Assisting Reinforcement Learning in Real-time Strategy Environments with SCENIC

Qiancheng Wu

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2022-129
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2022/EECS-2022-129.html

May 15, 2022

Copyright © 2022, by the author(s).
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

Acknowledgement

I’m deeply indebted to my advisor Professor Sanjit A. Seshia, for giving me the opportunity to join the Learn and Verify Group to pursue this research and for his guidance and support. I also want to express my gratitude to Professor Sergey Levine for his inspiring lecture on Reinforcement Learning and for being the second reader of this work. In addition, I am tremendously grateful to Edward Kim, who introduced me to this research project and provided continued guidance and mentorship. I also wish to thank Abdus Salam Azad for his guidance and invaluable contribution to the project. Thanks also to Debbie Liang and Michael Wu for their help on the experiments. Finally, I want to thank my family for their unwavering support and love. This work would not be possible without them.

Assisting Reinforcement Learning in Real-time Strategy Environments with SCENIC

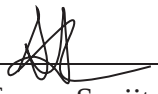
by Qiancheng Wu

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:



Professor Sanjit A. Seshia
Research Advisor

5/13/2022

(Date)

* * * * * * *

Professor Sergey Levine
Second Reader

5/13/22

(Date)

Assisting Reinforcement Learning in Real-time Strategy Environments with SCENIC

by

Qiancheng Wu

A thesis submitted in partial satisfaction of the

requirements for the degree of

Master of Science

in

Electrical Engineering and Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Abstract

Assisting Reinforcement Learning in Real-time Strategy Environments with SCENIC

by

Qiancheng Wu

Master of Science in Electrical Engineering and Computer Science

University of California, Berkeley

The success of Reinforcement Learning (RL) methods relies heavily on the diversity and quality of learning scenarios generated by the environment. However, while RL methods are applied to increasingly complex environments, the support for environment modeling is lagging behind. This work introduces the Scenic4RL interface that enables researchers to use a probabilistic scenario specification language, Scenic, to intuitively model, specify, and generate complex environments. We interface Scenic with a real-time-strategy game environment, Google Research Football (GRF), to demonstrate the benefits of adopting a formal scenario specification language to assist RL researchers in training, debugging, and evaluating RL policies. Our interface allows researchers to easily model players' initial positions and dynamic behaviors as Scenic scenarios, and with options to customize the environment's reward and termination conditions. In addition, we release a benchmark of mini-game scenarios encoded in Scenic for both the single-agent and multi-agent settings to train and test the agents' generalization abilities. Lastly, we demonstrate that researchers can use the interface to facilitate automated curriculum learning.

Contents

List of Figures

List of Tables

Acknowledgments

I'm deeply indebted to my advisor Professor Sanjit A. Seshia, for giving me the opportunity to join the Learn and Verify Group to pursue this research and for his guidance and support. I also want to express my gratitude to Professor Sergey Levine for his inspiring lecture on Reinforcement Learning and for being the second reader of this work. In addition, I am tremendously grateful to Edward Kim, who introduced me to this research project and provided continued guidance and mentorship. I also wish to thank Abdus Salam Azad for his guidance and invaluable contribution to the project. Special thanks to Kimin Lee for his insights on the work. Thanks also to Debbie Liang and Michael Wu for their help on the experiments. Finally, I want to thank my family for their unwavering support and love. This work would not be possible without them.

Chapter 1

Introduction

Reinforcement learning (RL) methods have shown great potential in solving challenging tasks in complex environments. It has demonstrated great success in games, with RL agents exceeding top human performance in games from Atari [34] to Go [27] to multi-agent real-time strategy games [36]. Meanwhile, it is used extensively in real-world settings such as traffic management [28], Internet of Things networks [4], and robot coordination [17]. These successes rely heavily on simulation environments [2, 3], publicly accessible benchmarks [5, 9, 30], and the ability to quickly model and specify the environment to facilitate development [8].

However, as the environment complexity grows, environment modeling and specification capabilities fall behind [26]. Existing real-time strategy (RTS) games, including Starcraft [32], Dota 2 [23], and soccer [16], lack the support for modeling diverse scenarios involving sophisticated interactive behaviors. These RTS games pose several challenges to effectively modeling the environment. First, these environments consist of many RL-controlled entities cooperating and competing with non-RL-controlled entities, resulting in a large state space over a long horizon. Second, stochasticity in the environment makes modeling the environment difficult. Third, effective strategies require interactions with multiple entities, so defining dynamic, sophisticated behaviors is difficult. Yet, existing simulators only support the generation of environments based on predefined layout settings and offer limited flexibility and control over the environment dynamics.

The lack of environment modeling and specification capabilities poses several challenges to RL research [6, 5, 19, 20]. First, finding large sets of diverse and realistic training data is difficult since it is infeasible to gather trajectories manually. Second, it was challenging to comprehensively evaluate the policy's generalization ability due to the lack of control over the environment dynamics in complex RTS environments.

This work addresses these challenges by introducing the Scenic for RL interface, which enables researchers to use a formal scenario specification language called Scenic [12, 13] to intuitively model and generate various realistic scenarios in a flexible, systematic, and programmatic way. Each Scenic scenario describes a distribution over the environment's initial conditions and the transition dynamics, and can be used to create a wide range of

RL environments for training and evaluation of RL policies. We demonstrate the benefits of the Scenic for RL interface on an RTS environment [1], Google Research Football (GRF), a football video game that resembles real-world soccer [16]. The code is open-source and can be found at `https://github.com/BerkeleyLearnVerify/Scenic4RL`.

1.1 Backgrounds

Reinforcement Learning

In the standard reinforcement learning setting, the RL agent interacts with the environment, and the sequential process is formulated as a Markov decision process (MDP). Formally, a MDP is represented as a tuple (S, A, T, r), where S is state space, A action space, T the transition operator capturing the environment's dynamics, $r : S \times A \to R$ the reward function. The agent's decision making procedure is characterized by a policy $\pi_\theta(a|s)$ where $a \in A, s \in S$ and θ is the parameter vector. At each time step t, the agent chooses an action a according to its policy and receive an reward r. The goal for the RL agent is to learn a policy that gives the highest rewards in expectation.

