Multi-Agent Reinforcement Learning in Zombie-Survival Games

Peter Hollows [*]
stanford@dojo7.com

Mark Presser [*]
mark.presser95@gmail.com

Abstract

We investigate challenges of agent design using multi-agent zero-sum game called "Zombie-Survival" in which we train 7 agents (2 survivors, 5 zombies) under different scenarios using the same reward function to investigate how different environmental dynamics can lead to very different learned policies.

We show how state-space compression can improve the stability of a learned policy, how a health mechanism that impacts maximum speed can change the dominant strategy, and how arming the survivors with projectile weapons can lead to a social dilemma.

Ultimately the survivors do not prevail.

1 Introduction

Reinforcement Learning (RL) methods have enjoyed recent popularity, in part due to widely publicized successes on challenging tasks (Silver et al., 2018; Badia et al., 2020; Vinyals et al., 2019; Barth-Maron et al., 2018), and the availability of quality open-source frameworks containing implementations of state-of-the-art models (Brockman et al., 2016; Hoffman et al., 2020; Hill et al., 2018).

Though RL methods exceed human-level performance in some tasks such as simulated games, network optimisation (Chen et al., 2018; Mao et al., 2019; Valadarsky et al., 2017) and even flying helicopters (Abbeel et al., 2006), there are many challenges discouraging the real-world deployment of RL systems (Dulac-Arnold et al., 2019).

1.1 The Reality Gap

One of these challenges is that most 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 (Jakobi et al., 1995) to the simulation (to mimic noisy sensors and stochastic dynamics), or by sufficiently improving the simulation to model the relevant dynamics.

1.2 Multi-Agent RL

When designing an RL agent for use in cooperative settings like factories, or competitive setting like markets, we have to allow for the fact that other agents may be present in the environment and may change their behavior in response to the activity of other agents. Training multiple agents concurrently in simulation can capture some of the dynamics around agents learning to respond to each other's actions. However, there are technical challenges to do with non-stationarity that prevent success in the naive approach of using multiple, single-agent algorithms.

These technical issues motivated the Multi-Agent Deep Deterministic Policy Gradient (MADDPG) algorithm (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 the impacts of environmental dynamics on policy evolution in the multi-agent RL setting.

[*]Group submission for the final paper of XCS229ii, Stanford Center for Professional Development. Code available at https://github.com/captainpete/zombie-survival.

We use MADDPG to train multiple agents in a mixed competitive-cooperative simulated environment called "Zombie-Survival". We make several modifications to the environment in order to observe agent behavior under different scenarios.

In (2) we review theory, training 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 this 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 of our findings (4).

2 Background

2.1 The Markov Decision Process

A Markov Decision Process (MDP) formalizes the interaction of an agent with an environment (Sutton and Barto, 2018). 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), an agent's 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 is 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

DQN 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

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

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

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

Learning a policy directly is useful in situations where extraction of a policy from a value function would be difficult (consider the scenario of a very large, or continuous action space).

REINFORCE (Williams, 1992) is a simple Monte-Carlo policy-gradient algorithm that uses all the steps in an episode to sample $Q^\pi(s,a) = \mathbb{E}_\pi[G|s,a]$ (where G is the expected return of an episode) to make a policy update. 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):

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

Actor-critic The Actor-Critic algorithm (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:

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

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

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

DPG The Deterministic Policy Gradient (DPG) algorithm (Silver et al., 2014) 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 suited to continuous control tasks (taking the gradient of Q^μ with respect to a requires the action space to be continuous).

DDPG The Deep Deterministic Policy-Gradient algorithm (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

Under MDP additional agents are regarded as part of the environment. From the perspective of an agent, the influence on rewards from other agents can make it harder to learn a policy; this is especially so when the environment contains other agents that are also learning policies, because the other agents can cause the environment's dynamics to change while the agent is trying to learn them.

Lowe et al., 2020 showed that the chance of sampling a policy-gradient update for an agent in the direction of higher expected return 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.

Under MADDPG, agents agents are permitted to have different state and action spaces, and to be goverened by different reward functions. We use the MADDPG algorithm for all agents within our scenarios.

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.

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, whereas survivors have the advantage of faster accleration and a higher maximum speed (Table 1).

