Multi-Agent Reinforcement Learning in Zombie-Survival Games

Peter Hollows *
stanford@dojo7.com

Mark Presser *
mark.presser95@gmail.com

Abstract

The abstract goes here...

1 Introduction

Reinforcement Learning (RL) methods are seeing increasing use in real-world control systems. In part, the rising popularity of RL methods is due to its widely publicized success on challenging tasks, as well as the availability of quality open-source frameworks containing implementations of state-of-the-art models.

1.1 Closing the Reality Gap

Current RL methods are not sample-effecicient, meaning the policy learner needs a lot of data before an effective policy can be found. Often, there is not enough data available, or it cannot be captured safely (consider an autonomous driver learning the best policy for a near-crash scenario, where it will make mistakes in order to learn). To work around these issues, RL policies intended for real-world control tasks are usually trained in simulated environments before being further trained or deployed in the real-world.

However, simulations often do not capture all the relevant real-world dynamics, so it can be difficult to transition a policy from the simulator to reality. This difficulty, known as the "reality gap", is often mitigated by adding noise to the simulation (to mimic noisy sensors and stochastic dynamics), or by sufficiently improving the simulation to model the relevant dynamics.

This approach has enjoyed success in single-agent RL settings like autonomous aircraft and robotics.

*Group submission for the final paper of XCS229ii, Stanford Center for Professional Development

1.2 Multi-Agent RL

In cooperative settings like factories, or competitive settings like markets, using the simulation technique becomes more challenging; simulating the dynamics of other agents present in the environment is a difficult problem, even in simulation. This is especially the case when the policies of multiple agents are being modified and improved at the same time.

These issues in multi-agent settings motivated a training algorithm called Multi-Agent Deep Deterministic Policy Gradient (MADDPG) (Lowe et al., 2020). MADDPG addresses some issues in multi-agent learning, and allows running experiments that explore the emergence of competitive and cooperative policies in complex environments.

1.3 Overview

We explore MADDPG in the context of a mixed competitive-cooperative simulated environment called "Zombie-Survival". We make several modifications in order to test different scenarios. In these scenarios we show how specific kinds of environmental dynamics can affect changes in the evolution of policies.

In (2) we review theory, RL algorithms, and aspects of agent design. In (3.1) we detail the basic simulation environment. In (3.2) we detail the experimental setup and initial results. In (3.4) we investigate partial observability, and how compression of the observation-space can speed policy discovery. In (3.6) we alter the environment to cause a change in the value-function (keeping reward unchanged) to study the impact on learned policies. In (3.8) we arm the survivors with projectile weapons to investigate how additional capability can make it harder for agents to achieve their objectives. We present methods and results for each scenario, and conclude with a discussion (4) of our findings.

2 Background

2.1 The Markov Decision Process

A Markov Decision Process (MDP) formalizes the interaction of an agent with an environment. At every time step, an *agent* receives observations of the *state* of the *environment*, and sends *actions* to the environment. The environment transitions from one state to the next over time, and actions from the agent can influence which states the environment transitions to. At every time step, the agent receives a *reward* based on the state of the environment. In the case when state transitions are not independent (i.e. the chance of arriving in a state depends on visiting previous states), then agent actions in a state can impact future expected rewards as well as immediate rewards.

RL algorithms attempt to solve the MDP in order to maximize return (reward over time). This can be done by learning a policy (a function that maps from state to action). A policy can often be learned directly, or by learning a value function (a mapping from states and actions to expected returns) from which a policy can be extracted.

2.2 Algorithms

Deep Q-Learning Networks: Q-Learning (Watkins, 1989) is an RL method that makes use of an action-value function $Q^\pi(s, a)$ corresponding to a policy π (where s is the state, and a the action). Deep Q-Learning Networks (DQN) (Mnih et al., 2015) learn the action-value function Q^* corresponding to the optimal policy by minimizing the loss function

$$\mathcal{L}(\theta) = \mathbb{E}_{s,a,r,s'}[(Q^*(s, a|\theta) - y)^2], \quad (1)$$

where $y = r + \gamma \max_{a'} \bar{Q}^*(s', a')$, by repeatedly adjusting the parameters θ of the target action-value function $\bar{Q}$. The Q-function can be expressed as $Q^\pi(s, a) = \mathbb{E}[R|s, a]$, where R_t is the discounted sum of future rewards at time t, given by: $R_t = \sum_{k=t}^{\infty} \gamma(k - t)r_k$. To reduce the variance of estimates for the above expectation, DQN makes use of an Experience Replay (ER) buffer that stores the experiences of many state transitions.