However, in practice, the agent usually does not have perfect information of the state of the environment and can only make observations to estimate the state. In this case, we extend the MDP formulation to be a Partially Observable Markov decision process (POMDP). A POMDP is defined as a tuple $(S, A, O, T, \varepsilon, r)$ where (S, A, T, r) are the same as MDP, O is the observation space and ε is the emission probability $p(o|s)$ where $o \in O, s \in S$. At each time step t, the agent observed o and choose an action a according to its policy. Its goal remains the same to maximize expected rewards.

Google Research Football

The Google Research Football (GRF) simulator [16] is a physics-based 3D simulator providing a realistic soccer environment to train and test RL agents. The game rule closely resembles those defined by Fédération Internationale de Football Association [11]. Unlike other football environments [21, 15], GRF focus on high-level actions, allowing inputs such as dribbling and shooting, rather than low-level physical control of humanoid robots. The simulator supports an arbitrary number of RL-controlled players. All players are either controlled by the RL agents or by rule-based built-in AI bots. In the single-agent setting, similar to most soccer video games, the simulator dynamically determines the RL-controlled player to be the left team player who is closest to the ball. In the multiagent setting, the controlled players are determined once at the start of the game based on the proximity to the ball. In addition, GRF comes with 11 offensive scenarios to help researchers train and evaluate RL agents and provides some pre-trained model checkpoints for a subset of the scenarios.

Scenario Specification Language: Scenic

Scenic [12, 13] is an object-oriented, probabilistic programming language with syntax and semantics specifically designed to support intuitive modeling and specification of environments. The user can encode a distribution of environment using the Scenic language in a script called a Scenic program. A Scenic program describes an abstract scenario, which includes distributions over the properties of objects, such as the initial position and its dynamic behaviors over time. To generate a concrete environment, or a scene, from a Scenic program, the Scenic server samples the defined distributions to determine the initial position of all objects in the scene to create the environment. Distributions defined inside dynamic behaviors are sampled and executed at runtime. Therefore, the user can describe a distribution of environments and model the dynamic behaviors in a Scenic script, and use Scenic to generate scenes from the scenarios.

To interface Scenic with an environment, the user needs to define the model library and the action library. The model library defines objects in the environment and their properties, such as position and heading. The action library defines the basic actions an object can take in the environment. The user can specify a default distribution over these properties in the library, so the user doesn't have to re-define every property in the scripts.

1.2 Contributions

Acknowledgement

The Scenic for RL interface described in Chapter 2 was led by Abdus Salam Azad and Edward Kim, who made significant contributions, and in collaboration with Kimin Lee, Professor Ion Stoica, Professor Pieter Abbeel, and Professor Sanjit A. Seshia [1]. Abdus Salam Azad and Edward Kim provided insightful advice and guidance throughout the work in Chapters 3 and 4. In Chapter 4, Edward Kim contributed to the implementation effort, and Michael Wu and Debbie Liang helped with the experiments.

Contribution Outline

The contribution of this work can be summarized as follows.

- In Chapter 2, we introduce our open-sourced Scenic4RL interface that enables researchers to flexibly model, specify, and generate scenarios to assist RL development. We demonstrate the benefits of adopting a scenario specification language to facilitate training and testing of RL agents in the single agent setting of Google Research Football. The interface enables researchers to endow domain knowledge in training to improve agent performance and to fine-tune the agent in various scenarios to enhance generalization.

- Chapter 3 demonstrates Scenic4RL's functionality in the multi-agent RL setting. In addition to allowing researchers to define complex, interactive behaviors for multiple entities, it allows researchers to select which entities to be controlled by the RL agent to create custom multi-agent scenarios. We show that researchers can easily endow domain knowledge to pretrain RL agents in the multi-agent setting to improve performance, and release a multi-agent mini-game scenario benchmark to assist future research.

- In Chapter 4, we demonstrate Scenic naturally facilitates unsupervised environment design framework. By modeling a Scenic program into an under-specified POMDP, we can automatically generate a curriculum of scenarios with progressively increasing difficulty that can be used to train RL agents to solve challenging tasks. We interface Scenic4RL with PAIRED algorithm [8] and evaluate its performance.

Chapter 2

Formal Scenario Modeling Language Support for RL

This chapter introduce the Scenic for RL (Scenic4RL) interface, which enables researchers to use Scenic, a Scenario Specification Language, to intuitively model, specify, and generate RL environments. Researchers can specify the desired environment setup, which can be either a concrete scene or a distribution of scenarios, using Scenic language. The interface will sample environments according to the distribution defined in the Scenic program and generate an OpenAI-Gym environment to be interfaced with the RL framework.

2.1 Problem Definition

Formally, a concrete environment, or a concrete scenario, can be represented as a Markov Decision Process (MDPs) [29] defined as a tuple $(\mathcal{S}, \mathcal{A}, p, r, \rho_0)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ the action space, $p\left(s'|s,a\right)$ the transition dynamic, $r\left(s,a\right)$ the reward function, and ρ_0 the initial state distribution. Researchers can model a distribution of environment as a Scenic scenario. Scenic allows the user to model the environment's (i) the initial state distribution, (ii) the transition dynamics (specifically players' behaviors), and (iii) the reward function.

