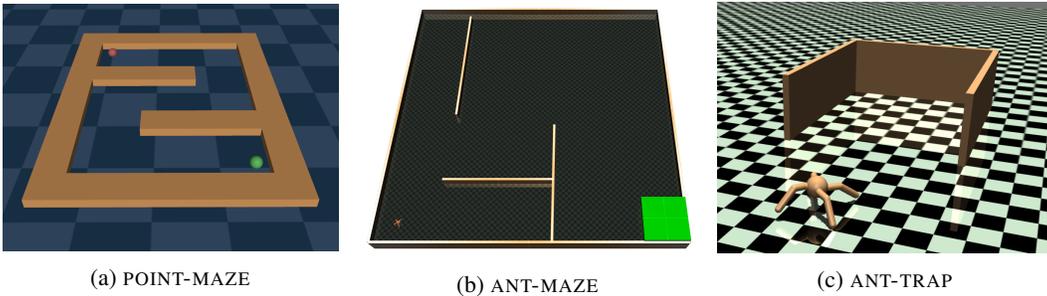


Figure 4: Evaluation environments. Though they may look similar, the state and action spaces in POINT-MAZE are two-dimensional, whereas they are 29×8 in ANT-MAZE and 113×8 in ANT-TRAP.

589 D Environments analysis

590 In POINT-MAZE, the state and action spaces are two-dimensional. By contrast, in ANT-MAZE and
 591 ANT-TRAP, the dimensions of their observation spaces are respectively equal to 29 and 113 while the
 592 dimensions of their action spaces are both equal to 8, making these two environments much more
 593 challenging as they require larger controllers. The ANT-TRAP environment also differs from mazes
 594 as it is open-ended, i.e., the space to be explored by the agent is unlimited, unlike mazes where this

Table 2: QD-PG Hyper-parameters: ANT-MAZE and ANT-TRAP hyper-parameters are identical and grouped under the Ant column

Parameter	PointMaze	Ant
TD3		
Optimizer	SGD	SGD
Learning rate	6.10^{-3}	3.10^{-4}
Discount factor γ	0.99	0.99
Replay buffer size	10^6	5.10^5
Hidden layers size	64/32	256/256
Activations	ReLU	ReLU
Minibatch size	256	256
Target smoothing coeff.	0.005	0.005
Delay policy update	2	2
Target update interval	1	1
Gradient steps ratio	4	0.1
State Descriptors Archive		
Archive size	10000	10000
Threshold of acceptance	0.0001	0.1
K-nearest neighbors	10	10
MAP-Elites		
Nb. of bins per dimension	5	7

595 space is restricted by the walls. In this case, a state descriptor corresponds to the ant position that is
 596 clipped to remain in a given range. On the y -axis, this range is defined as three times the width of
 597 the trap. On the x -axis, this range begins slightly behind the starting position of the ant and is large
 598 enough to let it accelerate along this axis. Figure 7b depicts the BD space in ANT-TRAP.

599 In all environments, state descriptors $\psi(s_t)$ are defined as the agent’s position at time step t and
 600 behavior descriptors $\xi(\theta)$ are defined as the agent’s position at the end of a trajectory. Therefore, we
 601 have $\mathcal{B} = \mathcal{D} = \mathbb{R}^2$, $\psi(s_t) = (x_t, y_t)$ and $\xi(\theta) = (x_T, y_T)$ where T is the trajectory length. We also
 602 take $\|\cdot\|_{\mathcal{B}}$ and $\|\cdot\|_{\mathcal{D}}$ as Euclidean distances. This choice does not always satisfy Equation (3) but is
 603 convenient in practice and led to satisfactory results. The peculiarity of ANT-TRAP lies in the fact
 604 that the reward is expressed as the forward velocity of the ant, thus making the descriptors not totally
 605 aligned with the task.

606 Figure 5 highlights the deceptive nature of the POINT-MAZE and the ANT-MAZE objective functions by
 607 depicting gradient fields in both environments. Similarly, the reward is also deceptive in ANT-TRAP.

