ETH zürich

Curriculum Learning

Nicolas Küchler

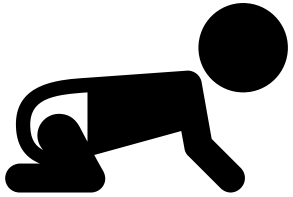
Intrinsic Motivation and Automatic Curricula via Asymmetric Self-Play

Sainbayar Sukhbaatar, Ilya Kostrikov, Arthur Szlam and Rob Fergus (2017)

Reverse Curriculum Generation for Reinforcement Learning

Carlos Florensa, David Held, Markus Wulfmeier and Pieter Abbeel (2017)

Curriculum Learning



Crawling

Walking

Skating

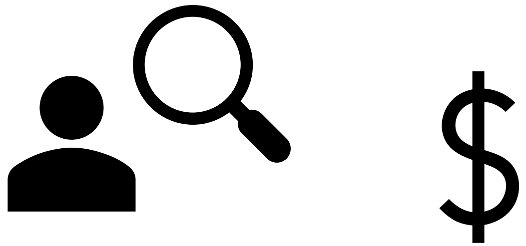
“ *The basic idea is to start small, learn easier aspects of the task or easier sub-tasks, and then gradually increase the difficulty level.*

Bengio et al., 2009

ETH zürich

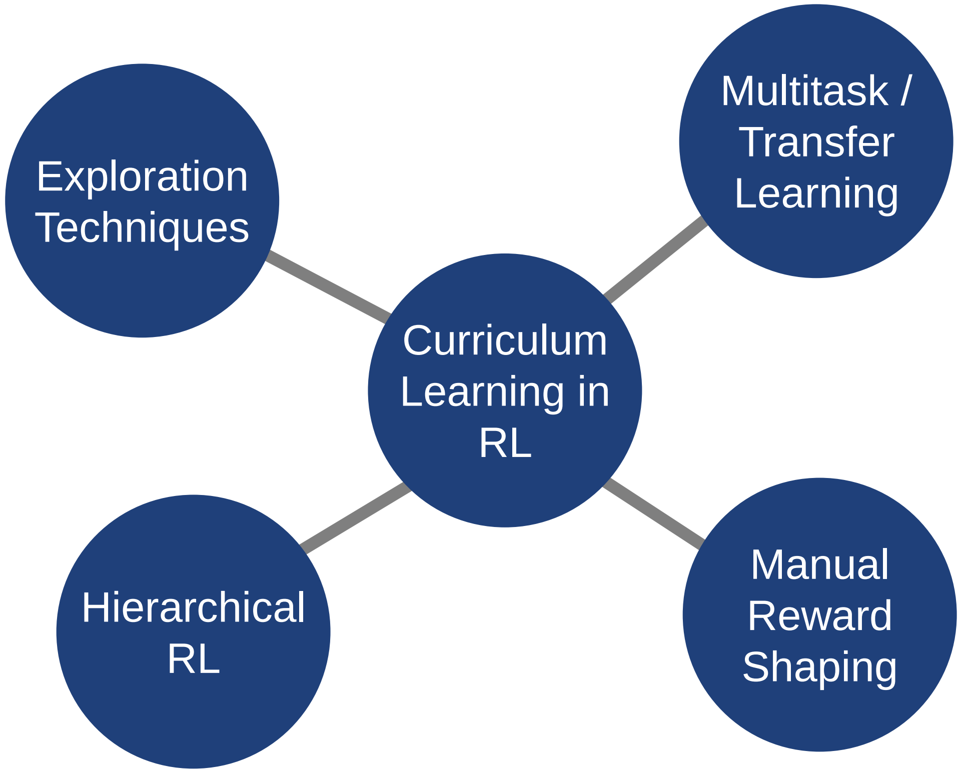
Curriculum Learning in Reinforcement Learning