Modeling Initial State Distribution

Users can easily specify initial state distributions of the players and the ball with Scenic's intuitive syntax that resembles natural English. For example, the user can easily define a region on the field to be the player's spawning region. In the 3v3 Cross from side example 2.1, the get_reg_from_edges function defines a rectangular area from which Scenic can sample points to be the spawning position (see line 32, 34, 36). Scenic also supports more than 20 different syntaxes to specify the advanced spatial relationships, such as positioning the ball in front of a player's heading (line 37). Therefore, the user can quickly define a complex distribution over the spawning positions for all players using Scenic. Scenic also provides

```
param game_duration = 400
param deterministic = False
param offsides = False
param end_episode_on_score = True
param end_episode_on_out_of_play = True

leftLeftBackRegion = get_reg_from_edges(-70, -60, 20, 15)
leftCenterBackRegion = get_reg_from_edges(-70, -65, 10, -10)
leftRightMidRegion = get_reg_from_edges(-70, -65, -10, -20)

rightRightMidRegion  = get_reg_from_edges(-55, -50, 20, 15)
rightCenterMidRegion = get_reg_from_edges(-65, -60, 0, 5)
rightLeftMidRegion   = get_reg_from_edges(-55, -50, -30, -35)

rightRM_AttackRegion = get_reg_from_edges(-80, -70, 5, -5)
rightAM_AttackRegion = get_reg_from_edges(-90, -85, -5, -10)
rightLM_AttackRegion = get_reg_from_edges(-80, -75, -25, -30)

behavior runToReceiveCrossAndShoot(destinationPoint):
  do MoveToPosition(destinationPoint)
  do HoldPosition() until self.owns_ball
  do dribbleToAndShoot(-80 @ 0)
  do HoldPosition()

behavior rightLMBehavior(destinationPoint):
  do MoveToPosition(destinationPoint)
  do HighPassTo(Uniform(ego, right_RightMid))
  do HoldPosition()

RightGK
right_RightMid = RightRM on rightRightMidRegion,
  with behavior runToReceiveCrossAndShoot(Point on rightRM_AttackRegion)
ego = RightAM on rightCenterMidRegion,
  with behavior runToReceiveCrossAndShoot(Point on rightAM_AttackRegion)
right_LeftMid = RightLM on rightLeftMidRegion,
  with behavior rightLMBehavior(Point on rightLM_AttackRegion)
ball = Ball ahead of right_LeftMid by 2

LeftGK with behavior HoldPosition()
leftLB = LeftLB on leftLeftBackRegion
leftCB = LeftCB on leftCenterBackRegion
leftRB = LeftRM on leftRightMidRegion
```

Figure 2.1: The Scenic Program defining the 3v3 Cross From Side Scenario.

Figure 2.2: An initial scene of the 3v3 Cross From Side Scenario

default spawning regions based on player roles. The role is abbreviated as two letters in the player's name. For example, in line 30, despite the user not specifying its spawning position, the opponent goalkeeper will be spawned in the opponent goal region.

Modeling Transition Dynamics of the Environments

Scenic allows the user to flexibly control the transition dynamics of the environment by modeling the dynamic behaviors of all players. Each player in Scenic can be assigned a behavior that models its actions throughout the episode. Scenic behaviors are hierarchical, so each behavior can invoke other behaviors. Scenic4RL provided an extensive library of pre-defined behaviors and actions to allow the user to quickly model desired interactive behaviors. For example, line 19 models a behavior that invokes four pre-defined behaviors, and this behavior is assigned to players defined in lines 31 and 33. Users can also model conditional behaviors that invoke other behaviors and actions based on the defined conditions using if-else statements.

Modeling Rewards

Users can customize environment rewards directly in the Scenic program. User can implement the logic to modify scenic rewards in the `monitor` construct. At every simulation step,

```
1  monitor CustomRewardMonitor:
2      sim = simulation()
3      while True:
4          ball_x = sim.game_ds.ball.position.x
5          ball_y = sim.game_ds.ball.position.y
6          if ball_y > 70 and abs(ball_y) < 50:
7              sim.scenic_reward = 0.1
8          else:
9              sim.scenic_reward = 0
10
11         wait
```

Figure 2.3: An example monitor that gives additional rewards based on the ball's position.

```
1  monitor CustomTerminationMonitor:
2      sim = simulation()
3      while True:
4          ball_x = sim.game_ds.ball.position.x
5          if ball_x > 0:
6              terminate
7
8          wait
```

Figure 2.4: An example monitor that terminates the scene if the ball is in the right half of the field.

the monitor is executed and the scenic_reward parameter is added to the environment reward. Example 2.3 defines a monitor that gives a small additional reward if the ball is close to the opponent goal at each timestep. The wait syntax in line 11 tells the Scenic server to halt executing the monitor until the next timestep.

Custom Termination Conditions

There are two ways to specify termination conditions in a Scenic program. First, user can enable GRF built-in termination conditions by setting the relevant environment parameters to true using the `param` keyword. For example, in line 4 of Scenic program 2.1, we set the GRF parameter to end each episode on scoring. Second, users can specify custom termination conditions in the `monitor` construct, which are checked at every simulation time step, to end the episode when conditions are met. Figure 2.4 shows an example monitor that terminates