Policy-gradient methods learn a policy π_θ, parameterized by θ, by sampling the gradient of a performance measure $J(\theta)$ with respect to θ.

$$\nabla_\theta J(\theta) = \mathbb{E}_\pi \left[Q^\pi(s, a) \frac{\nabla_\theta \pi_\theta(a|s)}{\pi_\theta(a|s)} \right] \quad (2)$$

REINFORCE (Williams, 1992), a simple policy-gradient algorithm, uses all the steps in an episode to find $Q^\pi(s, a) = \mathbb{E}_\pi[G|s, a]$, where G is the total reward of an episode. Because reward functions sometimes give high values for some states, this method can give high-variance gradient estimates. This can be remedied by using a baseline $b(s)$ (Sutton and Barto, 2018):

$$\nabla_\theta J(\theta) = \mathbb{E}_\pi \left[(G - b(s)) \frac{\nabla_\theta \pi_\theta(a|s)}{\pi_\theta(a|s)} \right] \quad (3)$$

Actor-critic (Bhatnagar et al., 2009) improves on REINFORCE by learning a state-value function with baseline (critic), and an action-value function (actor) concurrently. Actor-critic also has the advantage that the parameters for both functions are updated at every step in an episode. The policy gradient at each time step t can be expressed:

$$\nabla_\theta J(\theta) = \mathbb{E}_\pi \left[\delta_t \frac{\nabla_\theta \pi_\theta(a|s)}{\pi_\theta(a|s)} \right] \quad (4)$$

$$\delta_t = r_{t+1} + \gamma \hat{v}(s_{t+1}, w) - \hat{v}(s_t, w) \quad (5)$$

where γ is a discount parameter, and $\hat{v}(w)$ the state-value function with parameters w.

Deterministic Policy Gradient (DPG) (Silver et al., 2014) algorithms extend policy gradient methods to deterministic policies. A deterministic policy (μ_θ) maps a state to an action, whereas a stochastic policy (π_θ) provides a conditional probability of an action in a given state.

$$\nabla_\theta J(\theta) = \mathbb{E}_s \left[\nabla_\theta \mu_\theta(a|s) \nabla_a Q^\mu(s, a)|_{a=\mu_\theta(s)} \right] \quad (6)$$

DPG methods are often used in continuous control tasks (taking the gradient of Q^μ with respect to a requires the action space to be continuous).

Deep Deterministic Policy Gradient (DDPG) (Lillicrap et al., 2019) combines ideas from DPG and DQN. In DDPG the actor and critic are approximated with neural networks. DDPG uses target networks for both actor and critic, and makes use of an experience replay buffer to store agent trajectories.

2.3 MADDPG

The Markov Decision Process (MDP) framework regards other agents as part of the environment from the perspective of a single agent. From the

perspective of an agent, the influence on rewards from other agents in the environment can make it harder to learn a policy; this is especially so when the environment contains other agents that are also learning policies, because elements of the 'environment' change while the agent is trying to learn them.

Lowe et al., 2020 showed that the chance of sampling a policy-gradient update in an improved direction decreases exponentially with the number of agents being trained in the shared environemnt.

MADDPG addresses this stabilitiy issue by using centralized critics. In MADDPG, the critic $Q_i^{\pi}(x, a_1, a_2, ..., a_N)$ for agent i has access to the actions of all N agents, and some state information x. The critic is often only used in training, so it's possible to deploy MADDPG actors in continuous control tasks where knowledge of other actors' actions are not available. Agents can have different state and action spaces, and be goverened by different reward functions. We use the MADDPG algorithm for all agents within our scenarios.