When

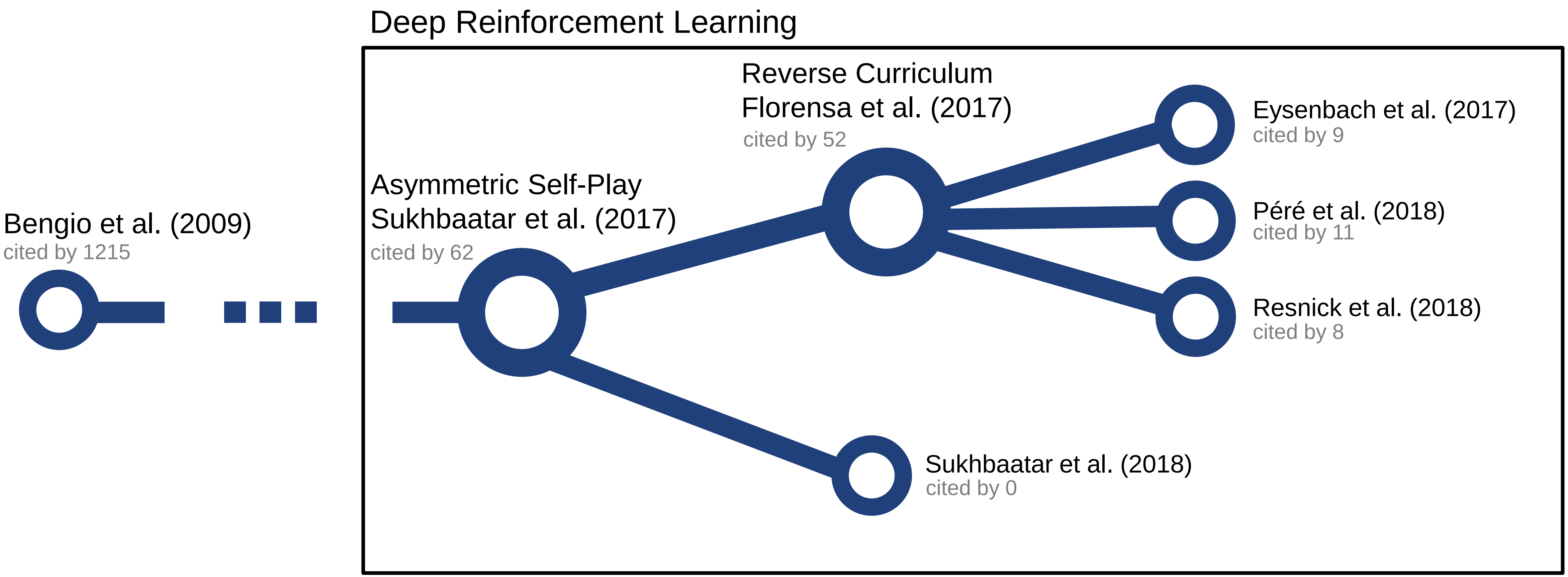


Hard Tasks with Sparse Rewards

Alternatives and Related Concepts



Research – Curriculum Learning

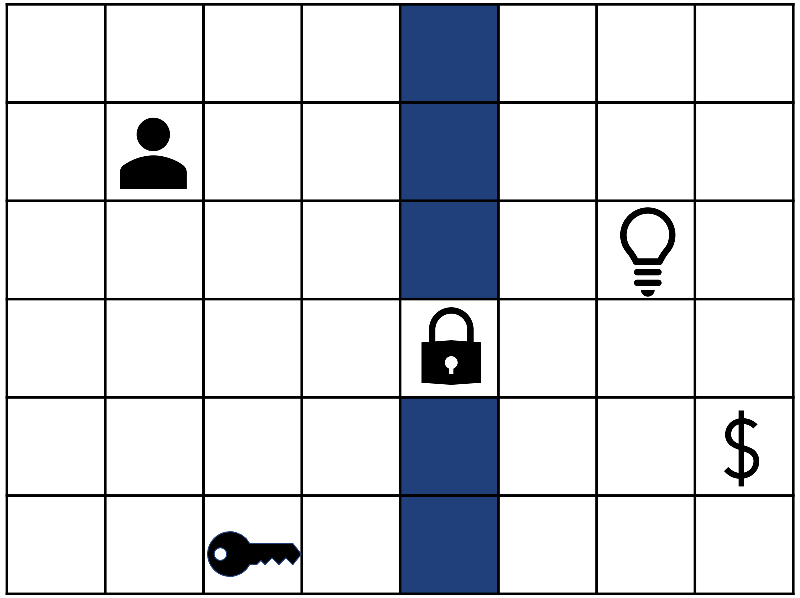


Source Citations: Google Scholar 24.04.2019

Intrinsic Motivation and Automatic Curricula via Asymmetric Self-Play

Sainbayar Sukhbaatar, Ilya Kostrikov, Arthur Szlam and Rob Fergus

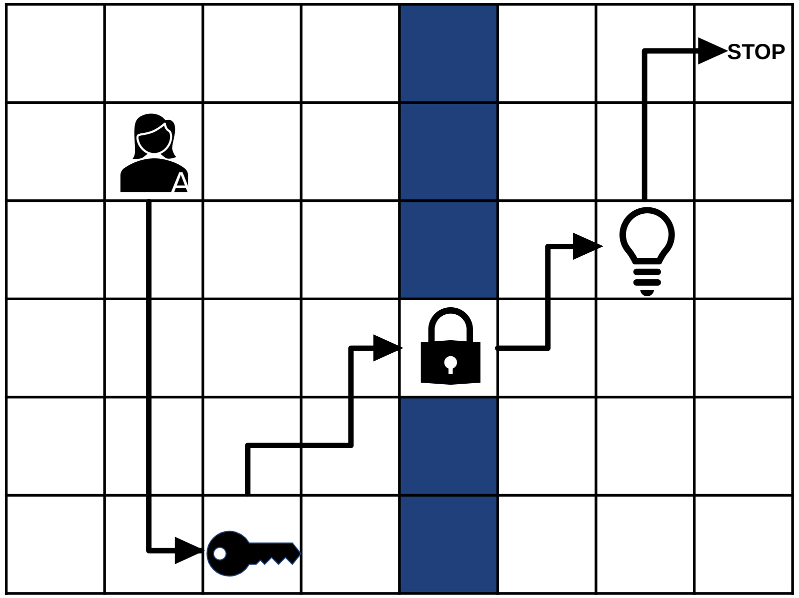
MazeBase Environment



Idea: Asymmetric Self-Play

1. Propose Task

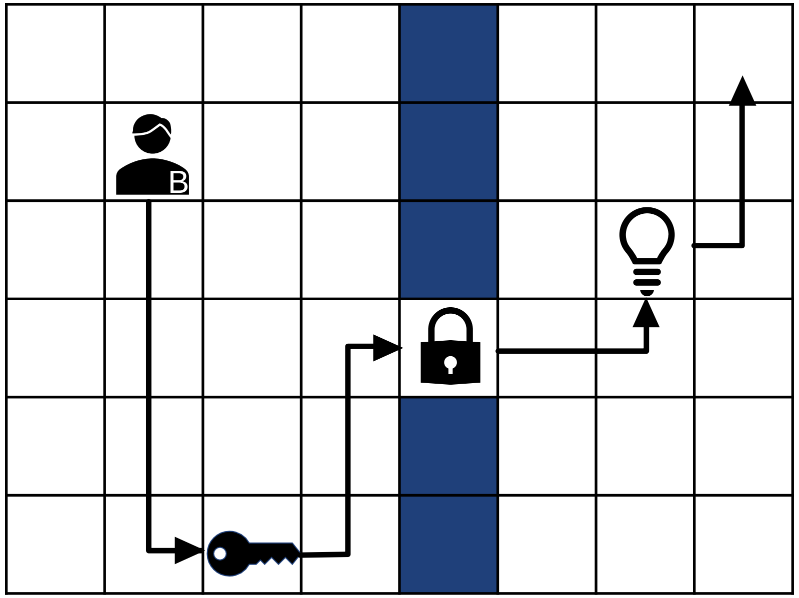
Alice $a_A = \pi_A(s_t, s_0)$



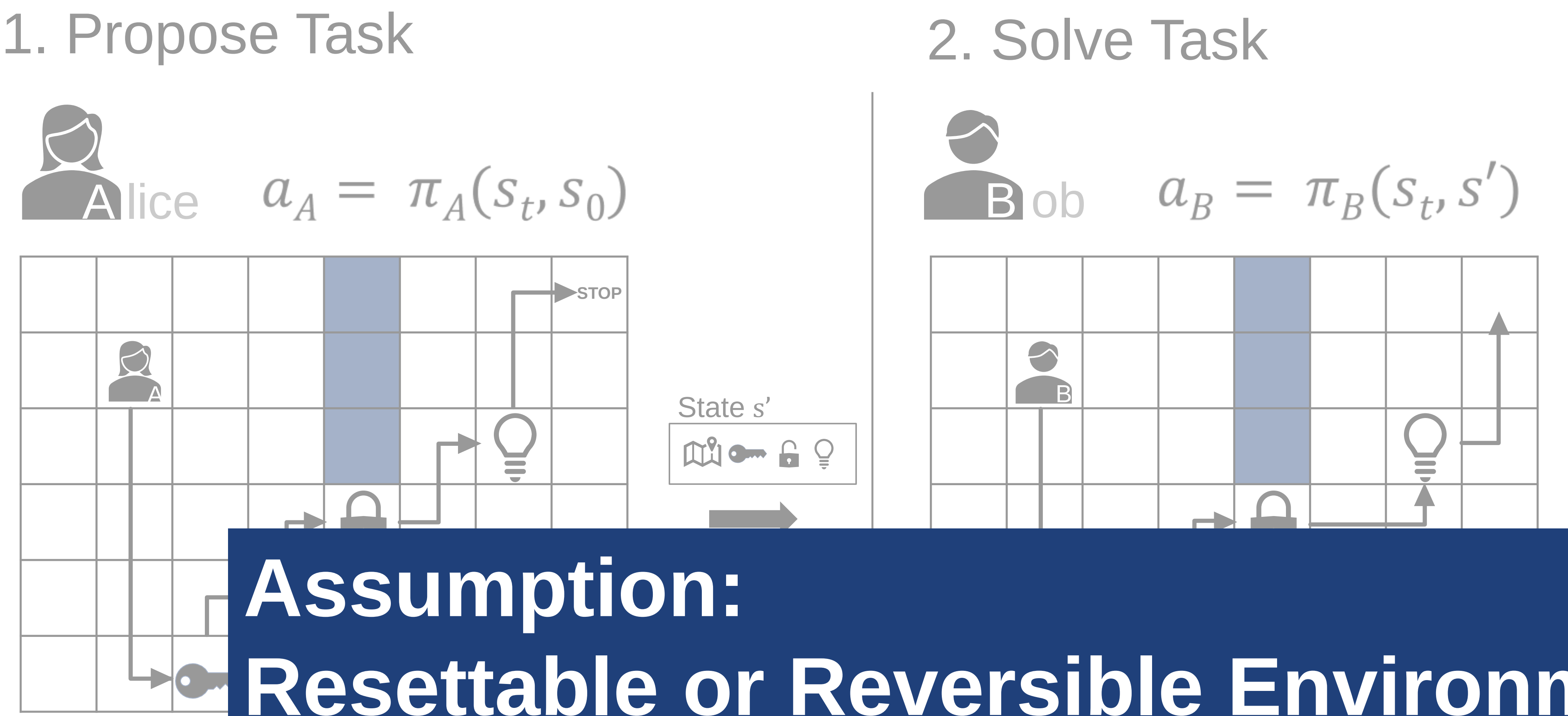
State s'

2. Solve Task

Bob $a_B = \pi_B(s_t, s')$

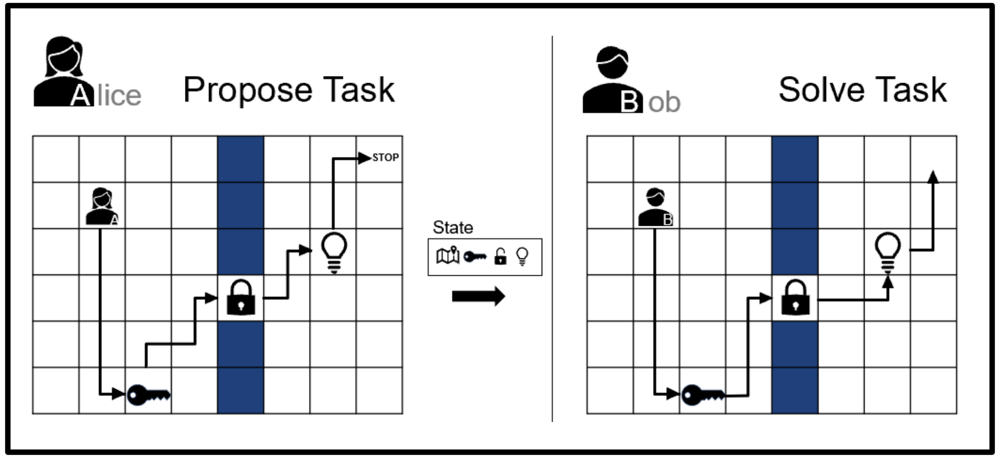


Idea: Asymmetric Self-Play



Assumption:
Resettable or Reversible Environment

Key Idea Curricula: Internal Reward



❝ Encourage Alice to push Bob past his comfort zone but not give him impossible tasks

➡ Automatic Curriculum

$R_{Bob} = -\gamma \, t_{Bob}$ (if Bob fails: $t_{Bob} = t_{max} - t_{Alice}$)

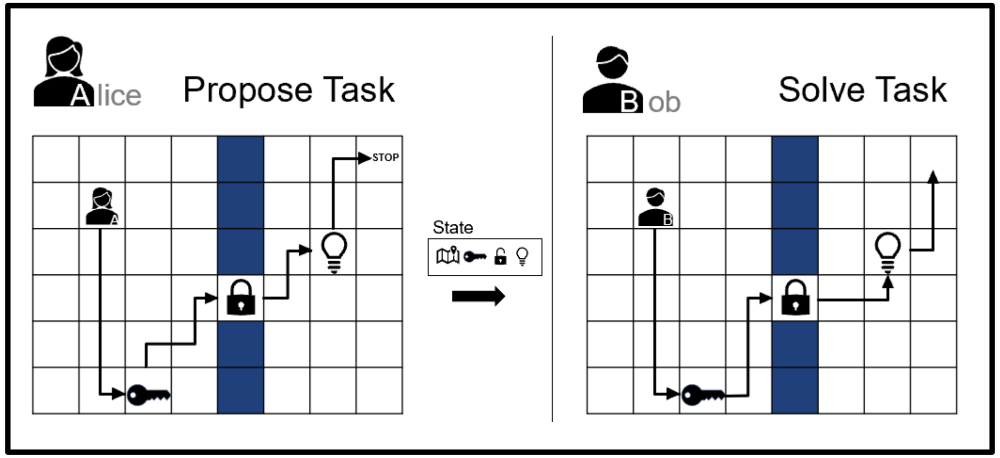
scaling constant: hyperparameter

steps Bob

limit: hyperparameter

steps Alice

Key Idea Curricula: Internal Reward



Encourage Alice to push Bob past his comfort zone but not give him impossible tasks
➡ Automatic Curricula

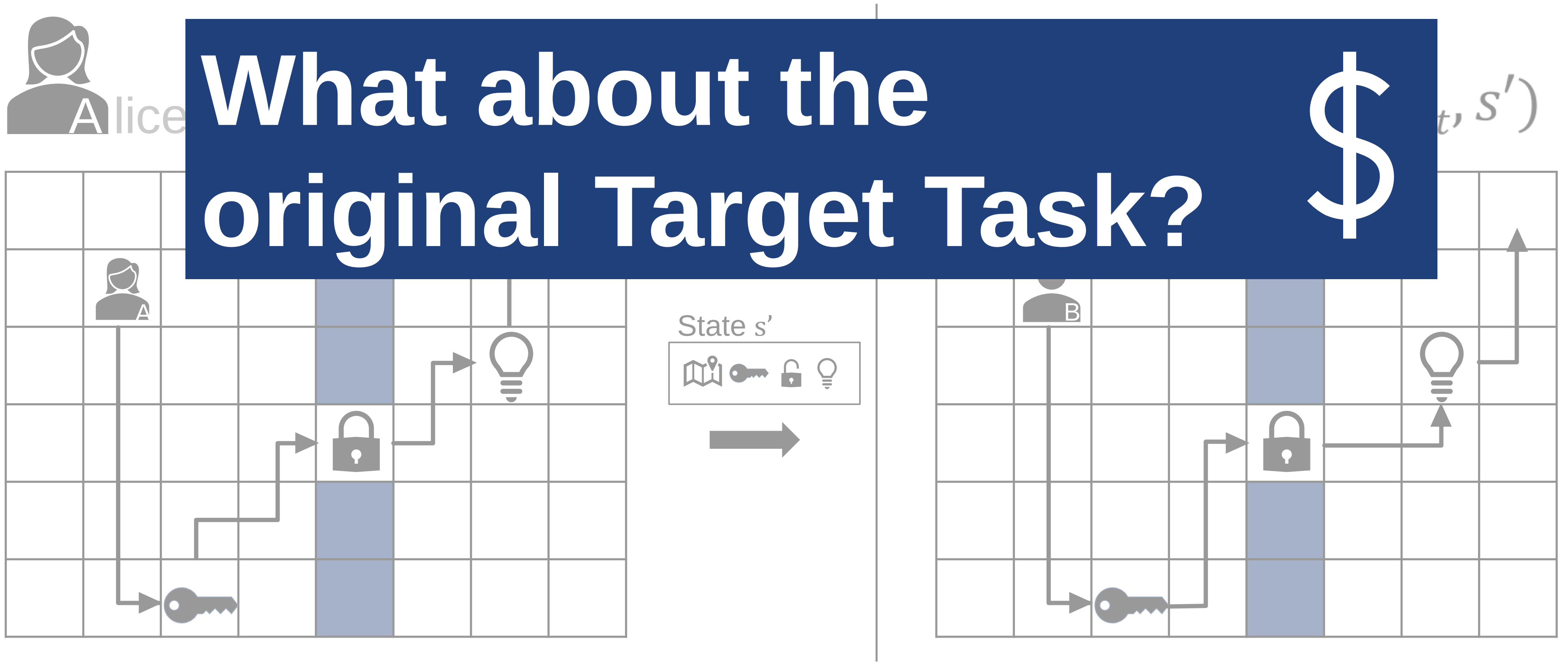
$R_{Bob} = -\gamma\, t_{Bob}$ (if Bob fails: $t_{Bob} = t_{max} - t_{Alice}$)
→ optimal behaviour: solve task as fast as possible

$R_{Alice} = \gamma \max(0, t_{Bob} - t_{Alice})$
→ optimal behaviour: find simplest task Bob cannot complete

Idea: Asymmetric Self-Play

1. Propose Task

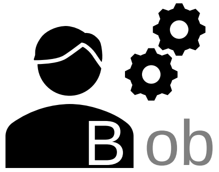
2. Solve Task



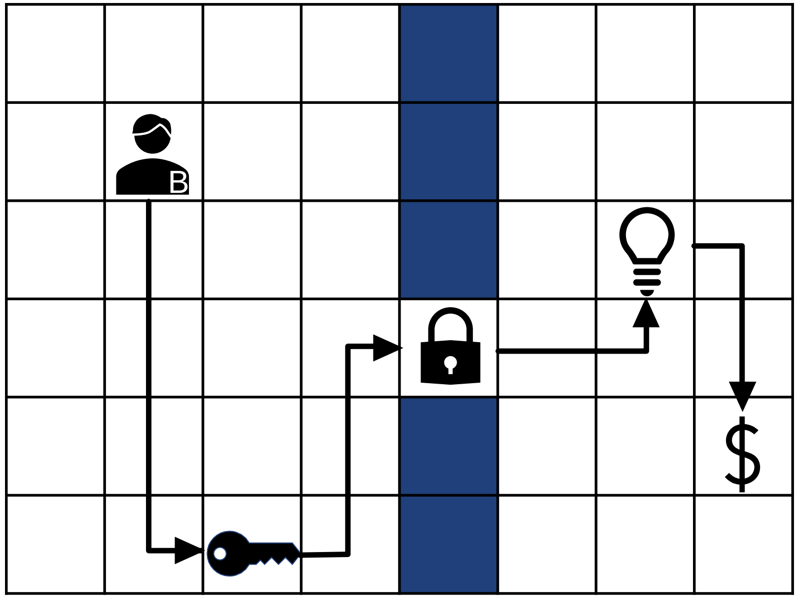
What about the original Target Task?