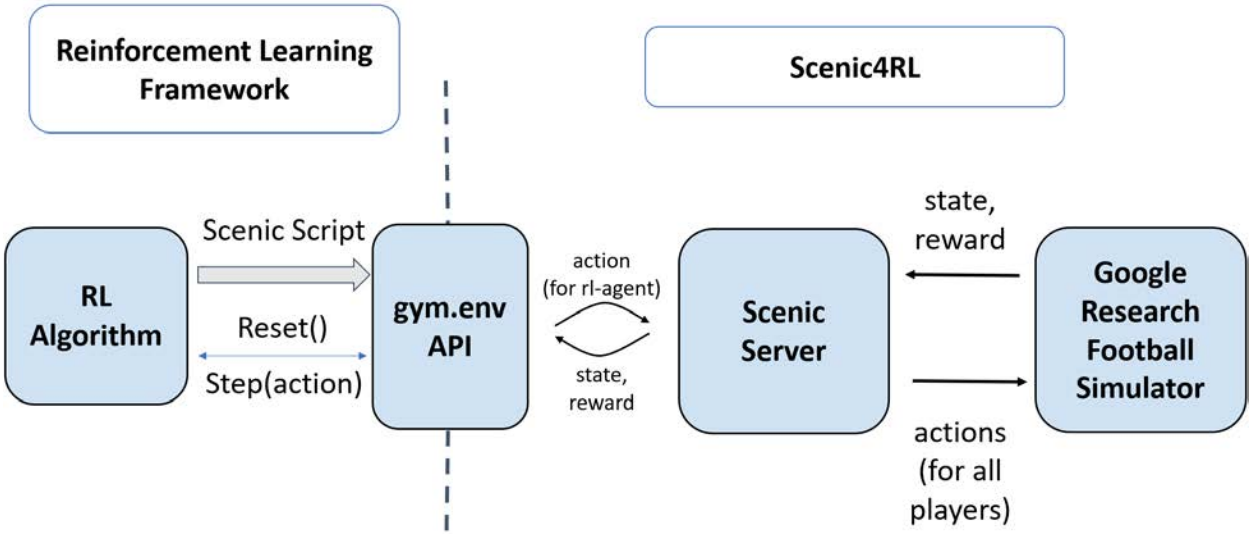


Figure 2.5: Overview of the Scenic4RL interface architecture.

an episode if the ball is in the right half of the field.

2.2 Approach

Scenic4RL Interface Architecture

The scenic4RL interface consists of two main components: the RL interface and the Scenic Server. The RL interface generates a Gym environment from the Scenic simulation and bridges the internal Scenic server and the RL framework. The Scenic server executes the Scenic program and maintains the simulation objects by communicating with the underlying simulator. The interface creates concrete environments from the given Scenic scenario and exposes a Gym RL interface [3] so that researchers can seamlessly use Scenic4RL with RL frameworks. Figure 2.5 shows the overview of the architecture.

When a new simulation is created from a Scenic Program, the Scenic server parses the program and maintains a list of all samplable variables, such as the players' initial positions if they are defined as a distribution over a region. When reset() is called, the server randomly samples all samplable variables to generate a scene and establish communications with the underlying GRF environment to start simulating the scene. Based on the initial state, it updates its internal world models that include the positions of all players and the ball. It is worth noting that the Scenic server controls the actions of all players, regardless of the number of players controlled by the RL agent. At every timestep, the Scenic server communicates with the underlying GRF environment to receive the next state and observations, while the RL interface receives input from the RL framework about the RL agent's action. Then the

Scenic server calculates the actions of all players based on RL agent inputs and the player behaviors defined in the Scenic program. It then sends the actions to the underlying GRF environment to receive the next state and observations and update its internal world models. Finally, it executes the monitors to determine custom rewards and check termination conditions before forwarding the updated observations to the RL framework. The process repeats until the environment is terminated or the reset() function is called, in which case the server will start to create a new scene.

Interfacing Scenic to a new RL environment

To use Scenic to model and generate an RL environment such as an RTS game like Google Research Football environment, we must first interface Scenic to the simulator. It is relatively straightforward to interface Scenic with a new simulator since Scenic has already been interfaced with other simulators [7] in fields such as autonomous driving, aviation, and robotics. We defined a model library, an action library, and a behavior library according to the environment specifications. Therefore, the user can easily reuse these libraries to build complex scenarios and model dynamic player behaviors.

In the *model* library, we define the template for all the objects in the environment. For example, in GRF, the objects include players and the ball. For players, we define a default spawning area based on the player's roles and assign their default behavior to use the built-in AI bot. We also define region objects such as the goal area and the penalty box areas and directional objects representing the compass directions for convenience.

The *action* library defines the action space of the RL environment. These actions are used in behaviors that model the environment dynamics. In GRF, we included 19 actions, including an idle action, movement actions in eight compass directions including reset direction, three types of passing, a shooting action, a sliding action, and actions to start and stop dribbling and sprinting. Although GRF includes a built-in AI Bot action that lets the built-in AI control the player, we did not allow RL agents to take this action.

The *behavior* library defines the basic behaviors that can be used as the building block for more complex behaviors and the helper functions. In GRF, it consists of widely used basic soccer skills such as passing to the nearest teammate, evasive zigzag dribbling to avoid the opponent from intercepting the ball, and aiming at the corner to shoot. In addition, we defined several helper functions that can be used to create more complex behaviors. These include finding the closest teammates, finding the player closest to the ball, and whether there is an opponent near a teammate's running direction.

2.3 Evaluation

This section demonstrates some use cases of Scenic4RL to train, debug, and evaluate RL agents in the GRF single-agent settings.