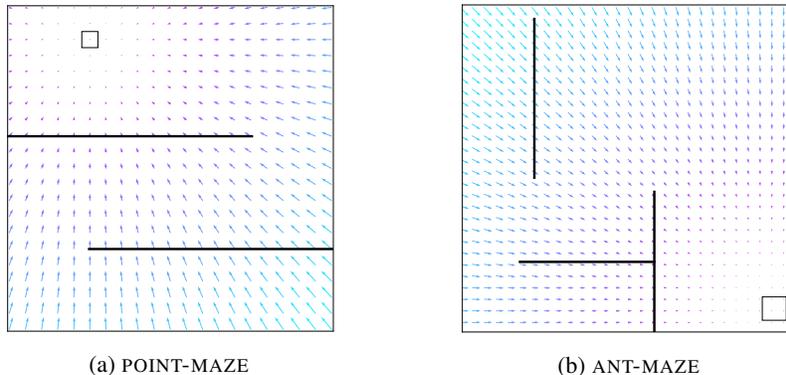


Figure 5: Gradients maps on POINT-MAZE and ANT-MAZE. Black lines are maze walls, arrows depict gradient fields and the square indicates the maze exit. Both settings present deceptive gradients as naively following them leads into a wall.

608 **E Detailed results**

609 In this section, we provide performance charts corresponding to Table 1b and Table 1a, coverage
 610 maps highlighting the exploration capabilities of QD-PG, and detailed results of the fast adaptation
 611 experiment. Table 3 summarizes the different components present in QD-PG, its ablations and all
 612 baselines.

613 **E.1 Performance charts**

614 Figure 6 compares QD-PG to competitors on all environments. In QD-PG, the current population
 615 of solutions is evaluated every 150.000 time steps in ANT-MAZE and ANT-TRAP, and every 5000
 616 time steps in POINT-MAZE. At evaluation time, agents are set to be deterministic and stop exploring.
 617 Figure 6 reports the performance obtained by the best agent in the population at a given time step.

Table 3: Ablations and baselines summary. Selec. stands for selection. The last column assesses whether the method optimizes for a collection instead of a single solution.

	Algorithm	QPG	DPG	Q Selec.	D Selec.	Collection
Ablations	QD-PG	✓	✓	✓	✓	✓
	QD-PG SUM	✓	✓	✓	✓	✓
	D-PG	X	✓	✓	✓	✓
	Q-PG	✓	X	✓	✓	✓
PG	SAC	✓	X	X	X	X
	TD3	✓	X	X	X	X
	RND	✓	✓	X	X	X
	CEM-RL	✓	X	✓	X	✓
QD	ME-ES	X	X	✓	✓	✓
	NSR-ES	X	X	✓	✓	✓
	NSRA-ES	X	X	✓	✓	✓

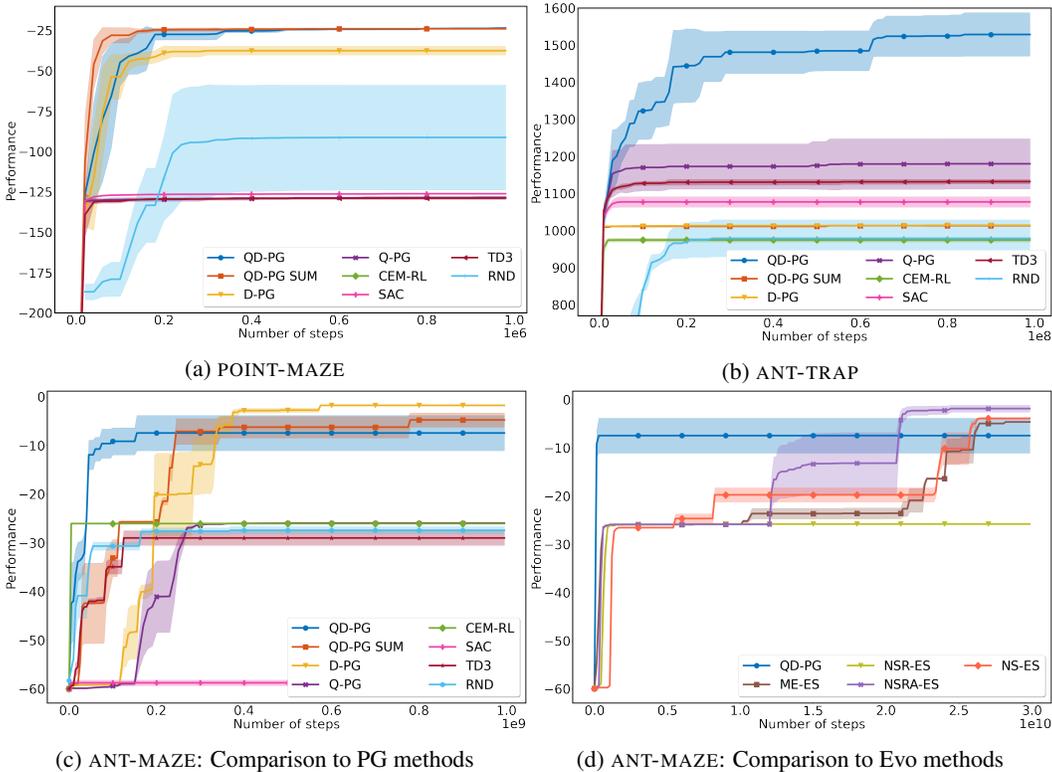


Figure 6: Learning curves of QD-PG versus ablations and baselines. In POINT-MAZE and ANT-TRAP, the performance is the highest return. In ANT-MAZE, it is the negative lowest distance to the goal. We separate the comparison on ANT-MAZE into two graphs for better readability.