$

Idea: Application to Target Task



$$a_B = \pi_B(s_t, \phi)$$



Framework: Putting Everything Together

Function `SelfPlayEpisode`($\pi_{Alice}, \pi_{Bob}, s_0$):

$s', t_{Alice} \leftarrow proposeTask(\pi_{Alice}, s_0)$

$t_{Bob} \leftarrow solveTask(\pi_{Bob}, s_0, s')$

$R_{Alice} \leftarrow \gamma\, max(0, t_{Bob} - t_{Alice})$

$R_{Bob} \leftarrow -\gamma\, t_{Bob}$

$\pi_{Alice} \leftarrow updatePolicy(\pi_{Alice}, R_{Alice})$

$\pi_{Bob} \leftarrow updatePolicy(\pi_{Bob}, R_{Bob})$

End Function

Function `TargetTaskEpisode`(π_{Bob}, s_0)

Framework: Putting Everything Together

Function `SelfPlayEpisode`$(\pi_{Alice}, \pi_{Bob}, s_0)$:

$$s', t_{Alice} \leftarrow proposeTask(\pi_{Alice}, s_0)$$
$$t_{Bob} \leftarrow solveTask(\pi_{Bob}, s_0, s')$$

$$R_{Alice} \leftarrow \gamma\, max(0, t_{Bob} - t_{Alice})$$
$$R_{Bob} \leftarrow -\gamma\, t_{Bob}$$

$$\pi_{Alice} \leftarrow updatePolicy(\pi_{Alice}, R_{Alice})$$
$$\pi_{Bob} \leftarrow updatePolicy(\pi_{Bob}, R_{Bob})$$

End Function

Function `TargetTaskEpisode`(π_{Bob}, s_0)

1. Propose Task
2. Solve Task

Framework: Putting Everything Together

Function `SelfPlayEpisode`$(\pi_{Alice}, \pi_{Bob}, s_0)$:

$s', t_{Alice} \leftarrow proposeTask(\pi_{Alice}, s_0)$

$t_{Bob} \leftarrow solveTask(\pi_{Bob}, s_0, s')$

$R_{Alice} \leftarrow \gamma\, max(0, t_{Bob} - t_{Alice})$

$R_{Bob} \leftarrow -\gamma\, t_{Bob}$

$\pi_{Alice} \leftarrow updatePolicy(\pi_{Alice}, R_{Alice})$

$\pi_{Bob} \leftarrow updatePolicy(\pi_{Bob}, R_{Bob})$

End Function

Function `TargetTaskEpisode`(π_{Bob}, s_0)

1. Propose Task
2. Solve Task

Internal Reward

Framework: Putting Everything Together

Function `SelfPlayEpisode`($\pi_{Alice}, \pi_{Bob}, s_0$):

$s', t_{Alice} \leftarrow proposeTask(\pi_{Alice}, s_0)$

$t_{Bob} \leftarrow solveTask(\pi_{Bob}, s_0, s')$

$R_{Alice} \leftarrow \gamma\, max(0, t_{Bob} - t_{Alice})$

$R_{Bob} \leftarrow -\gamma\, t_{Bob}$

$\pi_{Alice} \leftarrow updatePolicy(\pi_{Alice}, R_{Alice})$

$\pi_{Bob} \leftarrow updatePolicy(\pi_{Bob}, R_{Bob})$

End Function

Function `TargetTaskEpisode`(π_{Bob}, s_0)

Alice Bob

1. Propose Task
2. Solve Task

Internal Reward

(e.g. Policy Gradient with Baseline, TRPO, …)

Framework: Putting Everything Together

Function `SelfPlayEpisode(`$\pi_{Alice}, \pi_{Bob}, s_0$`):`

$s', t_{Alice} \leftarrow proposeTask(\pi_{Alice}, s_0)$
$t_{Bob} \leftarrow solveTask(\pi_{Bob}, s_0, s')$

$R_{Alice} \leftarrow \gamma\, max(0, t_{Bob} - t_{Alice})$
$R_{Bob} \leftarrow -\gamma\, t_{Bob}$

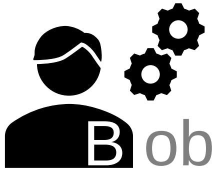
$\pi_{Alice} \leftarrow updatePolicy(\pi_{Alice}, R_{Alice})$
$\pi_{Bob} \leftarrow updatePolicy(\pi_{Bob}, R_{Bob})$

End Function

Function `TargetTaskEpisode(`π_{Bob}, s_0`)`

Internal Reward

(e.g. Policy Gradient with Baseline, TRPO, …)

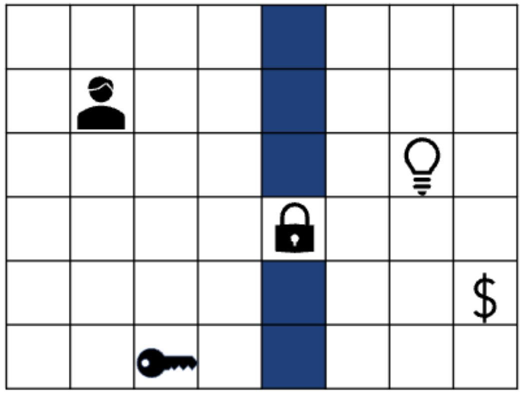


Experiments

Discrete State Space

- Long Hallway

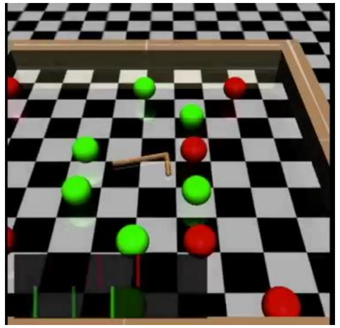
- MazeBase → Focus



Continuous State Space

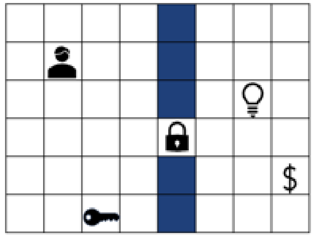
- Mountain Car

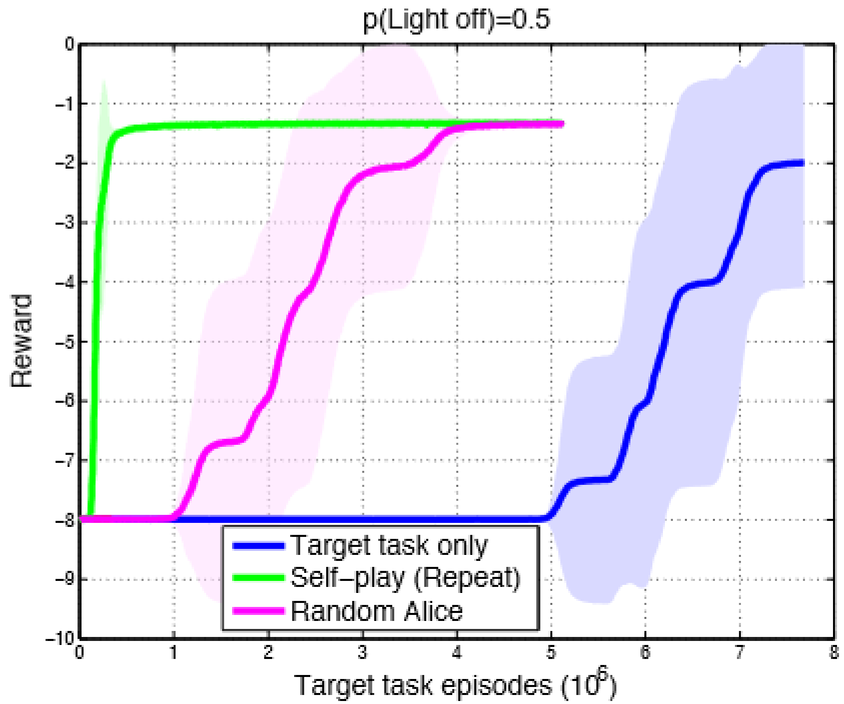
- Swimmer Gather → Focus



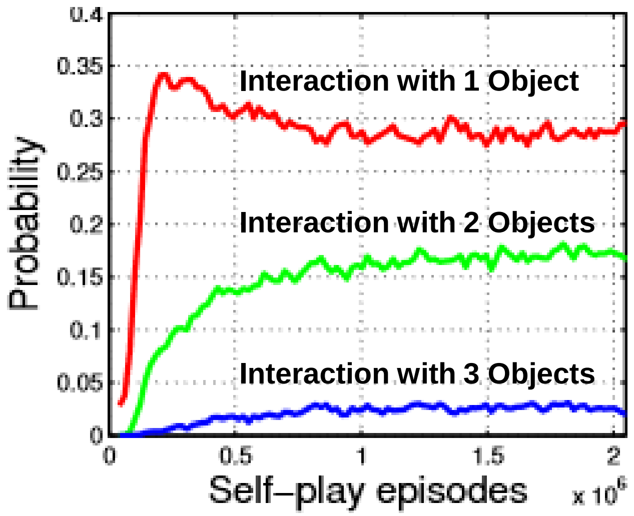
- Build Marine Units Mini Game in StarCraft

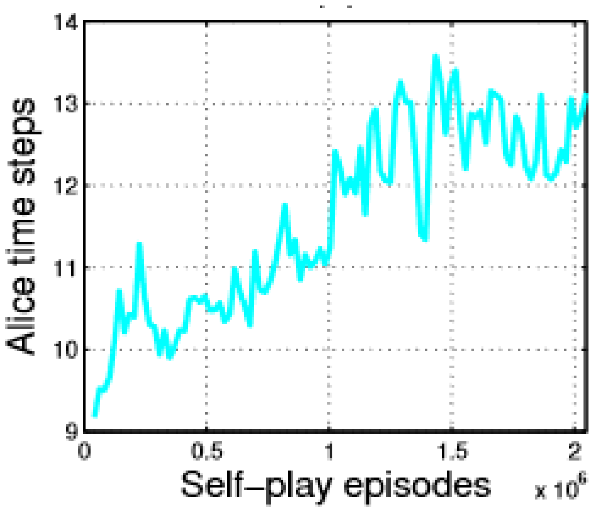
Experiments: MazeBase





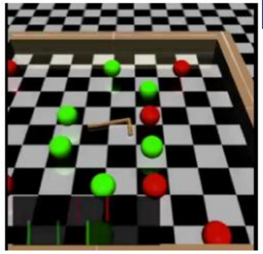
Self-Play (Repeat):

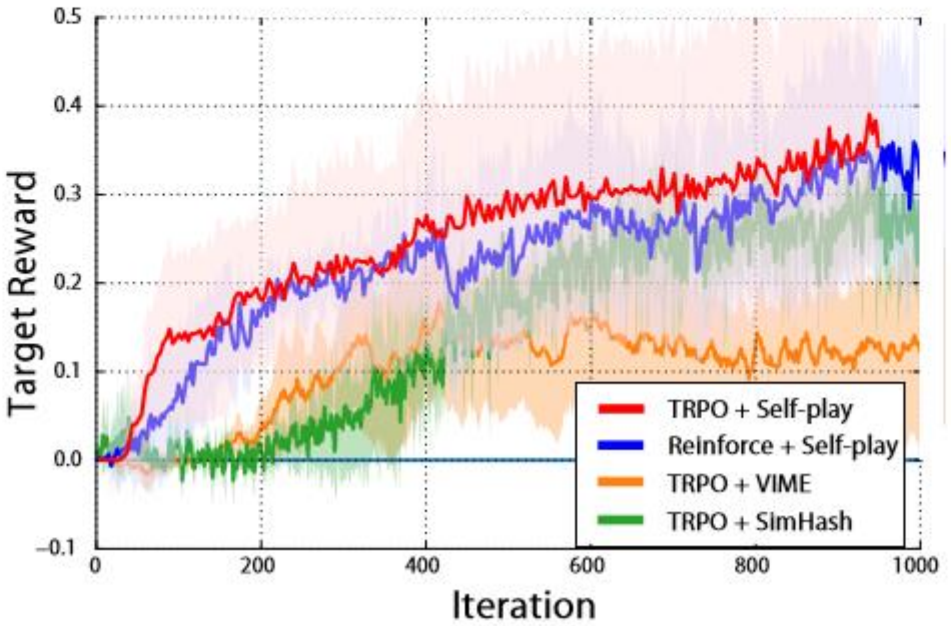




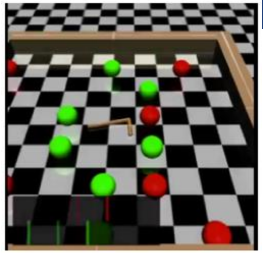
Objects:

Experiments: Swimmer Gather



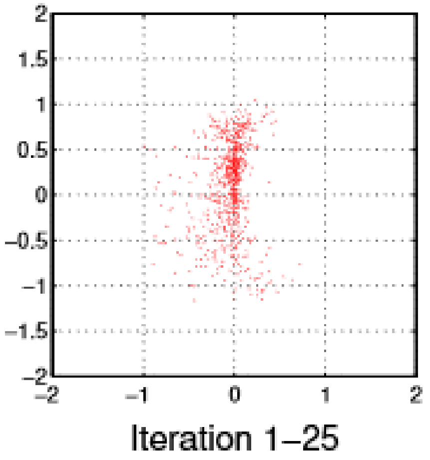


Experiments: Swimmer Gather

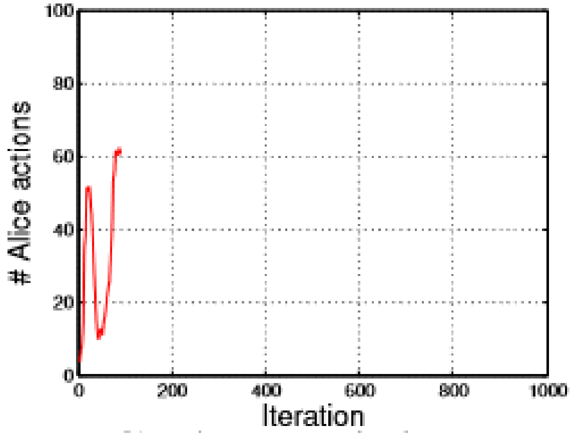


Phase: Initialization

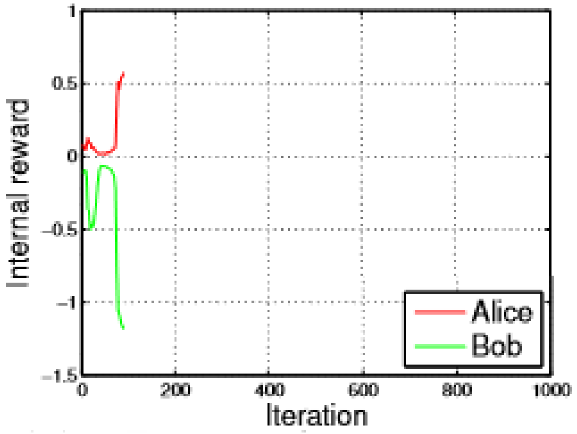
Alice: Position



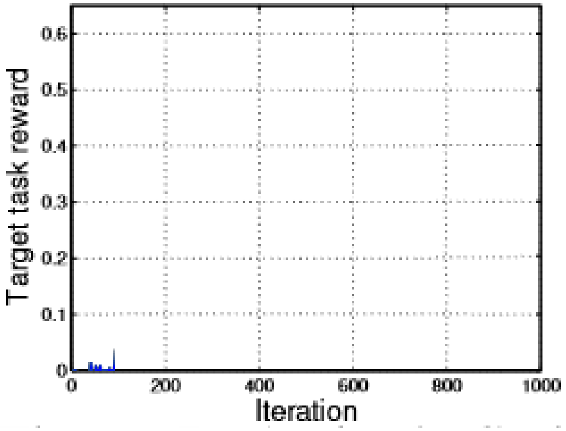
Alice: Difficulty Task



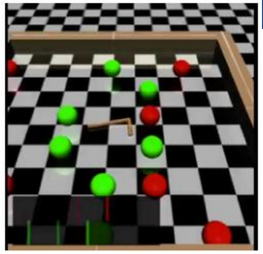
Self-Play Reward



Target Task Reward

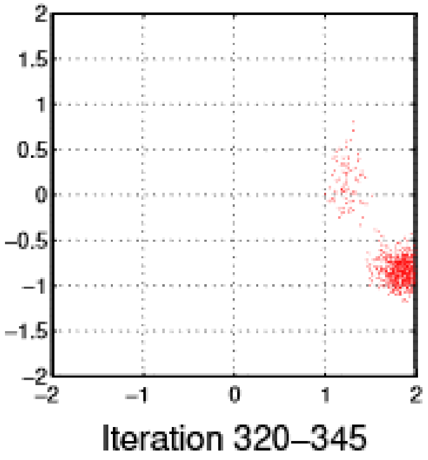


Experiments: Swimmer Gather

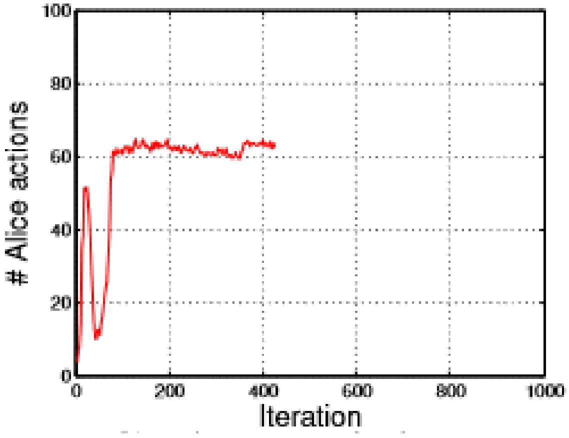


Phase: Alice better than Bob

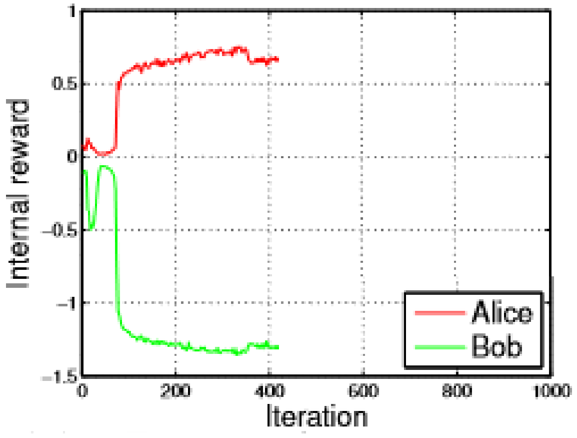
Alice: Position



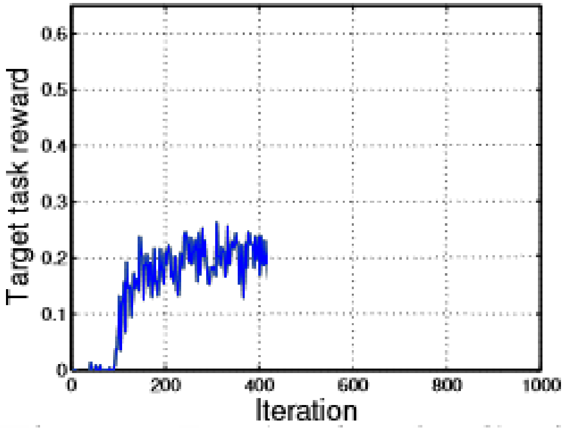
Difficulty Task Alice



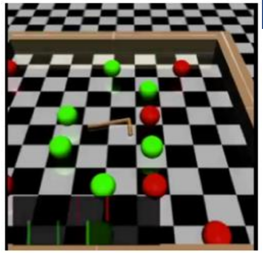
Self-Play Reward



Target Task Reward

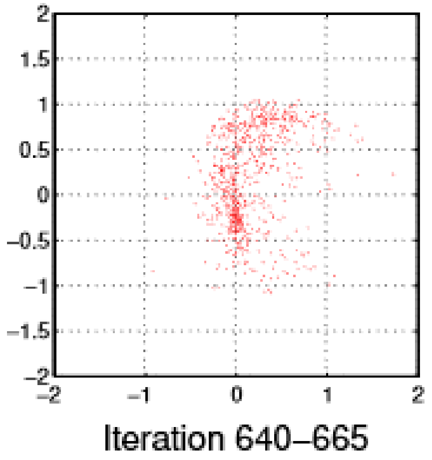


Experiments: Swimmer Gather

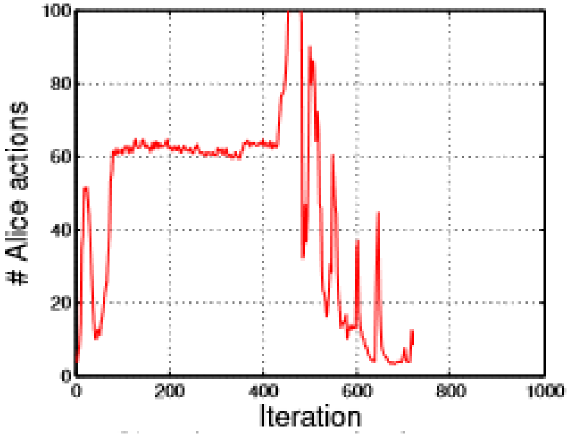


Phase: Bob overpowers Alice → Alice gives up

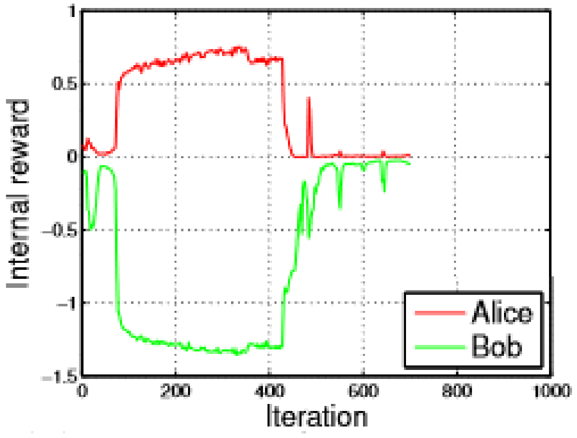
Alice: Position



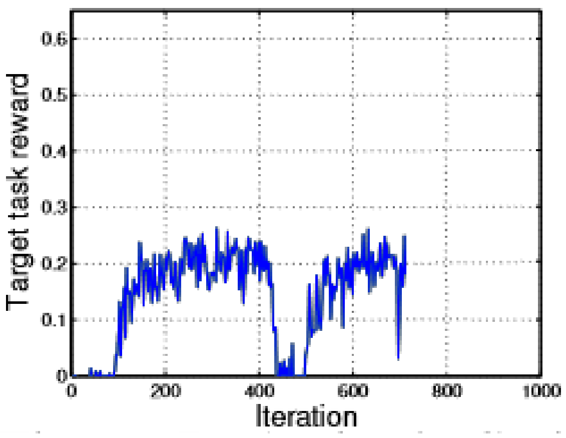
Difficulty Task Alice



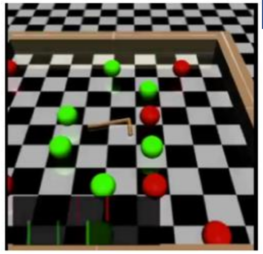
Self-Play Reward



Target Task Reward

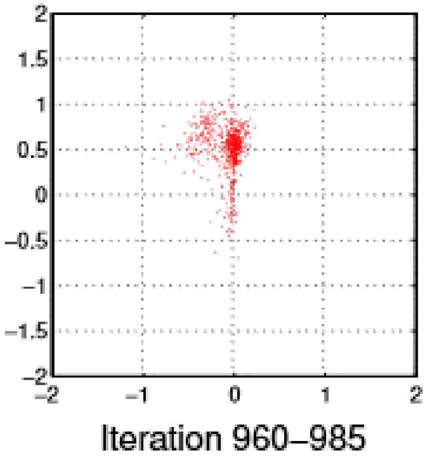


Experiments: Swimmer Gather

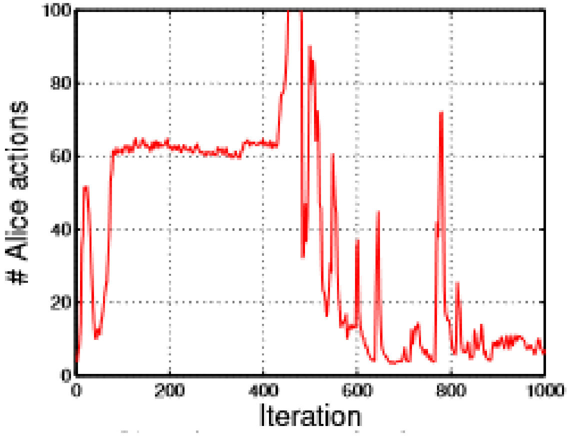


Phase: Alice can’t recover

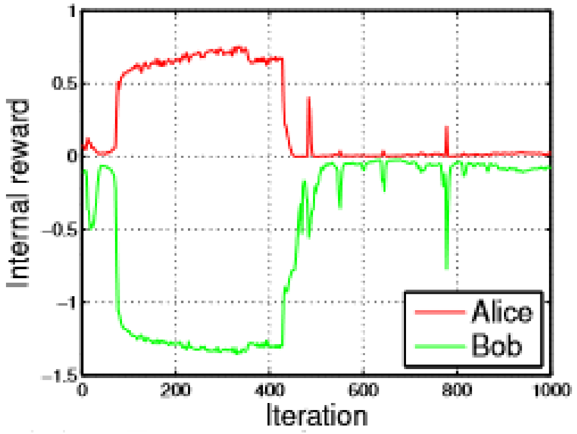
Alice: Position



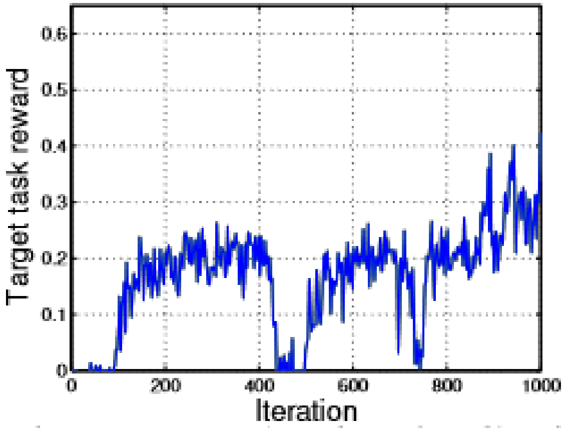
Difficulty Task Alice



Self-Play Reward

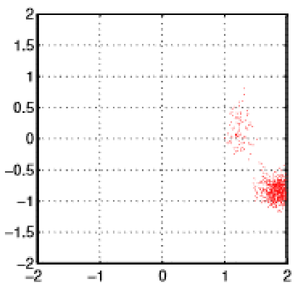


Target Task Reward



Potential Problems

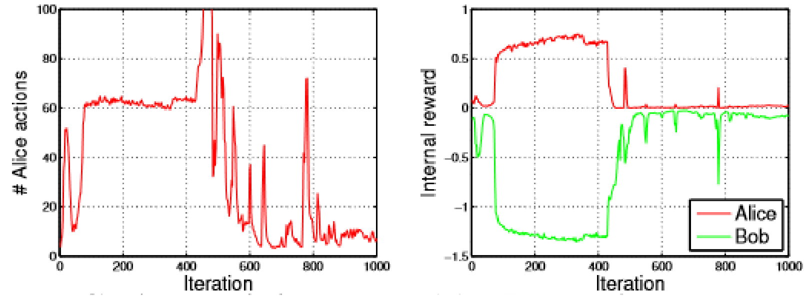