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 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 (Table 1). We run multiple experiments in which we train agents over 10,000 episodes.

3.3 Results: Baseline

Variability We observed variability in policy convergence between training runs. Figure 9 shows agent reward graphs over multiple runs. The reward trace for r_5 shows the zombie agent experienced a

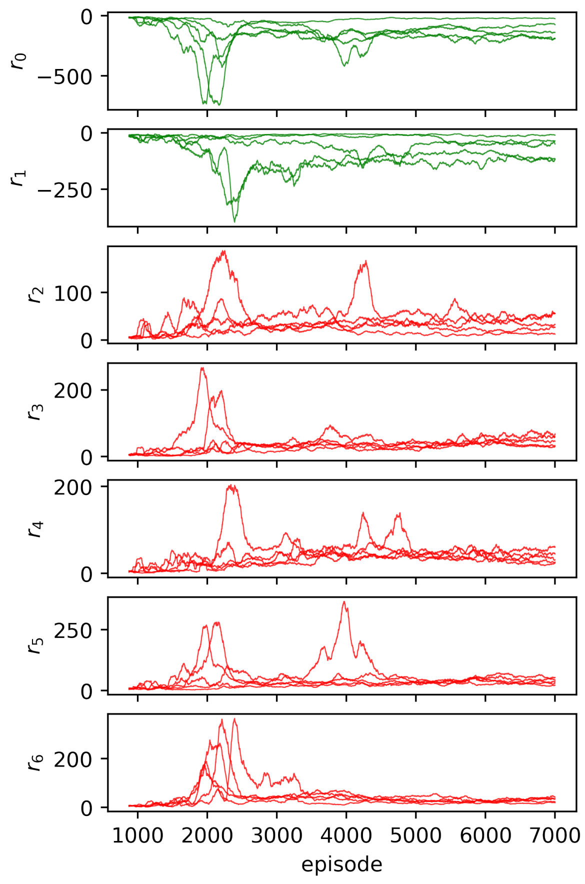

Figure 1: Reward (moving average over 100 episodes) for survivors (green) and zombies (red) for 10 training runs. Agents experience policy shifts between episode 2000 to 6000. Policies stabilize after 7000 episodes.

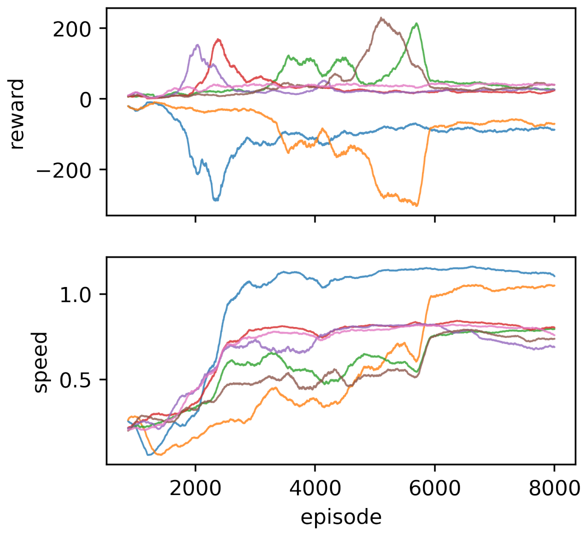

Figure 2: Reward and average speed (moving averages over 200 episodes) for all agents in a single training run of the *baseline* scenario. Survivors (blue and orange) learn faster movement at different times.

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 the expected reward specific to every other agent. We propose that from an agent's perspective, when considering reward the most important thing about another 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 as 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

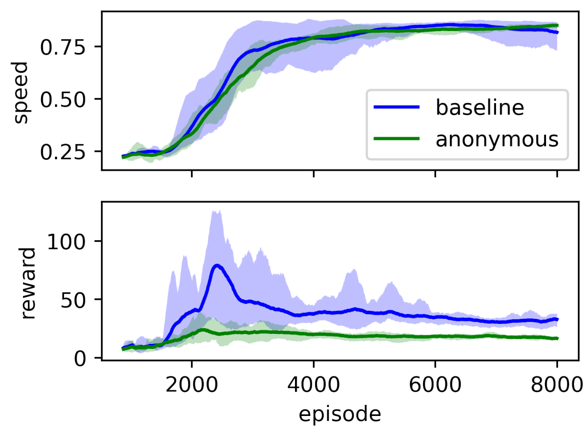

Figure 3: Speed and reward evolution for baseline and anonymous scenarios over 8 training runs. Results taken as moving averages over 200 episodes prior to aggregation.