2.4 Agent Design

When implementing an RL system there are many important design considerations:

- *Algorithm*: Different algorithms are suited to different tasks. A selected algorithm will need to perform well with the selected action-space, state-space, reward function and environmental dynamics.

- *State-space*: The state-space describes how environmental observations are presented to the algorithm. Constructing the correct, relevant abstractions allow an agent to better learn the environment.

- *Action-space*: The action-space describes actions available to the agent. The level of abstraction here should be fine enough for precise control, but general enough for a policy to be learnable.

- *Reward function*: The reward function provides a scalar feedback value (positive or negative) to the agent. Some algorithms can model uncertainty in the reward function (Ghavamzadeh et al., 2016), but most expect a point-estimate rather than a distribution. Crafting a consistent reward function that reflects the exact desires of the designer can be challenging (Leike et al., 2017). Some algorithms can infer the a reward function from feedback (Ng et al., 2000), but most simply try to maximize the explicit reward from a hard-coded function.

Setting	Survivors	Zombies
world size (m)	2x2	2x2
timestep (s)	0.1	0.1
episode length (s)	10	10
num. agents	2	**5**
agent radius (m)	0.05	0.05
agent mass (kg)	1.0	1.0
acceleration ($ms^{-}2$)	**4.0**	3.0
max. speed ($ms^{-}1$)	**1.3**	1.0
bite reward	-10	+10

Table 1: Scenario settings.

In our scenarios we examine the imapct of design choices for state-space and action-space under a fixed reward-function.

3 Scenarios

3.1 Zombie-Survival Game

The Zombie-Survival scenarios cover a set of games simulating survivors and zombies in a zombie-apocalypse scenario. When a zombie is in physical contact with a survivor this results in a 'bite' between zombie and survivor. Zombies attempt to bite survivors and are rewarded for each step they are in contact. Survivors attempt to distance themselves from zombies and are penalized every time they are bitten. Survivors can suffer a bite from multiple zombies in the same time step.

Zombies have the advantage of numbers (there are 5 zombies and 2 survivors), whereas survivors have the advantage of faster accleration and a higher maximum speed. All reward functions are independent, agents do not share reward. The game offers a scenario with asymmetric power dynamics between competitive teams, and allows the emergence of competitive or cooperative behavior within a team.

3.2 Scenario: Baseline

We modify the "Predator-prey" particle environment from Lowe et al., 2020, which implements a multi-agent environment with a continuous action-space, a fully-observable state-space, and basic 2D

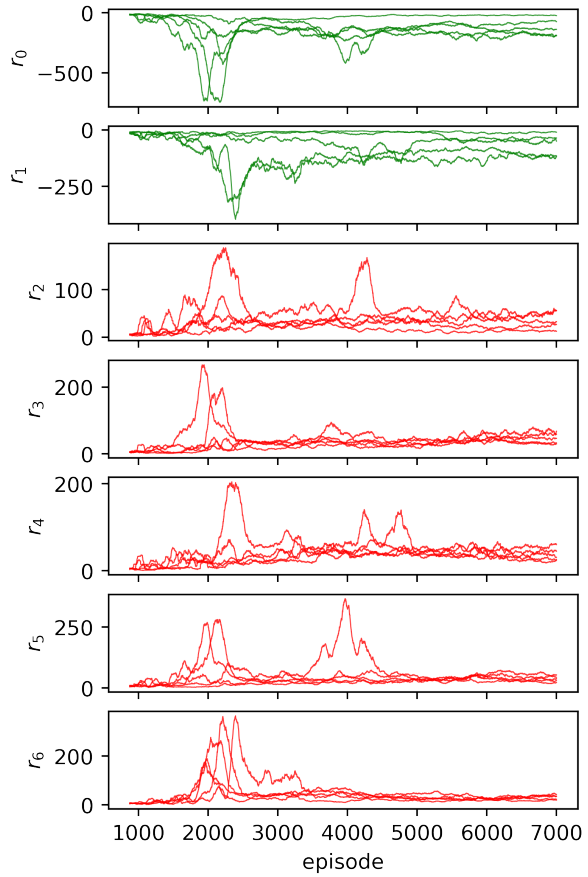

Figure 1: Reward (moving average over 100 episodes) for survivors (green) and zombies (red) for 10 training runs.