- Meta Exploration for Alice: Single Hardest Task vs All Hard Tasks



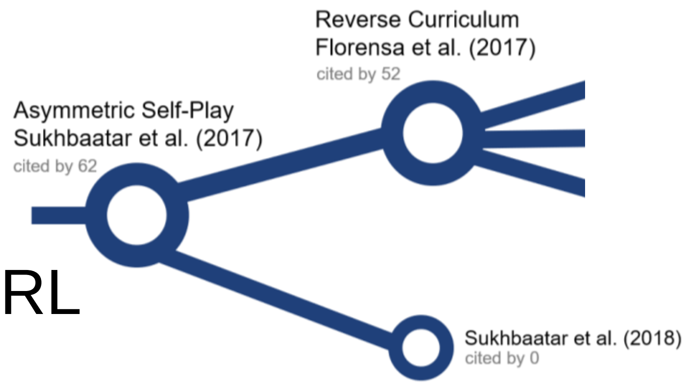
- Performance in Continuous Action Spaces (Florensa et al. 2017)
  - Effect of Meta Exploration enhanced when using a unimodal Gaussian Distribution to parametrize Action Space → Alice converges to moving into single direction
- Reward Function of Alice often Sparse (Florensa et al. 2017)

$$R_{Alice} = \gamma \max(0, t_{Bob} - t_{Alice})$$

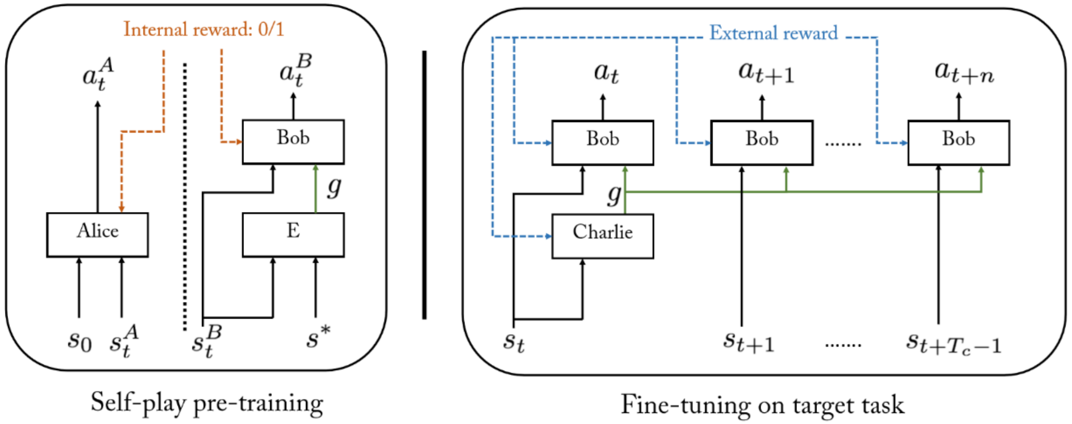
  - If Bob outperforms Alice: → Alice receives 0 reward
    → Alice gives up



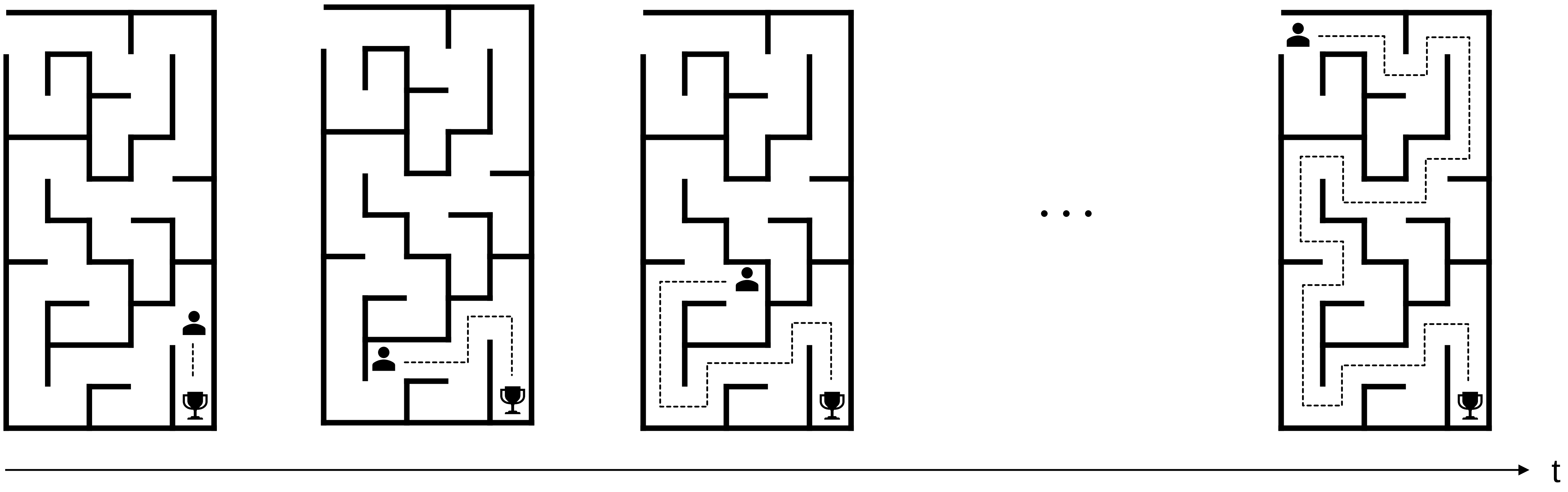
Asymmetric Self-Play: Follow Up Work



- Learning Goal Embeddings via Self-Play for Hierarchical RL (Sukhbaatar et al. 2018)



- Reverse Curriculum Generation for Reinforcement Learning (Florensa et al. 2017)



Reverse Curriculum Generation for Reinforcement Learning

Carlos Florensa, David Held, Markus Wulfmeier and Pieter Abbeel

Starting State Distribution p_i at each Iteration i

Good Starting State $s \in \mathbb{R}^d$?

“ *… in goal-oriented environments, a strong learning signal is obtained when training on start states s $\sim \rho_i$ from where the agent reaches the goal sometimes, but not always.*

Florensa et al. 2017

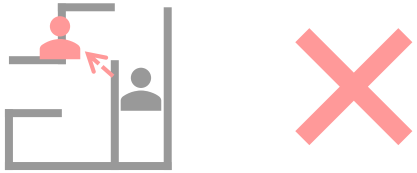
How to find such states at each iteration automatically?

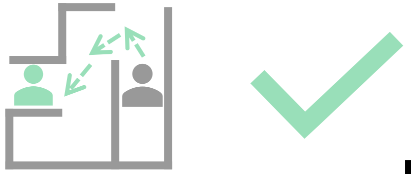
Starting State Distribution p_i at each Iteration i

Good Starting State $s \in \mathbb{R}^d$?