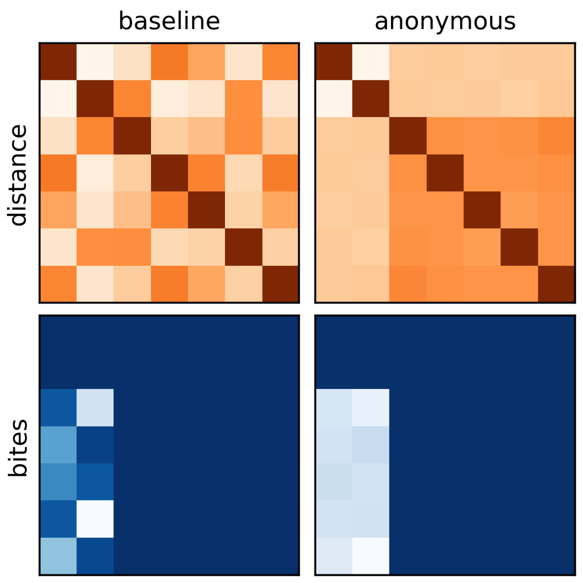

Figure 4: Distance and bite count comparison between *baseline* and *anonymous* scenarios. Plot colors indicate average distance and average bite count per episode, averaged across the last 5000 episodes of a single training run. Both axes are agent indexes, and shade of color indicates relative value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

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 more general policies with faster convergence than in the *baseline* scenario. 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 less variance in 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 (one bite per zombie in contact per time step). An agent's health level is added to the private observation space for the agent. 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 6, plot 1). This slowdown occurred before the reward ramp-up caused by zombies learning to immobilize survivors. In part the initial 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 reward but 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 reduces the survivors maximum speed leading to a very bad outcome after only a few bounces.

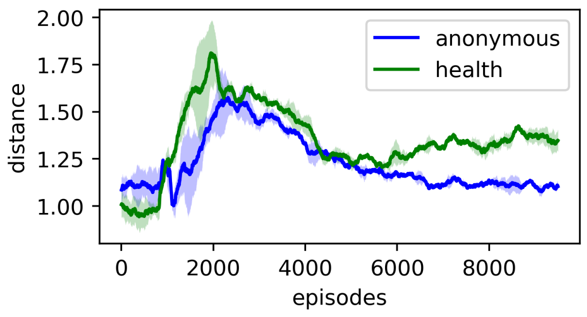

Figure 5: Distance between survivors in the *health* scenario. Survivors learn to keep their distance from one another more than in the *anonymous* scenario. Results taken as moving averages over 200 episodes prior to aggregation.

Game-over We measured a drastic increase in average zombie reward (Figure 6, 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 (Figure 6, plot 4).

Survivor competition Survivors that were not the focus of 'zombie collaboration' learned to stay out of the way by distancing themselves from the other survivor and their zombie entourage (see Figure 5).

In some episodes, survivors would trade places as the focus of zombie attention. A fast moving survivor with zombies in persuit would move directly toward the other survivor (who was often be keeping their distance in a corner). The first survivor would have higher velocity, and use it to risk a wall-bounce manouver near the second slower-moving one. The first survivor in this instance would retain the speed required to escape while the zombies switched their persuit to overrun the second survivor. We found it difficult initially to predict which survivor would ultimately attract the zombies, but around episode 6000 the survivors started to better avoid this trap by moving away from areas that would soon receive zombies.