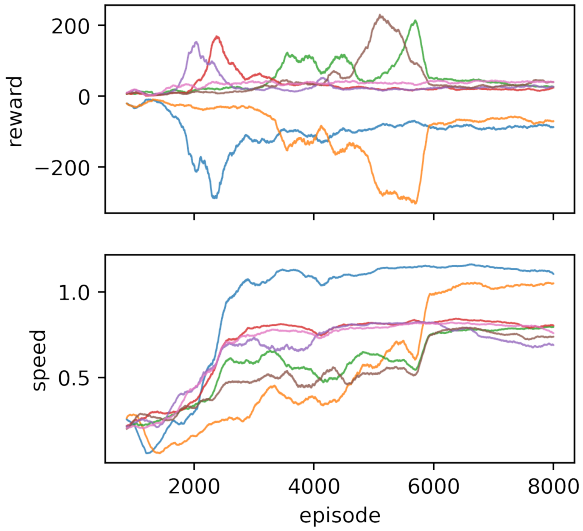

Figure 2: Reward and speed (moving averages over 200 episodes) for a single training run in baseline scenario. Survivors in blue and orange.

physics. Each agent is represented as a circle that has mass, position, velocity, velocity dampening, maximum acceleration, and maximum speed. The predator-prey environment uses reward shaping to discourage agents from moving out of bounds. We remove this additional term in the reward function and implement fixed boundary walls. We implement the biting rules, modify the agent count and adjust the agents' physical capabilities (see Table 1). We run several training jobs for 10,000 steps each to capture the emergence of different policies.

3.3 Results: Baseline

Variability: We observed variability in policy convergence between training runs. Figure 1 shows agent reward graphs over multiple runs. The reward trace for r_5 shows the zombie agent experienced a spike in reward corresponding to a drop in reward for a survivor in r_0 that occurred only once in 10 training runs. Most agents show a similar pattern for at least one run.

Target fixation: On further investigation we discovered the reason for the variability; a zombie usually fixates on a target, but zombies learn policies at different rates. Figure 2 shows speed and reward over time for a single training run. We see that one survivor is not pursued as often to start with, but then recieves increasing attention from zombies learning to persure it. Following this attention the survivor learns to move much faster. We observed similar patterns in other runs.

In the baseline state-space, every agent is unique, so all agents have to learn their expected reward relating to each other. From an agent's perspective, when considering reward the most important thing about an agent is whether they are a survivor or a zombie, not necessarily which survivor or zombie. We'd prefer agents learn general policies that consider other agents exchangeable within a team.

3.4 Scenario: Anonymity

In the *anonymity* scenario we modify the *baseline* scenario to introduce partial observability. We reorder agents' state vectors each step so that agents within teams are ordered by distance to the observing agent. To a survivor, a zombie now appears the same as any other zombie, and does not have identity by virtue of index in the state vector. This reduces the size of the state-space considerably, and allows agents to learn policies that focus on nearby agents.

We expect both survivors and zombies to learn

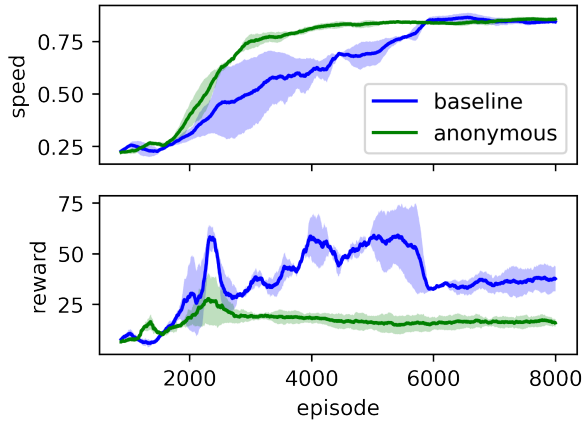

Figure 3: Speed and reward evolution for baseline and anonymous scenarios. Results taken as moving averages over 200 episodes prior to aggregation.