Experimental Setup

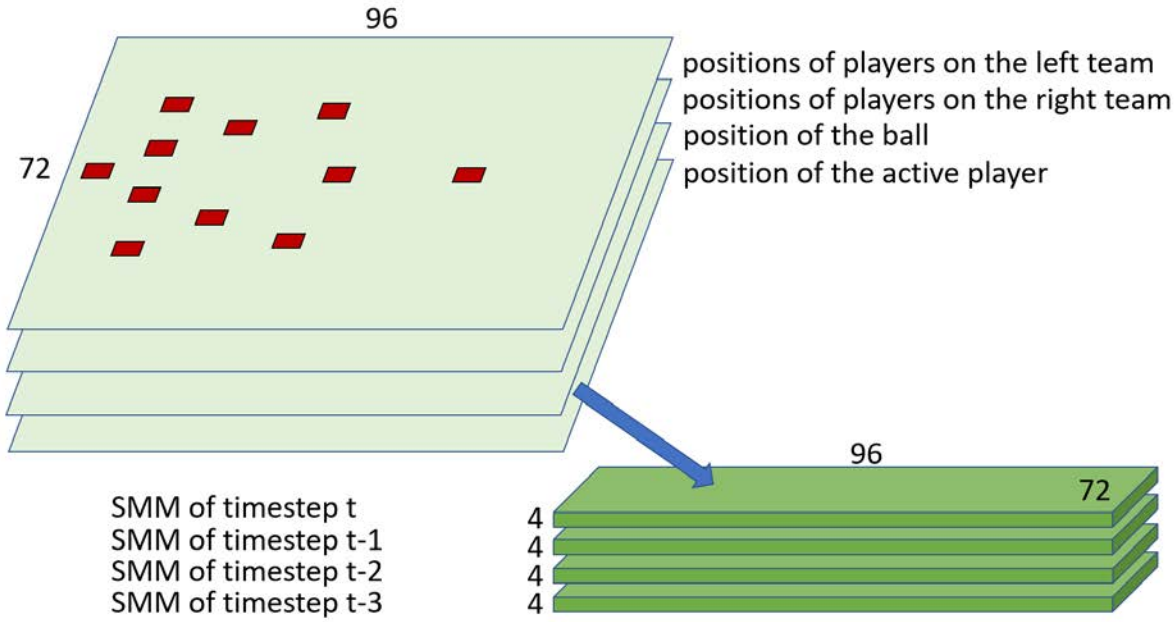


Figure 2.6: Figure illustrating the stacked SMM representation of the observation used in the experiment. Each SMM representation is a 72*96*4 bitmap representing the position of players and the ball. The observation is a concatenation of the SMM representations of the previous four timesteps. The red color in the figure denotes the value of 255, and green denotes the value of 0.

We run the experiments on the single-agent setting of GRF, where the RL agent always controls the left player who is closest to the ball. We train the RL policy using the PPO [25] algorithm for 5M timesteps on a single GPU machine (NVIDIA T4) with 16 parallel workers. Unless otherwise specified, all experiments are repeated ten times using random seeds. Finally, we evaluate the trained policy on the evaluation scenarios for 10k timesteps. The GRF environment setup is as follows. We use the stacked Super Minimap (SMM) representation of the observation, where the observation at timestep t is the concatenation of the SMM representation from time $t, t-1, t-2, t-3$. Each SMM observation is a bit map of dimension 72*96*4, where each channel represents the location of left players, right players, the control player, and the ball, respectively. Figure 2.6 illustrates the observation format. The agent receives a reward of 1 if the left (RL-controlled) team scores and -1 if the opposing team scores. We set GRF termination conditions as those in provided GRF scenarios [16], ending if either team scores, the ball changes team possession, or the ball goes out of the field.

Minigame Benchmark

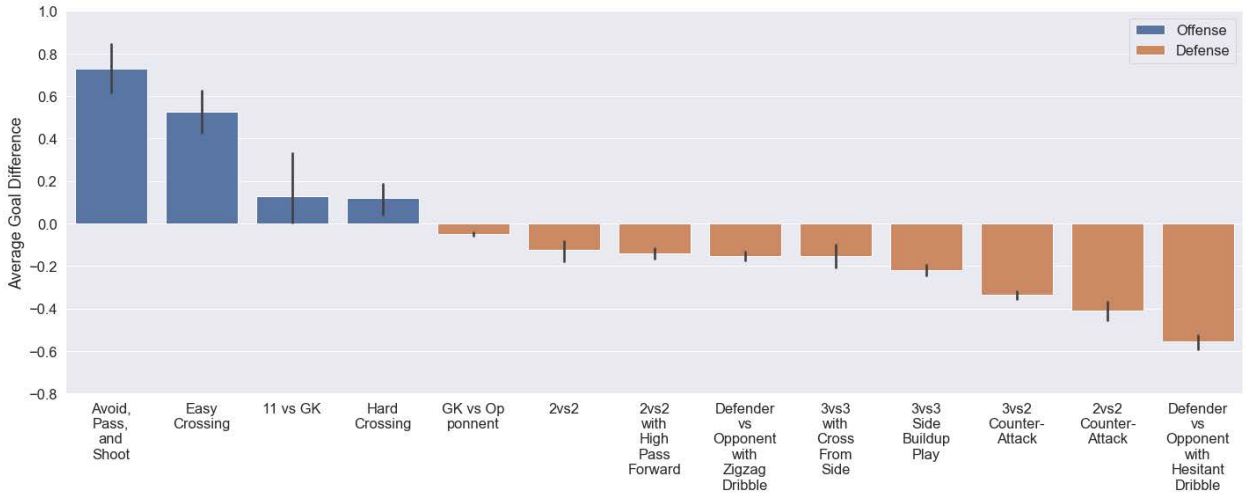


Figure 2.7: Average Goal Difference of PPO agents on the mini-game scenarios. Blue bars represents offense scenarios and yellow bars represents defense scenarios. The error bar shows 95% confidence intervals.

It is challenging to train an RL agent to play the 11vs11 full game. Previous study [16] has shown that training an agent to solve the complete game can take up to 50M timesteps. To enable RL researchers to develop and test models quickly, we released a set of 13 mini-game scenarios inspired by everyday situations in real-world soccer games. These mini-game involves few players, but they still require the policy to master combinations of soccer skills to solve. The scenario's simple environment configuration reduces the training time to speed up RL development.

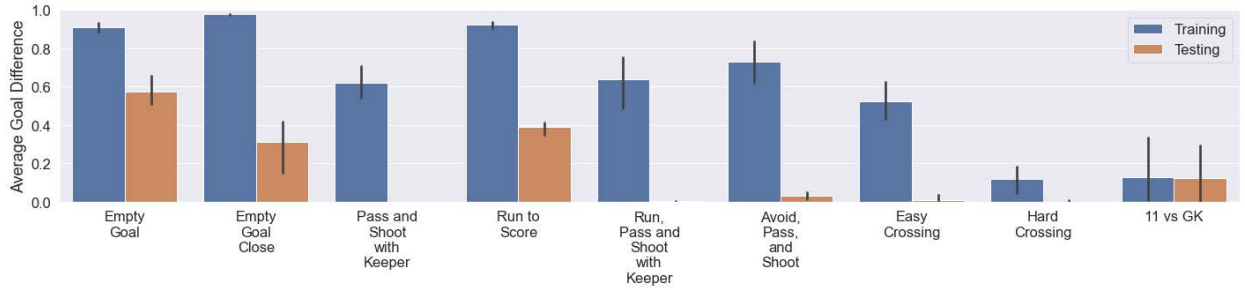
The mini-game scenarios include four offense and nine defense scenarios. The offense scenarios require the RL-controlled player to create an attack opportunity to score. In defense scenarios, the opponent players spawn in the left half of the field with the ball, ready to attack and score, and the RL-controlled player must intercept the ball to prevent the opponent from scoring. We modeled the behavior of the opponent players to ensure they were capable of launching an effective attack. In fact, example 2.1 is adopted from one of the defensive scenarios, 3v3 cross from the side.