- Idea 1.1 Select Starting States Uniform at Random
- Idea 1.2 Select Starting States Uniform at Random
  + Estimate P(success)
- Idea 2.1 Sample feasible States Nearby p_{i-1}

$s' = s + \varepsilon \quad , \; \varepsilon \sim \mathcal{N}(0, \Sigma)$ $\qquad s' = s \rightarrow a_1 \rightarrow a_2 \rightarrow \ldots \rightarrow a_T \quad , a_t \sim \mathcal{N}(0, \Sigma)$



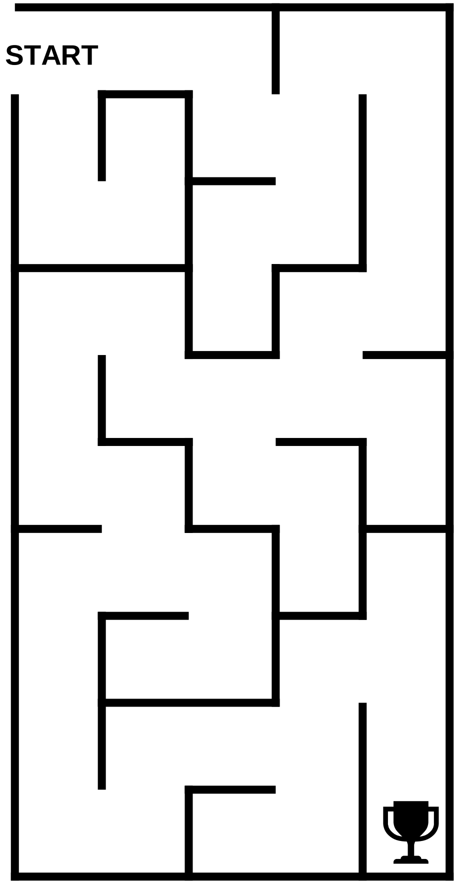


Reverse Curriculum Generation

- Idea 2.2 Sample feasible States Nearby p_{i-1}
  + Keep only Good States after Policy Training

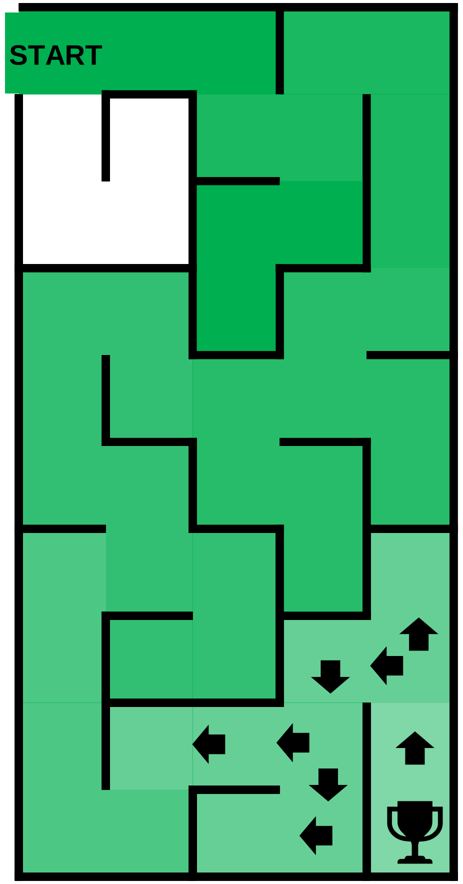
~

Starting State Distribution p_i – Original p_{start}



Original Starting State Distribution $p_{start} = \{start\}$

Starting State Distribution p_i – Original p_{start}



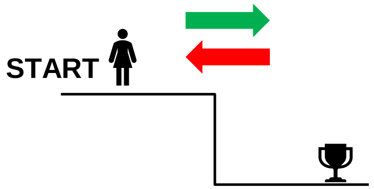
Original Starting State Distribution $p_{start} = \{start\}$

$$p_1 = \{\blacksquare\}$$
$$p_2 = \{\blacksquare\ \blacksquare\}$$
$$p_3 = \{\blacksquare\ \blacksquare\ \blacksquare\}$$
$$p_4 = \{\blacksquare\ \blacksquare\ \blacksquare\ \blacksquare\}$$
$$p_5 = \{\blacksquare\ \blacksquare\ \blacksquare\ \blacksquare\ \blacksquare\}$$
$$p_6 = \{\blacksquare\ \blacksquare\ \blacksquare\ \blacksquare\ \blacksquare\ \blacksquare\}$$
$$p_7 = \{\blacksquare\ \blacksquare\ \blacksquare\ \blacksquare\ \blacksquare\ \blacksquare\ \blacksquare\} \qquad start \in p_7$$

Optimal Policy under start distribution p_i is also optimal under any start distribution p_{start} as long as their support coincide.

Assumptions

- Reset Agent into any Starting State
- Must know at least one Goal State (Goal Oriented Task) state?
- Environment without Irreversibilities



Policy Training

$starts_{\text{old}} \leftarrow [s^g]$
$starts,\ rews \leftarrow [s^g],\ [1]$
for $i \leftarrow\ 1$ **to** $Iter$ **do**
  $starts \leftarrow \texttt{SampleNearby}(starts,\ N_{\text{new}})$
  $starts.\text{append}[\text{sample}(starts_{\text{old}}, N_{\text{old}})]$
  $\rho_i \leftarrow \text{Unif}(starts)$
  $\pi_i,\ rews \leftarrow \texttt{train\_pol}(\rho_i,\ \pi_{i-1})$
  $starts \leftarrow \texttt{select}(starts,\ rews, R_{\text{min}}, R_{\text{max}})$
  $starts_{\text{old}}.\text{append}[starts]$
end

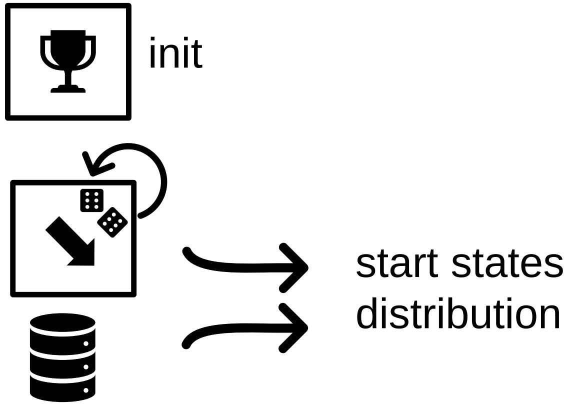
Policy Training

$starts_{\text{old}} \leftarrow [s^g]$
$starts,\ rews \leftarrow [s^g],\ [1]$
for $i \leftarrow 1$ **to** $Iter$ **do**
  $starts \leftarrow \texttt{SampleNearby}(starts,\ N_{\text{new}})$
  $starts.\text{append}[\text{sample}(starts_{\text{old}}, N_{\text{old}})]$
  $\rho_i \leftarrow \text{Unif}(starts)$
  $\pi_i,\ rews \leftarrow \texttt{train\_pol}(\rho_i,\ \pi_{i-1})$
  $starts \leftarrow \texttt{select}(starts,\ rews, R_{\text{min}}, R_{\text{max}})$
  $starts_{\text{old}}.\text{append}[starts]$
end

init

Policy Training

$starts_{\text{old}} \leftarrow [s^g]$
$starts,\ rews \leftarrow [s^g],\ [1]$
for $i \leftarrow 1$ **to** $Iter$ **do**
 $starts \leftarrow \texttt{SampleNearby}(starts,\ N_{\text{new}})$
 $starts.\text{append}[\text{sample}(starts_{\text{old}}, N_{\text{old}})]$
 $\rho_i \leftarrow \text{Unif}(starts)$
 $\pi_i,\ rews \leftarrow \texttt{train\_pol}(\rho_i,\ \pi_{i-1})$
 $starts \leftarrow \texttt{select}(starts,\ rews, R_{\text{min}}, R_{\text{max}})$
 $starts_{\text{old}}.\text{append}[starts]$
end



Policy Training

$starts_{\text{old}} \leftarrow [s^g]$

$starts,\ rews \leftarrow [s^g],\ [1]$

for $i \leftarrow 1$ **to** $Iter$ **do**

$\quad starts \leftarrow \texttt{SampleNearby}(starts,\ N_{\text{new}})$

$\quad starts.\text{append}[\text{sample}(starts_{\text{old}}, N_{\text{old}})]$

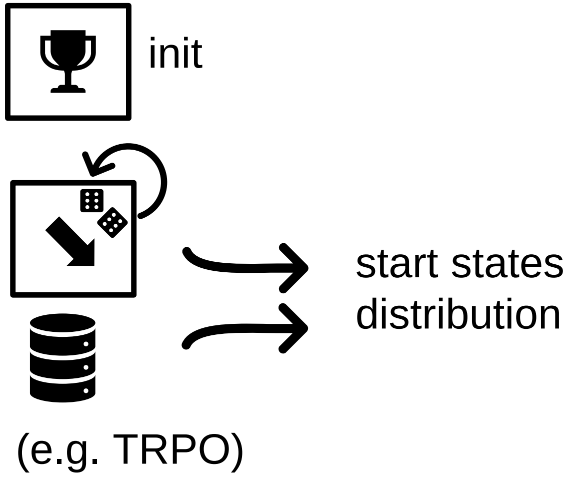
$\quad \rho_i \leftarrow \text{Unif}(starts)$

$\quad \pi_i,\ rews \leftarrow \texttt{train\_pol}(\rho_i,\ \pi_{i-1})$

$\quad starts \leftarrow \texttt{select}(starts,\ rews, R_{\text{min}}, R_{\text{max}})$

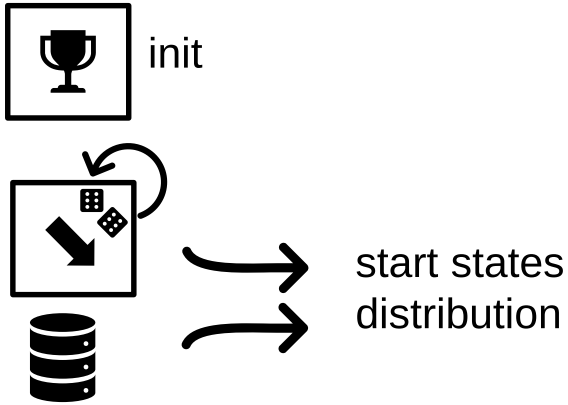
$\quad starts_{\text{old}}.\text{append}[starts]$

end



Policy Training

```
starts_old ← [s^g]
starts, rews ← [s^g], [1]
for i ← 1 to Iter do
    starts ← SampleNearby(starts, N_new)
    starts.append[sample(starts_old, N_old)]
    ρ_i ← Unif(starts)
    π_i, rews ← train_pol(ρ_i, π_{i−1})
    starts ← select(starts, rews, R_min, R_max)
    starts_old.append[starts]
end
```

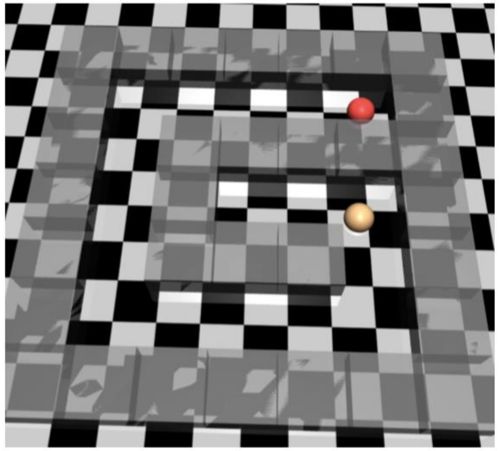


(e.g. TRPO)