618 **E.2 Coverage Maps**

619 Figure 7a shows coverage maps of the POINT-MAZE environment obtained with one representative
 620 seed by the different algorithms presented in the ablation study (see Table 1a). A dot in the figure
 621 corresponds to the final position of an agent after an episode. The color spectrum highlights the
 622 course of training: agents evaluated early in training are in blue while newer ones are represented in
 623 purple.

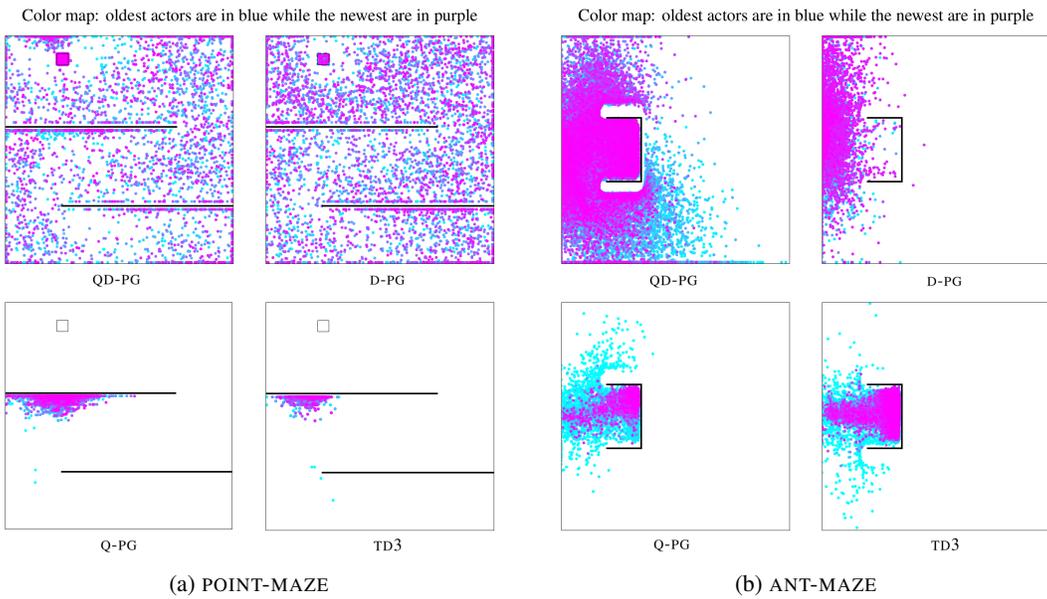


Figure 7: Coverage map of the POINT-MAZE and ANT-TRAP environments for all ablations. Each dot corresponds to the final position of an agent.

624 QD-PG and D-PG almost cover the whole BD space including the objective. Unsurprisingly, Q-PG and
 625 TD3 present very poor coverage maps, both algorithms optimize only for quality and the MAP-Elites
 626 selection mechanism in Q-PG contributes nothing in this setting. By contrast, algorithms optimizing
 627 for diversity (QD-PG and D-PG) find the maze exit. However, as shown in Table 1a, QD-PG which
 628 also optimizes for quality, is able to refine trajectories through the maze and obtains significantly
 629 better performance.

630 Figure 7b depicts the coverage maps of the ANT-TRAP environment by QD-PG and TD3. Only QD-PG
 631 is able to bypass the trap and to cover a large part of the BD space.

632 **E.3 Fast adaptation**

633 The fast adaptation experiment described in Section 7 uses a Bayesian optimization process to quickly
 634 find a high-performing solution for a new randomly sampled goal. Browsing the MAP-Elites grid in
 635 an exhaustive way is another option to find a good solution for a new objective. However, the number
 636 of solutions to be tested with this option increases quadratically w.r.t. the number of meshes used to
 637 discretize the dimensions of the BD space. As shown in Table 2, we use a 7×7 grid to train QD-PG
 638 in the ANT-MAZE environment, containing a maximum of 49 solutions. In this setting, the difference
 639 in computation cost between exhaustive search and Bayesian optimization is negligible.