The introduction of the health dynamic moved the simulation closer to what we consider a 'more realistic' zombie-survivor scenario. The change in state-space and environmental dynamics lead to a very different movement policy for agents without

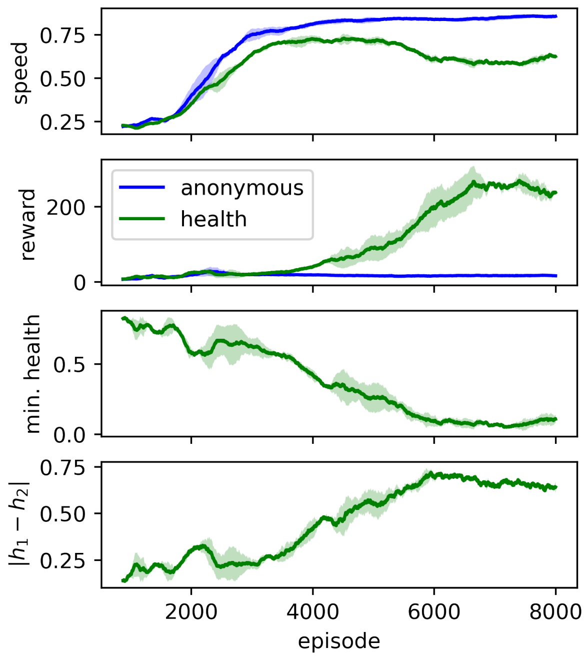

Figure 6: Comparison between *anonymous* and *health* scenario policy evolution. Results taken as moving averages over 200 episodes prior to aggregation. Plot 1: average speed of all agents. Plot 2: average reward of zombies. Plot 3: lowest health of all survivors at the conclusion of an episode. Plot 4: the difference between survivor health at the conclusion of an episode.

changing the reward function.

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 ($\angle ray \sim \mathcal{N}(\angle aim, 2^{\circ})$). 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 a each survivor's state-space, as well as a normalized time remaining until the firearm is reloaded. Zombies and the other survivor do not observe a survivor's aim or reloading states.

Reloading is automatic and does not require an action. The continuous aiming action (left and right) modifies angular velocity of aim directly (rather than through acceleration) 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 $p(fire) = \frac{health}{1+e^{20(x-0.5)}}$, where x is pressure on the trigger (note the ability for a survivor to fire depends on their health).

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

As expected, the introduction of projectile weapons in the *armed* scenario increased survivor health and rewards on average (Figure 7, plots 2-3), shifting the balance of power back toward the survivors. Relative to the *health* scenario, survivors took longer to learn to increase average speed (Figure 7, plot 1). This can be explained by the introduction of a mechanicm for reducing zombie health (the guns), leading to lower average maximum speeds for zombies, decreasing the urgency with which survivors must learn to move faster.

Interestingly, the initial period (episodes 2000-4000) that survivors in the *health* scenario used to learn increased speed is used by survivors in the *armed* scenario to learn increased fire rate.

Specialization During this phase we found one survivor was often faster than the other at learning to hit targets (Figure 8, plot 1). The difference in rate of fire in survivors persists over a period and may be indicative of role specialization, where one survivor learns a benefit from shooting while the other finds more benefit in learning evasion.

More practice required After the initial phase, average survivor speed increased to a similar level to that observed in the *health* scenario. Survivors were able to hold back the zombies initially, but the zombie's power-in-numbers eventually overcame the survivor's advantage. Examining the zombie hit count compared to the survivor shots fired (Figure 8, plots 1, 3) indicates that survivors did not become sharpshooters within the number of simulated episodes. We think this could be improved slightly by simplifying the firing action, or by increasing the incentive (indirectly) to learn accuracy