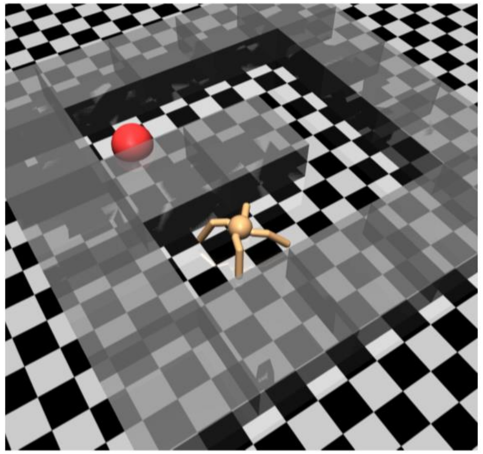
too easy ✗
adequate level ✓
too hard ✗

Experiments

Point – Mass Maze

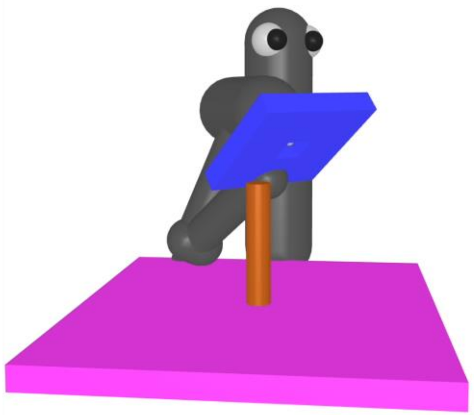


Ant Maze

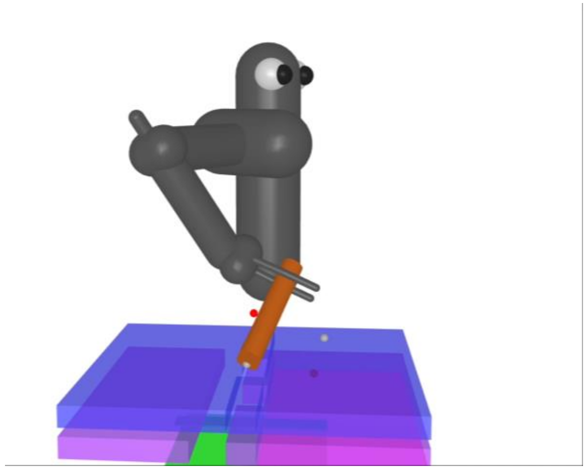


→ Focus

Ring on Peg

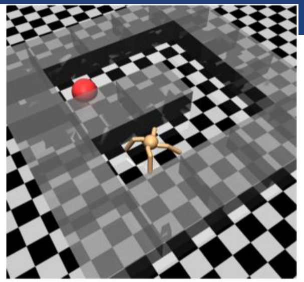


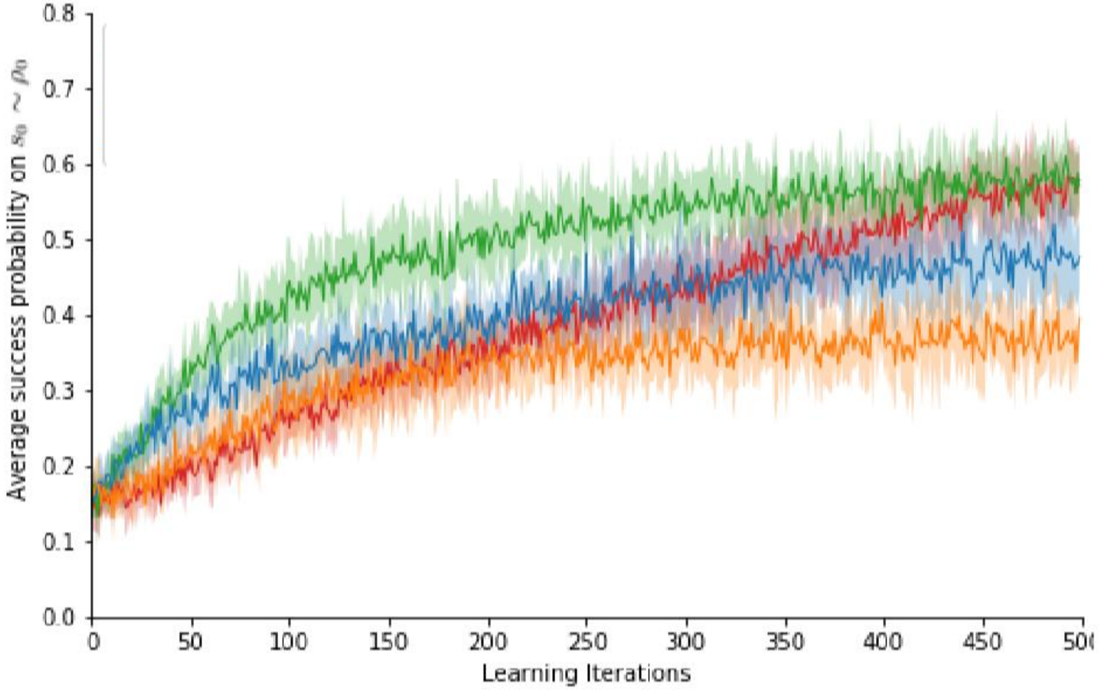
Key Insertion



→ Focus

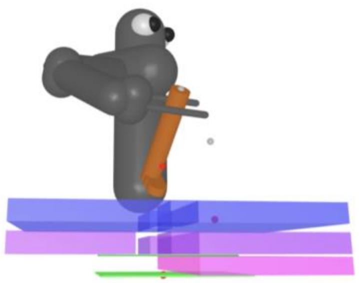
Experiments: Ant Maze

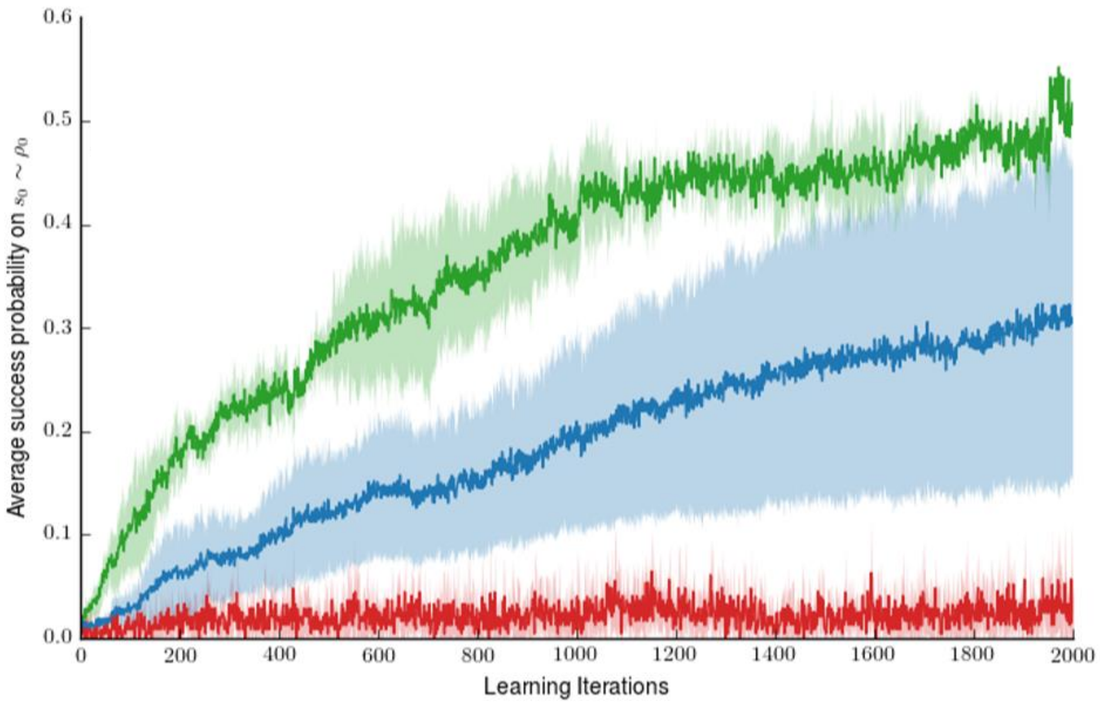




Start States: Reverse Curriculum
Start States: Uniform Sampling (Baseline)
Start States: Reverse Curriculum Ablation (without selecting good starts)
Asymmetric Self-Play

Experiments: Key Insertion





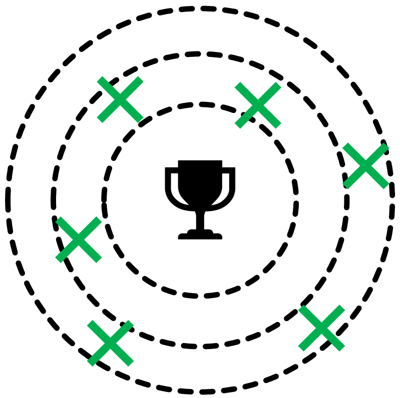
Start States: Reverse Curriculum

Start States: Reverse Curriculum Ablation (without selecting good starts)

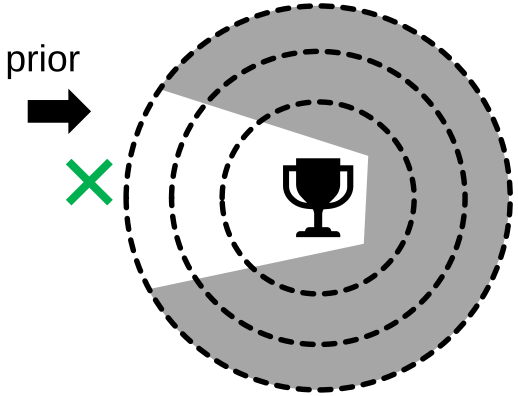
Start States: Uniform Sampling (Baseline)

Potential Problems

- Starting State Distribution evolves uniformly around Goal State



Ideal

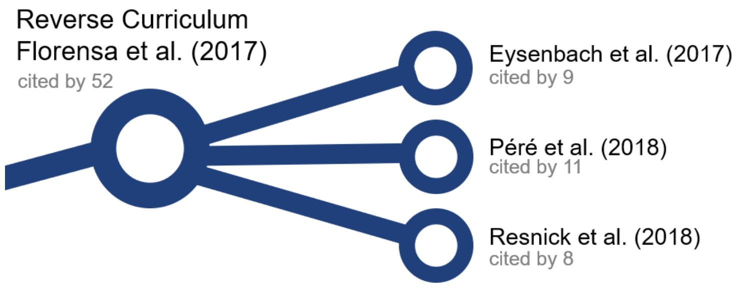


“Wasted” Exploration

original starting states of task

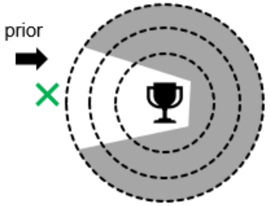
→ cannot incorporate prior information

Reverse Curriculum: Follow Up Work



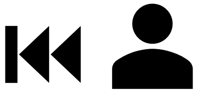
Potential Problems

- Uniform Exploration around Goal State



  → Backplay (Resnick et al. 2018)

Assumptions

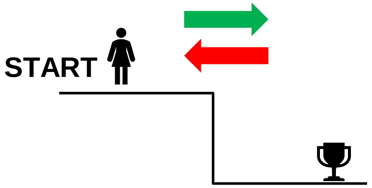
- Reset Agent into any Starting State 

  → Leave no Trace: Learning to Reset for Safe and Autonomous RL (Eysenbach et al. 2017)

- Must know at least one Goal State  state?

  → Unsupervised Learning of Goal Spaces for intrinsically motivated goal exploration (Péré et al. 2018)

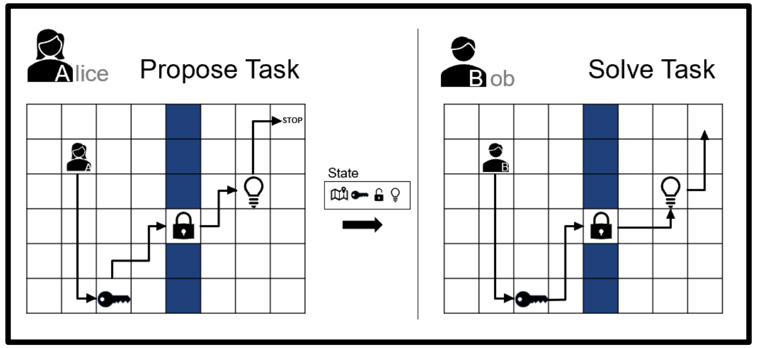
- Environment without Irreversibilities



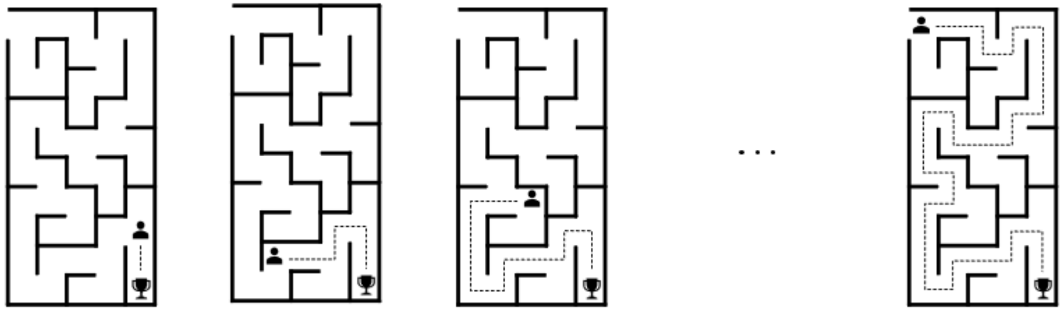
  → Backplay (Resnick et al. 2018)

Conclusion

Intrinsic Motivation and Automatic Curricula via Asymmetric Self-Play



Reverse Curriculum Generation for Reinforcement Learning



+ Conceptually Simple and Elegant
+ Automatic Curriculum (only few hyperparameter)
+ General Framework (exploration bonus)

- Results not super impressive
- Bob can overpower Alice
- Continuous Action Space