640 To ensure that fast adaptation scales to finely discretized MAP-Elites grids, we reproduce this experi-
 641 ment with a 100×100 grid, thus containing thousands of solutions. We first train QD-PG again on the
 642 standard objective of ANT-MAZE and obtain a 100×100 grid of solutions. Then, we repeat the fast
 643 adaptation experiment described in Section 7 using this large grid. With a budget of only 50 solutions
 644 to be tested during the Bayesian optimization process among the thousands of solutions contained in
 645 the grid, we are able to recover a good solution for the new objective. We repeat this experiment 100
 646 times, each time with a new random goal, and obtain an average performance of -9 with a standard
 647 deviation of 7.

648 Figure 8 maps these 100 fast adaptation experiments to their respective goal location and performance.
 649 In each square, we display the score of the best experiment whose goal was sampled in this region of
 650 the maze. For instance, the square in the top left corner of the performance map corresponds to one
 651 of the 100 fast adaptation experiments that sampled its goal in this part of the maze, and obtained
 652 a performance of -12 after testing 50 solutions from the MAP-Elites grid during the Bayesian
 653 optimization process. Some squares do not have a score when no experiment sampled its goal in this
 654 region of the maze.

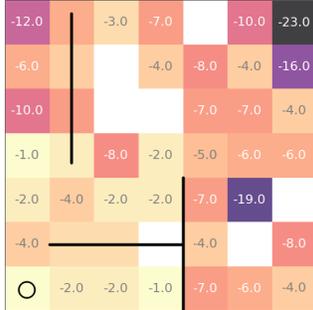


Figure 8: Performance map of 100 fast adaptation experiments in ANT-MAZE. In each square, we display the score of the best experiment whose goal was sampled in this region of the maze, as several experiments may have goals in the same square. White squares correspond to regions where no goal was sampled during the experiments. The black circle shows the agent’s starting position.

655 F An alternative Pareto Front approach

656 QD-PG uses a MAP-Elites grid as a convenient and powerful way to store and select solutions that
 657 are both novel and high-performing. Inspired from Cully & Demiris (2017), we also considered the
 658 Pareto front as an alternative structure to store and select solutions according to quality and diversity
 659 criteria. In this setting, the MAP-Elites grid (see Figure 2b) is replaced by an archive storing all
 660 solutions encountered so far with their respective performance and behavior descriptor. At each
 661 iteration of the QD-PG algorithm, a quality-diversity Pareto front is computed on the solutions present
 662 in the archive, and the N most novel and high-performing solutions are selected to be updated. Since
 663 the archive is evolving at each iteration, the novelty of a solution is recomputed from its behavior
 664 descriptor before the computation of the Pareto front.

665 Results obtained via this alternative method are presented in Table 4. QD-PG PARETO performs
 666 similarly to QD-PG MAP-Elites on all environments, but the additional computational cost induced by
 667 the Pareto front calculation makes it a slower method than QD-PG MAP-Elites.

Table 4: Comparison of QD-PG MAP-Elites and QD-PG PARETO on all environments.

Algorithm	Final Perf. (\pm std)		
	POINT-MAZE	ANT-MAZE	ANT-TRAP
QD-PG MAP-Elites	$-24(\pm 0)$	$-7(\pm 7)$	$1541(\pm 86)$
QD-PG PARETO	$-27(\pm 1)$	$-4(\pm 3)$	$1416(\pm 47)$

668 G Random Network Distillation Details

669 This section provides additional details on the RND baseline.

670 G.1 Details about the agent

671 The Random Network Distillation (RND) agent extends the Proximal Policy Optimization (PPO) agent
 672 to improve its exploration capabilities. RND computes an auxiliary reward r_t^i called intrinsic motiva-
 673 tion reward or, in short, intrinsic reward. In opposition to the reward provided by the environment r_t^e ,
 674 called in this case extrinsic reward, the intrinsic reward is used to encourage the agent to visit new
 675 states.

676 **Proximal Policy Optimization**

677 PPO is an on-policy actor-critic algorithm using a stochastic policy π_θ and a value function V_θ both
 678 parameterized by a neural network. It replaces the policy gradient by a surrogate loss to constrain the
 679 size of policy updates. More formally, the policy parameters are updated to maximise

$$L_\pi(\theta) = \sum_{batch} \min \left(r(\theta) \hat{A}, \text{clip}(r(\theta), 1 - \epsilon, 1 + \epsilon) \hat{A} \right), \quad (9)$$

680 where \hat{A} is an estimate of the advantage and is computed from the value network V_θ using the
 681 Generalized Advantage Estimation method (GAE) (Schulman et al., 2015). The $r(\theta)$ term denotes
 682 the policy ratio $r(\theta) = \frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)}$ where θ_{old} are the parameters before the update.