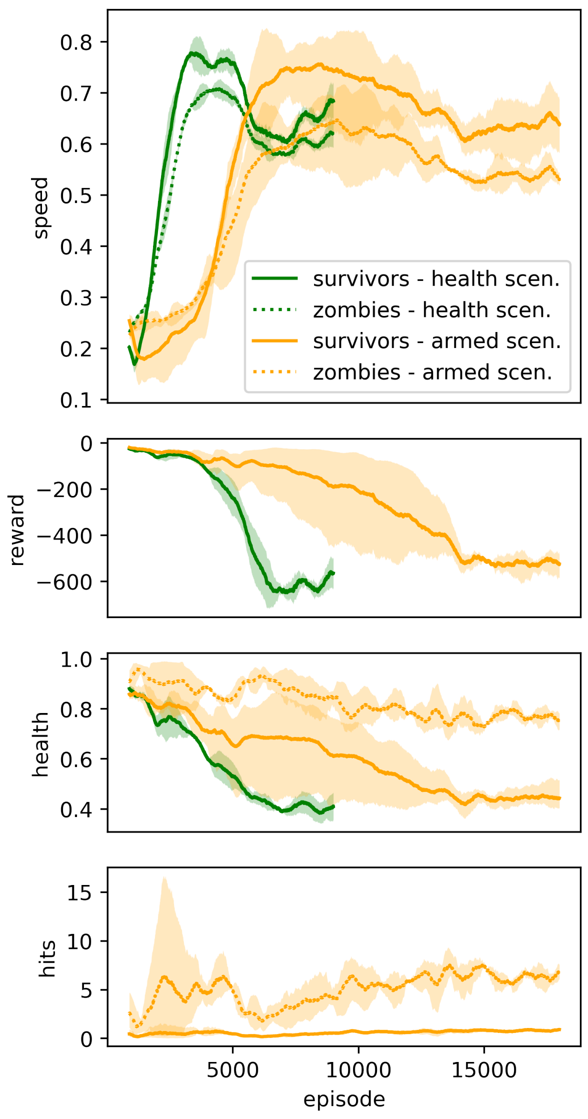

Figure 7: Comparison between *armed* and *health* scenario policy evolution over 3 training runs. Results taken as moving averages over 500 episodes prior to aggregation. Plot 1: average speed zombies and survivors per episode. Plot 2: survivor episode reward. Plot 3: episode-end health of zombies and survivors. Plot 4: total hits recieved by zombies and survivors.