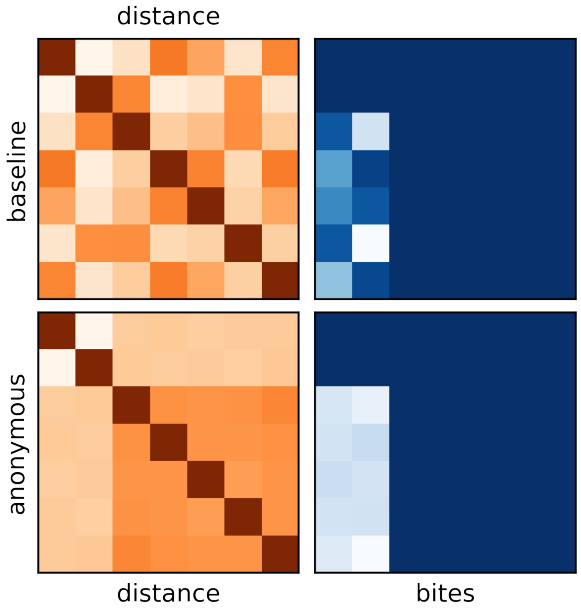

Figure 4: Distance and bite count comparison between baseline and anonymous scenarios. Plot colors indicate average distance and average bite count per episode average across the last 5000 episodes of a training run. Both axes are agent indexes, and color indicates relative value with white being the highest value.

more general policies with faster convergence than the baseline. We also expect to see zombies chasing which ever survivor is easiest to chase, rather than fixating on a single survivor.

3.5 Results: Anonymity

We observed faster convergence to a stable policy (as judged by average speed and reward) when compared to baseline (see Figure 3). We also observed more uniform persuit and biting (see Figure 4).

3.6 Scenario: Health

In the *health* scenario we modify the *anonymity* scenario to introduce a health mechanism for survivors. Survivors start each episode with 100% health, which for survivors is discounted ($\gamma = 0.99$) every bite. An agent's health level is added to the private observation space for the agent. Reward remains the same for bites regardless of health levels, however maximum speed at each step is limited to the fraction of health remaining. If a survivor is bitten too often, they become slower than the zombies and risk being overrun, leading to a large amount of bites for the remainder of the episode.

This scenario does not alter the reward function, but increases the variance and magnitude of state-value estimates for both zombies and survivors. Additionally, this scenario shifts the balance of power heavily toward the zombies. We expect to see survivor policies emerge that are more risk-averse, and to see zombies elect to chase slower-moving survivors.

3.7 Results: Health

Breakout: Compared to the *anonymous* scenario, agents in the *health* scenario were slower on average (see Figure 5, plot 1). It's interesting to note this slowdown occurred before the reward ramp-up due to zombies successfully immobilizing survivors. In part the slower speeds can be explained through partial health reductions, but rendering the simulation frames provided another explanation. In previous scenarios, survivors would run into walls at maximum speed to bounce back through the persuring group of zombies without losing speed. The bouncing strategy sacrifices minimal reward and allows a survivor to keep their distance ahead of the pursuing zombies after the manouver. In the *health* scenario, this strategy becomes unsustainable, as every contact with a zombie (during the bounce) reduces the survivors maximum speed leading to a very bad outcome after only a few bounces.

With health to consider, survivors opted for more impressive fake-and-dodge manouvers to avoid contact altogether. Often this would involve pausing to allow zombies to move closer to their target, then making use of the survivors' higher acceleration to escape at an angle orthagonal to the zombie group's velocity.