We benchmark our mini-game scenarios using the PPO algorithm [25]. Figure 2.7 shows the average goal difference of the RL agent in these scenarios. For offense scenarios, an ideal agent should achieve the goal difference of 1, which indicates it scores every time. Similarly, a goal difference of -1 is optimal for defense scenarios as it implies the RL agent can stop the attack every time. From the graph, we can see the scenarios vary in difficulty. For example, PPO agents scores consistently high on the avoid, pass and shoot offense scenarios but struggle to solve the hard crossing scenario. The provided defense scenarios have a range

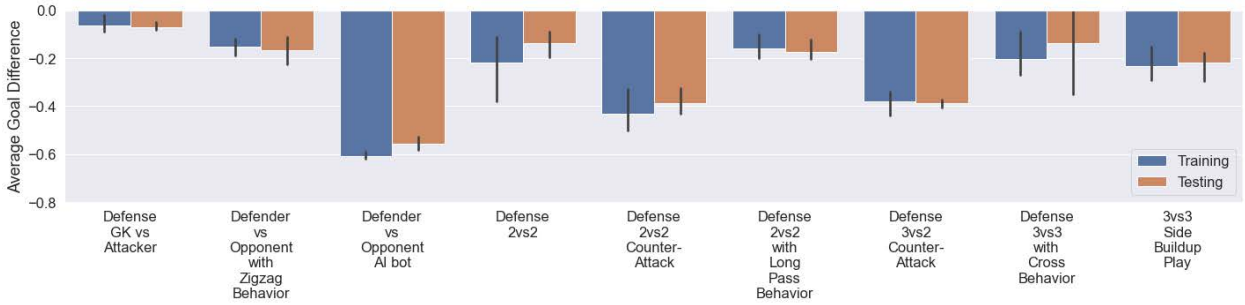
of difficulties, with GK vs. Opponent being the easiest and the defender vs. an opponent with hesitant dribble the most challenging.

It is worth noting that because of the stochasticity of the GRF environment, we observe some significant variations in training performance between runs in some scenarios. For example, in the 11 vs. GK scenario, one experiment achieves a 0.93 average goal difference while another failed to learn anything.

Testing for Generalization



(a) Offense and select GRF scenarios



(b) Defense scenarios

Figure 2.8: Evaluation of the generalization ability of PPO agents in varying initial conditions. The blue bar represents the agent performance on the training scenarios, and the yellow bar represents the performance on the test scenarios.

We test the generalization ability of the PPO agents by evaluating them on unseen scenarios with different initial positions. We create evaluation environments for all the mini-game scenarios and selected GRF scenarios by changing the distribution over the initial player position while keeping the formation and behaviors of the players the same. For example, figure 2.9 shows the bird-eye view of the pass and shoot scenario. In this case, the testing scenario is a mirrored image of the training scenario.

Fig 2.8 compares the performance of trained PPO agents in the training and testing scenarios. We observe a large drop in performance in the testing scenarios of offense scenarios,

(a) Training Scenario (b) Testing Scenario

Figure 2.9: Bird-eye view of the training and testing scenario for Pass and Shoot.

but the performance difference between training and testing scenarios in the defense scenarios is less significant. One explanation is that defensive scenarios generally contain larger distributions over the initial state, and the larger variation in scenes contributes to better generalization.

Facilitating Training with Probabilistic Scenic Policies

Scenic4RL enables researchers to model player behaviors to incorporate domain knowledge into training. Researchers can encode expert strategies as probabilistic Scenic behaviors in a scenario and generate demonstration data to pre-train the RL agent to reduce training time. In this section, we define semi-expert Scenic policies for five scenarios where the PPO agent struggles to learn. For each scenario, we generate 8k episodes of offline demonstration data and run the behavior cloning algorithm on the data for 2M timesteps. We then fine-tune the agent using PPO for an additional 5M timesteps. Figure 2.10 shows the performance of pre-trained agents and that of PPO agents without pretraining. It also includes the performance of our semi-expert policy and the performance of the intermediary behavior cloning (BC) agent.

The result demonstrates the effectiveness of imitation learning from a relatively low amount of demonstration data in the GRF single-agent setting, as the performance of the BC agent generally matches that of the provided Scenic policy. In addition, we find a significant improvement in agent performance after pretraining and fine-tuning compared to agents trained using PPO directly. In all five scenarios, the fine-tuned PPO agent overperforms the vanilla PPO agent and the provided semi-expert Scenic policy. Therefore, researchers

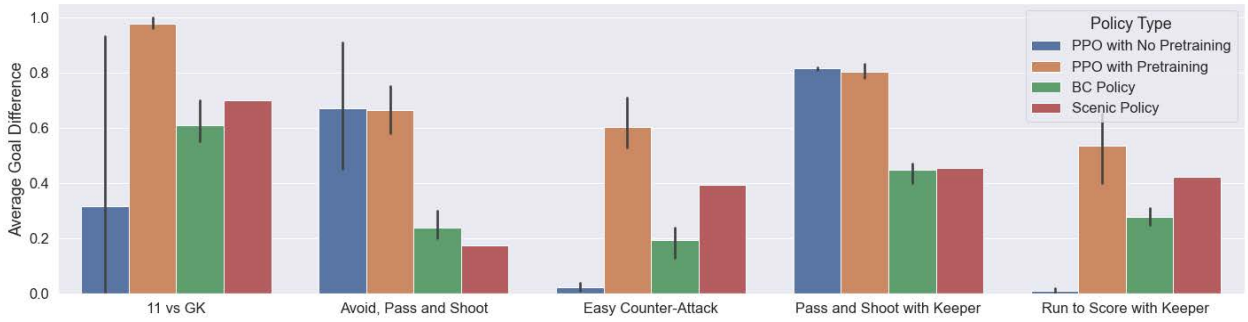


Figure 2.10: Performance of PPO agents with and without pretraining with demonstration data from semi-expert Scenic policies, and the performance of the behavior cloning agent and the semi-expert Scenic policies.