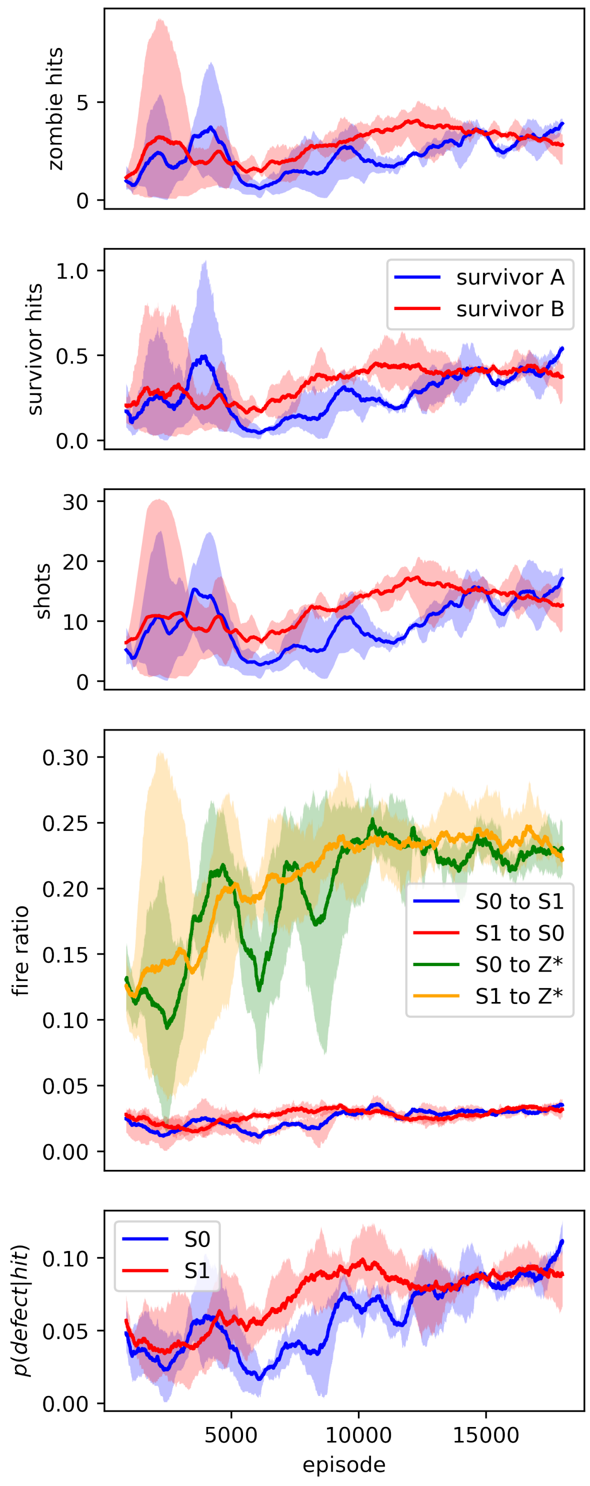

Figure 8: Comparison of hits from different survivors in the *armed* scenario over 3 training runs. Results taken as moving averages over 1000 episodes prior to aggregation due to high variance. Plot 1: zombie hit counts for each survivor. Plot 2: survivor hit counts for each survivor. Plot 3: shots fired for each survivor. Plot 4: ratio of zombie and survivor hit counts to shots fired for each survivor. Plot 5: probability a survivor shot the other survivor, conditional on them having fired and hit something, normaled for team size.

with more powerful guns and a longer reload time. Or potentially using transfer learning from a 'firing range' scenario.

Social dilemmas We observed the emergence of another competitive strategy among survivors. In some episodes we noticed a survivor would occasionally shoot another survivor. The slowdown caused to the target by the shot would sometimes cause them to become overrun with zombies, leaving the shooter free of zombie aggravation.

To establish whether this was a strategy and not just chance we modelled the conditional probability of defection (shooting another survivor) as a Bernoulli trial in which the survivor, having fired and hit something, has either hit the other survivor or a zombie.

$$p_i(defect|hit, i \in S) = \frac{\sum\limits_{j \neq i, j \in S} h_{i,j}}{\sum\limits_{j \neq i, j \in S} h_{i,j} + \sum\limits_{j \in Z} h_{i,j}} \tag{7}$$

where $h_{i,j}$ is the number of times agent i shot agent j in an episode, S is the set of survivors and Z the set of zombies.

We found that if we accept the model then survivors are learning that if they can hit an agent, it is more valuable to hit a survivor (Figure 8, plot 5). We were unable to train the model for more than 20,000 episodes (due to time constraints) to determine if defection emerges as a dominant strategy over time.

4 Discussion

In attempting to design scenarios to model a 'real-life' zombie-survival situation we encountered unexpected agent policies. We did not expect target-fixation, wall-bouncing or survivors having trouble using weapons.

When we encoded additional dynamics into the environment to make our expectations explicit we caused changes in agent policy; only some of these changes were successful. Altering an environment and modifying the state-space as we have done is one way to elicit desired policies, another approach is modifying the reward function to add detail and caveats to the agent's high-level goal. Both methods look to be blunt instruments for a real-world setting.

An implication of the challenges we saw is that if an RL agent were to be deployed in a real-world

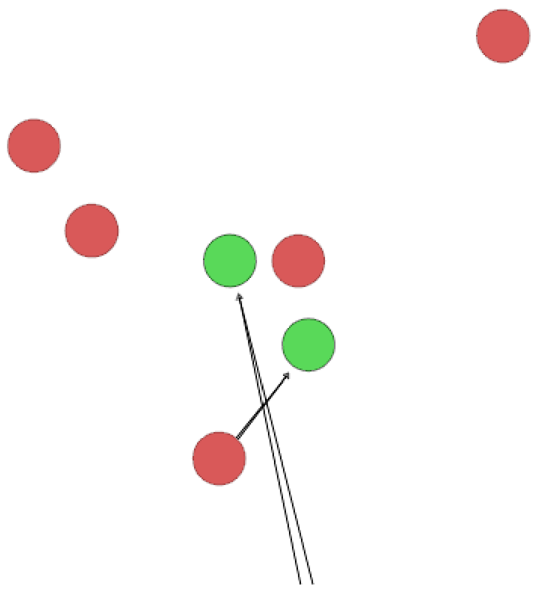

Figure 9: The *armed* environment. Survivors in green, zombies in red. Both survivors are firing weapons.

environment, the agent's behavior may only be consistent with expectations to the extent that the environment dynamics remain stationary. Beyond the known environmental dynamics, it is hard to know ahead of time (without modelling and simulation) what agent behavior would result. Additionally, policies learned are not Bayes-optimal, so an agent may not always learn a desired policy even in the presence of a well expressed reward function and a perfectly representative simulation environment.