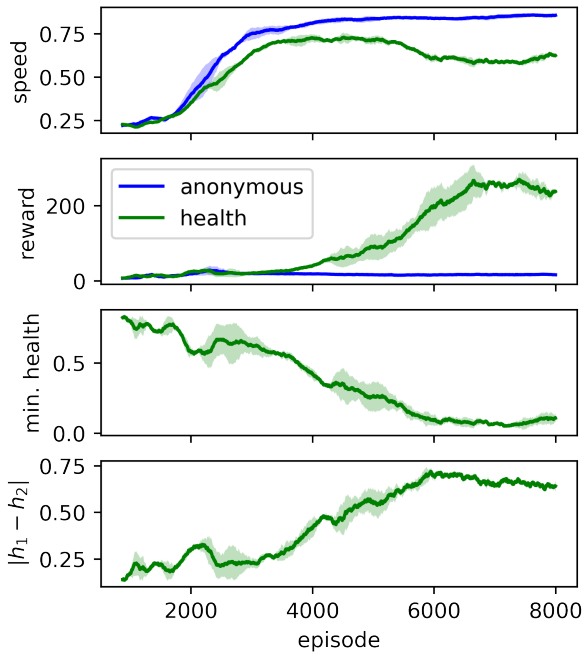

Figure 5: Comparison between anonymous and health scenario policy evolution. Results taken as moving averages over 200 episodes prior to aggregation. Plot 1: average speed of all agents. Plot 2: average reward of zombies. Plot 3: lowest health of all survivors at the conclusion of an episode. Plot 4: the difference between survivor health at the conclusion of an episode.

Game-over: We measured a drastic increase in average zombie reward (see Figure 5, plot 2) due to zombies learning to immobilize and bite survivors. In most cases, one survivor would succumb to the zombies first. Zombies learn that moving in the direction of the slower-moving survivor leads to states with higher expected value, so they collaborate in slowing down one survivor, leaving the other survivor to escape out of harm's way.

3.8 Scenario: Armed

In the *armed* scenario we attempt to rebalance the game by modifying the *health* scenario to arm the survivors with firearms. Each armed agent has a firearm that fires 2 shots simultaneously when activated. On firing, two rays are cast with a small random spread ($\angle ray \sim \mathcal{N}(\angle aim, 2°)$). When a ray first intersects with another agent it is counted as a 'hit'. An agent experiencing a hit has their health level reduced (by 0.2). Armed agents require time to reload (1 step). Both zombies and survivors can be rendered immobile through reduced health by being hit too often.

We add a normalized current heading and relative heading of locations of other agents to an armed agent's state space, as well as a normalized time remaining until the firearm is reloaded.

Reloading is automatic and does not require an action. The continuous aiming action (left and right) modifies angular velocity of aim directly up to a maximum speed of 1 revolution per second. The firing mechanic is implemented as a 'trigger pressure' which determines the probability of a loaded firearm discharging in a given timestep. Firing probability is implemented as the sigmoid function $p(fire) = \frac{1}{1+e^{20(x-0.5)}}$, where x is pressure on the trigger.

We expect introducing the arms mechanic will move the balance of power back toward the survivors. We're not sure if the social dilemma presented will cause the survivors to fire on one another more often than the zombies, but make no adjustment to the reward function to discourage it.

3.9 Results: Armed

TODO: Mark

4 Discussion

References

Shalabh Bhatnagar, Richard S Sutton, Mohammad Ghavamzadeh, and Mark Lee. 2009. Natural actor–critic algorithms. *Automatica*, 45(11):2471–2482.

Mohammad Ghavamzadeh, Shie Mannor, Joelle Pineau, and Aviv Tamar. 2016. Bayesian reinforcement learning: A survey. *arXiv preprint arXiv:1609.04436*.

Jan Leike, Miljan Martic, Victoria Krakovna, Pedro A. Ortega, Tom Everitt, Andrew Lefrancq, Laurent Orseau, and Shane Legg. 2017. Ai safety gridworlds.

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2019. Continuous control with deep reinforcement learning.

Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. 2020. Multi-agent actor-critic for mixed cooperative-competitive environments.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. 2015. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

Andrew Y Ng, Stuart J Russell, et al. 2000. Algorithms for inverse reinforcement learning. In *Icml*, volume 1, page 2.

David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. 2014. Deterministic policy gradient algorithms. In *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 387–395, Bejing, China. PMLR.

Richard S. Sutton and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA.

Christopher J. C. H. Watkins. 1989. *Learning from Delayed Rewards*. Ph.D. thesis, Kings College.

Ronald J. Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Machine Learning*, pages 229–256.

A Appendix

TODO: Mark