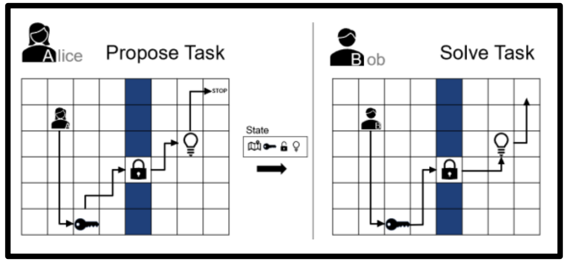
- Reversibility Assumption
- Uniform Exploration around Goal State

ETH zürich

Curriculum Learning (Bengio et al. 2009)

Intrinsic Motivation and Automatic Curricula via Asymmetric Self-Play (Sukhbaatar et al. 2017)

Idea



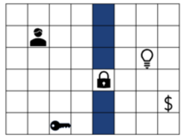
Algorithm

```
Function SelfPlayEpisode(π_Alice, π_Bob, s_0):
    s', t_Alice ← proposeTask(π_Alice, s_0);
    t_Bob ← solveTask(π_Bob, s_0, s');

    R_Alice ← γ max(0, t_Bob − t_Alice);
    R_Bob ← −γ t_Bob;

    π_Alice ← updatePolicy(π_Alice, R_Alice);
    π_Bob ← updatePolicy(π_Bob, R_Bob);
End Function
```

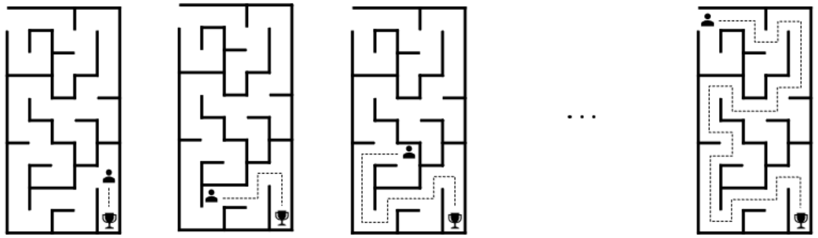
Experiments



Problems

Reverse Curriculum Generation for Reinforcement Learning (Florensa et al. 2017)

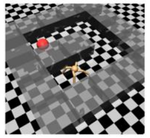
Idea

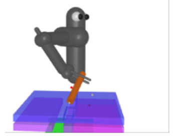


Algorithm

```
starts_old ← [s^g];
starts, rews ← [s^g], [1];
for i ← 1 to Iter do
    starts ← SampleNearby(starts, N_new);
    starts.append[sample(starts_old, N_old)];
    ρ_i ← Unif(starts);
    π_i, rews ← train_pol(ρ_i, π_{i−1});
    starts ← select(starts, rews, R_min, R_max);
    starts_old.append[starts];
end
```

Experiments





Problems

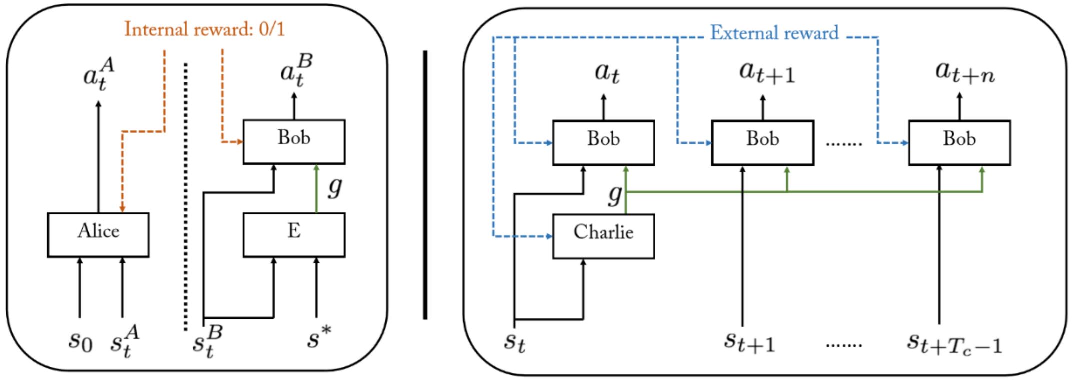
- Learning Goal Embeddings via Self-Play for Hierarchical RL (Sukhbaatar et al. 2018)

- Backplay (Resnick et al. 2018)
- Leave no Trace (Eysenbach et al. 2017)
- Learning of Goal Spaces (Péré et al. 2018)

Backup Slides

Learning Goal Embeddings via Self-Play for Hierarchical RL (Sukhbaatar et al. 2018)

Training Scheme



Self-play pre-training

Fine-tuning on target task

- Use Self-Play to learn Goal Embedding E (Transform State to Goal)
- Introduce 3rd Actor: Charlie
  - Charlie responsible for high level policy in HRL
  - Charlie uses learned embeddings to communicate task to Bob
  - Bob responsible for low level policy

Adjustments Self-Play Phase

- number of steps taken by Alice and Bob are fixed to T_A and T_B respectively
- break episodes into multiple shorter segments (if Bob succeeds, Alice continues from last position instead of start)

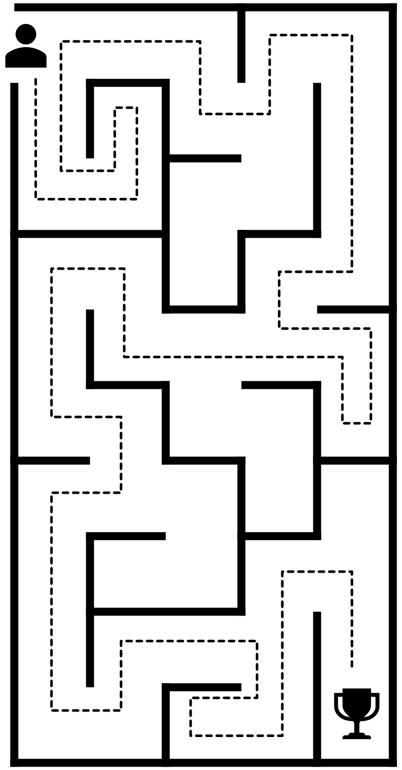
  ➔ more exploration, while keeping Bob's policy manageable for Charlie
- 0/1 reward for Bob (instead of time)
- entropy regularization in loss

$$L_A = \mathbb{E}_{a_t^A \sim \pi_A}[-R_A - \beta \mathrm{H}(\pi_A(s_t^A))]$$

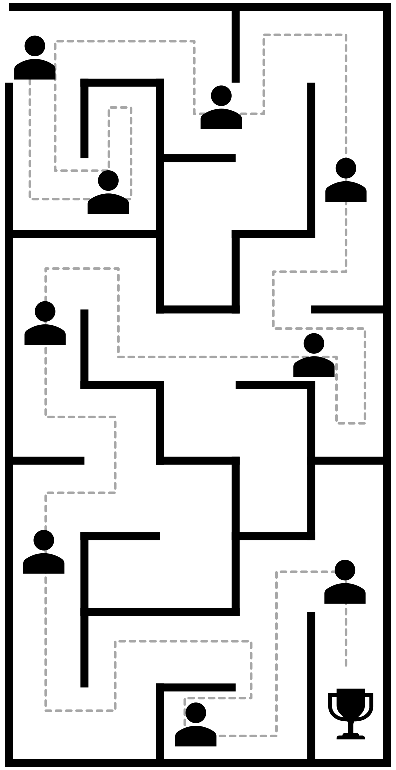
$$L_B = \mathbb{E}_{a_t^B \sim \pi_B}[-R_B] + \alpha \mathbb{E}_{a_t^A \sim \pi_A}[-\log(\pi_B(a_t^A | s_t^A))]$$

Backplay (Resnick et al. 2018)

One Good Enough Demonstration

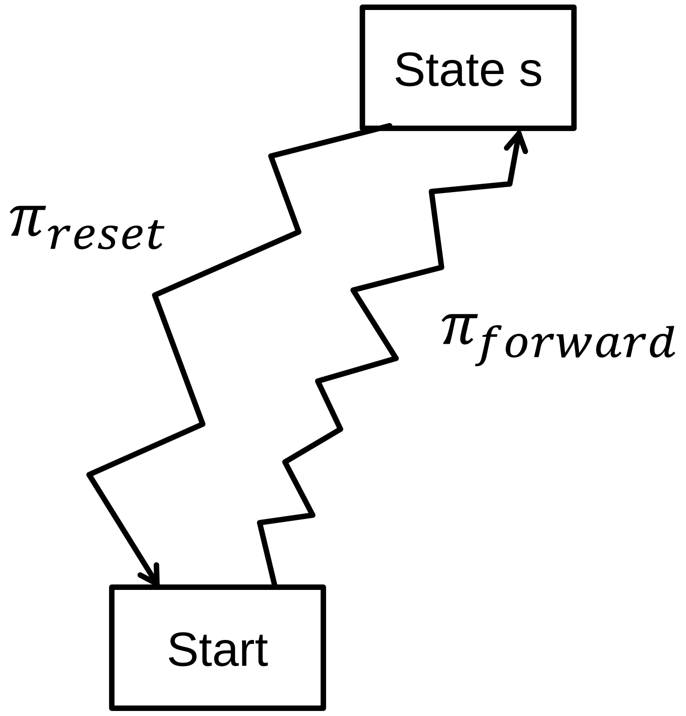


Curriculum of Starting States

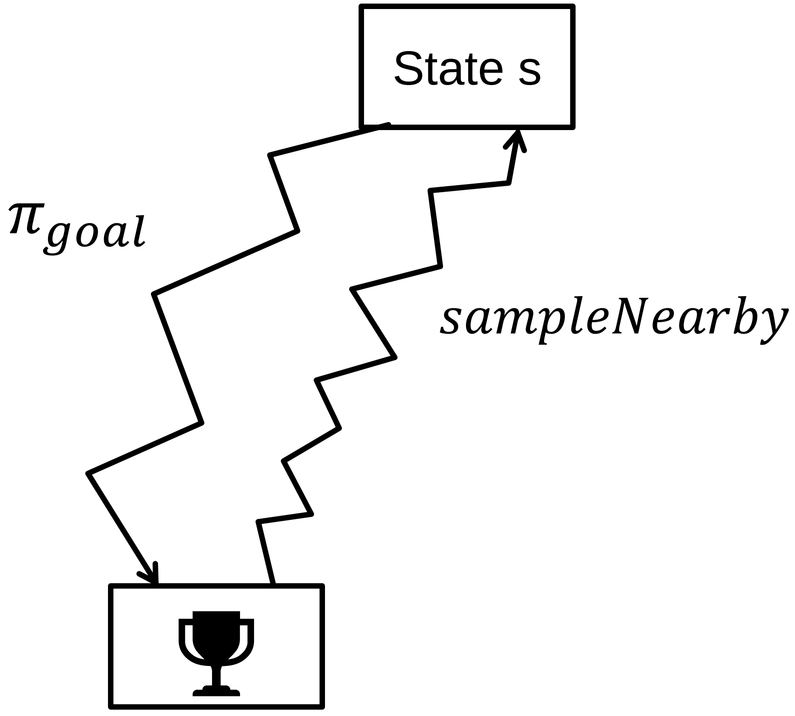


Leave no Trace: Learning to Reset for Safe and Autonomous RL (Eysenbach et al. 2017)

Learn both Forward and Reset Policy



Similarity to Reverse Curriculum Learning



Unsupervised Learning of Goal Spaces for intrinsically motivated goal exploration (Péré et al. 2018)

2-stage approach

- Perceptual Learning Stage:

  Deep learning algorithms use passive raw sensor observations of world changes to learn corresponding latent space

- Goal Exploration Stage:

  Sampling goals in this latent space

Asymmetric Self-Play in a Continuous Action Space (Florensa et al. 2017)

- Part of the reason that Asymmetric Self-Play gets stuck in a local optimum is that ***“Alice”*** is represented with a ***unimodal Gaussian distribution***, which is a common representation for policies in continuous action spaces. Thus Alice’s policy tends to ***converge to moving in a single direction***. In the original paper, this problem is somewhat mitigated by using a ***discrete action space***, in which ***a multi-modal distribution for Alice can be maintained***. However, even in such a case, the authors of the original paper also observed that Alice tends to converge to a local optimum.