683 The value network is trained to minimize the mean squared error between its prediction and an
 684 estimation of the return. This estimation can be computed from the advantage estimate. More
 685 formally, the value function parameters are updated to minimize

$$L_V(\theta) = \sum_{batch} \left(V_\theta(s) - (\hat{A} + V_{\theta_{old}}(s)) \right)^2. \quad (10)$$

686 **Intrinsic reward computation**

687 To compute intrinsic rewards, RND introduces two additional functions $f_\theta : \mathcal{S} \rightarrow \mathbb{R}^K$ and $f_{\theta^*} : \mathcal{S} \rightarrow \mathbb{R}^K$
 688 where K is a hyper-parameter. Both functions are parameterized by a neural network.
 689 Parameters θ^* are sampled randomly at the beginning of training and are never updated. During
 690 training, RND updates the parameters θ so that f_θ imitates f_{θ^*} behavior for the states that has been
 691 visited by the agent. Every time RND updates the PPO policy and value networks, it also updates f_θ
 692 so as to minimize

$$L_f(\theta) = \sum_{batch} (f_\theta(s) - f_{\theta^*}(s))^2. \quad (11)$$

693 This way, if for a state s both functions f_θ and f_{θ^*} provide close predictions, it means that the agent
 694 has already visited this state. Therefore, the intrinsic reward is computed as

$$r_t^i = (f_\theta(s_t) - f_{\theta^*}(s_t))^2. \quad (12)$$

695 RND is trained to maximise both the extrinsic rewards obtained from the environment and the intrinsic
 696 rewards computed by the agent to enhance exploration. RND maintains two value functions $V^i(s)$
 697 and $V^e(s)$ and uses them to compute separately two advantages estimations \hat{A}^i and \hat{A}^e . Both value
 698 functions $V^i(s)$ and $V^e(s)$ are updated with Equation (10) where the rewards are respectively taken
 699 as the intrinsic and extrinsic rewards. The final advantage estimation used to update the policy, see
 700 Equation (9), is computed as the sum $\hat{A} = \hat{A}^e + \hat{A}^i$.

701 **Implementation tricks**

702 In this work, in order to facilitate the comparison between RND and QD-PG, we conditioned functions
 703 f_θ and f_{θ^*} directly on the states descriptors rather than on the states. Formally, these functions are
 704 defined as $f_\theta : \mathcal{D} \rightarrow \mathbb{R}^K$ and $f_{\theta^*} : \mathcal{D} \rightarrow \mathbb{R}^K$ and the intrinsic reward is computed as

$$r_t^i = (f_\theta(\psi(s_t)) - f_{\theta^*}(\psi(s_t)))^2. \quad (13)$$

705 **G.2 Additional Results**

706 As shown in Table 1a, the RND baseline achieves performance comparable to TD3 on both ANT-MAZE
 707 and ANT-TRAP environments. However, these results do not reveal that RND is able to extensively
 708 explore the BD space as depicted in Figure 9. As opposed to conventional RL agents (TD3 and

709 SAC), RND is able to bypass the trap in ANT-TRAP and to cover a large part of the BD space. We
 710 hypothesize that its poor performance comes from the fact that the agent, although able to bypass the
 711 trap, does not acquire a sufficient speed along the x -axis, which is the goal in ANT-TRAP.

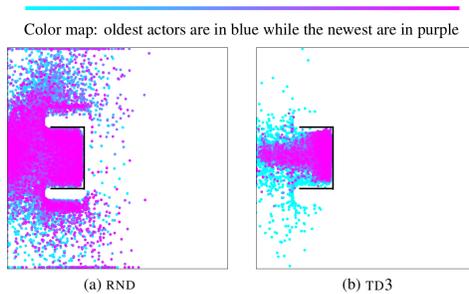


Figure 9: Coverage maps of ANT-TRAP.

712 G.3 RND Hyper-parameters

Table 5: RND Hyper-parameters: ANT-MAZE and ANT-TRAP hyper-parameters are identical and grouped under the Ant column

Parameter	PointMaze	Ant
PPO		
Optimizer	Adam	Adam
Learning rate	5.10^{-4}	5.10^{-5}
Discount factor	0.99	0.99
Clipping epsilon	0.2	0.2
Lambda GAE	0.95	0.95
Hidden layers size	64/64	256/256
Activations	ReLU	ReLU
Minibatch size	512	512
Number of epochs	30	30
RND Networks		
Output dimension K	16	16
Hidden layers size	64/64	256/256
Learning rate	5.10^{-6}	5.10^{-5}