can write stochastic Scenic policies to pretrain the agents to improve training performance, especially in challenging scenarios where the PPO method alone is not sufficient to train an RL policy.

Chapter 3

Scenic for RL in Multi-Agent Environments

Scenic enables intuitive modeling, specification, and generation of RL environments. In Chapter 2, we focus on the single-agent environment, in which the RL agent controls one player and interacts with bot-controlled teammates and opponents. While GRF's rule-based built-in bots can perform simple maneuvers, they do not coordinate with the controlled player or other bots. Since the RL agent has no control over other players in the single-agent environment, bot teammates will not move to correct positions or support the player to facilitate an ongoing attack, and opponent bots often fail to organize effective defenses. Fortunately, Scenic supports seamless generation of multi-agent environments from existing Scenic programs. Researchers can easily specify players to be controlled by RL agents, and the Scenic4RL interface would create the corresponding multi-agent RL environment. Researchers can also define interactive, dynamic behaviors for all players using Scenic's behavior and action library. It allows researchers to develop diverse training scenarios, and to endow human knowledge to help pretraining the RL agents to solve challenging tasks.

3.1 Problem Definition

In the multi-agent setting, we extend the POMDP formulation of reinforcement learning as a decentralized partially observable Markov decision process (Dec-POMDP) [22]. Formally, a Dec-POMDP can be described as a tuple $(I, S, A, O, T, \epsilon, r)$ where $I \equiv [1, ..., n]$ is the set of n agents, S the state space, A the joint action space, O the joint observation space, T the transition operator, $\epsilon = p(o|s, a)$ the emission probability, the probability of seeing observation o given the state is s and the action taken is a, and r the global reward function. At each time t, each agent i receives its local observation o_t^i from the joint observation $o_t \in O$ and choose an action a_t^i according to the RL policy. All agents' actions forms a joint action $a_t \in A$ and all agents receive the same reward r_t based on the reward function r.

3.2 Approach

Scenic4RL uses a control mode parameter to determine the type of generated RL environment. The control mode determines how RL agents should control the players defined in the Scenic program. By default, if the control mode is not set, it will generate a single-agent gym environment where the RL agent controls the left player closest to the ball at the moment. The environment expects one action at each timestep and returns the observation specific to the controlled player.

If the control mode is set to a multi-agent option, it will first determine players whose control be exposed to the RL agents, and create a multi-agent gym environment with modified observation space and action space to account for all controlled players. The environment expects a list of actions, one for each player at each timestep, and returns a list of observations, each corresponding to a currently-controlled player.

Calculating Player Actions

Recall the Scenic4RL interface consists of two components. The RL interface bridges the RL framework and the Scenic server, receiving input actions from the RL framework and outputting the observations and rewards. The Scenic server parses the Scenic program, communicates with the underlying simulator, and maintains the simulation state, including the position of all players and the ball.

At each time step, the Scenic server calculates and forwards all players' actions, including those for the opponent players, to the underlying simulator regardless of the environment type (single-agent or multi-agent environment). To properly apply the input action from the RL agent via step(), it maintains a list of players in the Scenic internal states who are to be controlled by the RL agent at the moment, and the list is recalculated at every step to ensure the RL agents always control the desired players. It then applies the input actions to the corresponding internal Scenic players. For the rest of the players not controlled by the RL agent, Scenic will compute their actions if that player has an active Scenic behavior; otherwise, it assigns the built-in bot action.

In the multi-agent setting, when reset() is called, the episode will start from a new sampled scene, but the control mode, observation space, and action space format will remain the same.

Please note that the Scenic4RL interface creates and maintains the gym multi-agent environment. Still, there are many flexibilities for the RL framework regarding using the environment, e.g., how to unpack the observation and assign agent-policy mapping.

Control Modes

To create a multi-agent RL environment from a Scenic program, the user needs to 1) specify the number of controlled agents and 2) decide how to map these controlled agents to players. The number of controlled agents determines the size of the action and observation spaces. It

can range from 2 to the total number of players in the left (RL controlled) team. Scenic4RL provides two methods to map controlled agents to players: Fixed and Dynamic. In the fixed method, the mapping of controlled agents to players is determined at the start of the episode and remains constant throughout the episode. In the dynamic method, the mapping is recalculated at every time step based on the player's proximity to the ball. As a result, the first controlled agent (index=0) may be mapped to player A at the start and be mapped to player B in another timestep because the distances between the ball and the players change.

The combination of both settings is summarized as the control mode, which the user specifies at the creation of the Scenic4RL environment. Scenic4RL currently supports the following control modes.

Fixed Mapping

All The RL agents control all left players. The number of controlled players is automatically set to the total number of left players. This mode is useful in defensive mini-game scenarios where we want RL agents to learn to coordinate defenses.

AllNonGK The RL agents control all left players except the goalkeeper. The goalkeeper will be controlled by the built-in AI bot. The number of controlled players is set to the total number of left players - 1. This mode is mostly used in offensive scenarios where the goalkeeper is unlikely to participate in attacks.