One can make the argument that all real-world environments are multi-agent settings, so the study of multi-agent learning dynamics is especially interesting for real-world applications.

A potential avenue for further exploration of the Zombie-Survival game would be a more thorough, empirical game-theoretic analysis, such as undertaken in Leibo et al., 2017.

We found it an interesting challenge to attempt to produce desired behavior in a multi-agent setting. While we had some success, we would hesitate before deploying our agents into the real world.

References

Pieter Abbeel, Adam Coates, Morgan Quigley, and Andrew Ng. 2006. An application of reinforcement learning to aerobatic helicopter flight. *Advances in neural information processing systems*, 19:1–8.

Adrià Puigdomènech Badia, Bilal Piot, Steven Kapturowski, Pablo Sprechmann, Alex Vitvitskyi, Daniel Guo, and Charles Blundell. 2020. Agent57: Outperforming the atari human benchmark.

Gabriel Barth-Maron, Matthew W Hoffman, David Budden, Will Dabney, Dan Horgan, Dhruva Tb, Alistair Muldal, Nicolas Heess, and Timothy Lillicrap. 2018. Distributed distributional deterministic policy gradients. *arXiv preprint arXiv:1804.08617*.

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

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. Openai gym.

Li Chen, Justinas Lingys, Kai Chen, and Feng Liu. 2018. Auto: Scaling deep reinforcement learning for datacenter-scale automatic traffic optimization. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pages 191–205.

Gabriel Dulac-Arnold, Daniel Mankowitz, and Todd Hester. 2019. Challenges of real-world reinforcement learning.

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

Ashley Hill, Antonin Raffin, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, Rene Traore, Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. 2018. Stable baselines. `https://github.com/hill-a/stable-baselines`.

Matt Hoffman, Bobak Shahriari, John Aslanides, Gabriel Barth-Maron, Feryal Behbahani, Tamara Norman, Abbas Abdolmaleki, Albin Cassirer, Fan Yang, Kate Baumli, Sarah Henderson, Alex Novikov, Sergio Gómez Colmenarejo, Serkan Cabi, Caglar Gulcehre, Tom Le Paine, Andrew Cowie, Ziyu Wang, Bilal Piot, and Nando de Freitas. 2020. Acme: A research framework for distributed reinforcement learning.

Nick Jakobi, Phil Husbands, and Inman Harvey. 1995. Noise and the reality gap: The use of simulation in evolutionary robotics. In *European Conference on Artificial Life*, pages 704–720. Springer.

Joel Z. Leibo, Vinicius Zambaldi, Marc Lanctot, Janusz Marecki, and Thore Graepel. 2017. Multi-agent reinforcement learning in sequential social dilemmas. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '17, page 464–473, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems.

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.

Hongzi Mao, Malte Schwarzkopf, Shaileshh Bojja Venkatakrishnan, Zili Meng, and Mohammad Alizadeh. 2019. Learning scheduling algorithms for data processing clusters. In *Proceedings of the ACM Special Interest Group on Data Communication*, pages 270–288.

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, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. 2018. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144.

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.

Asaf Valadarsky, Michael Schapira, Dafna Shahaf, and Aviv Tamar. 2017. Learning to route. In *Proceedings of the 16th ACM workshop on hot topics in networks*, pages 185–191.

Oriol Vinyals, Igor Babuschkin, Junyoung Chung, Michael Mathieu, Max Jaderberg, Wojtek Czarnecki, Andrew Dudzik, Aja Huang, Petko Georgiev, Richard Powell, Timo Ewalds, Dan Horgan, Manuel Kroiss, Ivo Danihelka, John Agapiou, Junhyuk Oh, Valentin Dalibard, David Choi, Laurent Sifre, Yury Sulsky, Sasha Vezhnevets, James Molloy, Trevor Cai, David Budden, Tom Paine, Caglar Gulcehre, Ziyu Wang, Tobias Pfaff, Toby Pohlen, Dani Yogatama, Julia Cohen, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy Lillicrap, Chris Apps, Koray Kavukcuoglu, Demis Hassabis, and David Silver. 2019. AlphaStar: Mastering the Real-Time Strategy Game StarCraft II. `https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcra`

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.