Variable The user can enter the number of controlled players as an integer as the control mode. In this case, the controlled player mapping is determined at the start of the episode based on the proximity to the ball. For example, if the control mode is set to 3 in the 11 versus goalkeeper scenario, the controlled agent at index 0 will be mapped to the left player closest to the ball at the start of the game, index 1 to the second closest player, and index 2 to the third closest player. If the number exceeds the total number of left players, the simulator will raise an exception. This setting is equivalent to the vanilla GRF multi-agent environment.

Dynamic Mapping

closest1 This is the default control mode and produces a single-agent environment. The controlled agent is mapped to the player closest to the ball at any time in the episode. This mode reflects the vanilla GRF single-agent environment.

closest2 The RL agents always control the two closest players to the ball at any moment in the episode. This is similar to the variable mode, but the mapping is updated at every step.

closest3 Similar to closest2, but the RL agents control the three closest players to the ball at any moment in the episode.

Multi-agent Observation and Action Spaces

In the multi-agent setting, the observation and action spaces are different from those in the single-agent settings because we have to distinguish individual observations and actions of each controlled agent. Let n denotes the number of controlled players. At each timestep, all controlled agents receive their individual observations specific to its mapped players, and the environment expects a joint action of length n consisting of all controlled-agents/players' actions. Formally, the observation space $O = \times_i O_i$ is the set of joint observations where O_i is the set of observations for agent i. The action space $A = \times_i A_i$ is the set of joint actions where A_i is the set of actions for agent i.

In the implementation, the observation is list of n individual observations, each corresponding to a controlled agent's observation. The action is a list of n actions, each corresponding to an agent's action.

3.3 Evaluation

Experimental Setup

For all the experiments, we use GRF's stacked super minimap (Stacked SMM) representation of the observation. Each controlled agent only sees the observation specific to that player. Each observation at time t is a concatenation of the SMMs of time t, $t-1$, $t-2$, $t-3$, where each SMM is a 4x72x96 bit matrix representing the position of left team players, right team players, controlled player and the ball. RL policies are not allowed to use the default built-in bot action. All agents receive a reward of $+1$ if the left team scores, and a -1 if the right (opponent) team scores. An episode ends if either team scores, or the ball is out of the fields, or the ball possession changes.

We use RLlib library to train and evaluate RL policies. We train a separate policy for each controlled agent in the multi-agent RL environment. Unless otherwise specified, we train each policy using PPO algorithm for 5M in-game timesteps with 16 parallel workers on a single GPU machine (Nvidia T4). Each experiment is repeated for 5 random seeds.

Multiagent Minigame Benchmark

Multi-agent RL is challenging. Recent studies show that conventional RL training methods are insufficient to train policies to solve the 11v11 full game [14]. In fact, in our testing, PPO agents struggle to learn to solve even moderately challenging scenarios involving fewer players. To give researchers a benchmark for evaluating RL methods, we create and benchmark a set of multi-agent mini-game scenarios consisting of 13 scenarios based on single-agent scenarios. It includes a mix of offensive and defensive scenarios involving 3 - 11 controllable players. Each scenario represents a situation common in the actual soccer game that requires coordination between players to solve.

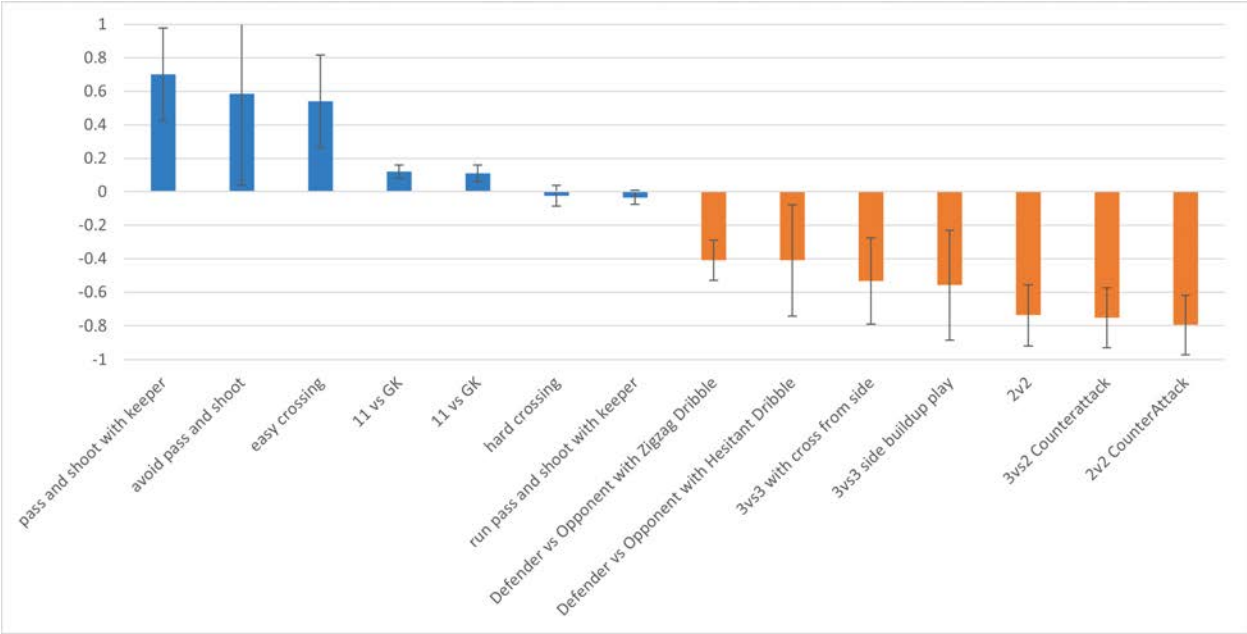


Figure 3.1: Bar chart showing the average goal difference of PPO agents on the multi-agent mini-game benchmark. The error bar represents 95% confidence interval. Blue bars denote offensive scenarios and orange bars denote defensive scenarios.

The user can control the number of RL-controlled players using the control mode setting. Although, in theory, increasing the number of RL-controlled players allows better coordination between players to achieve better performance, it significantly increases the environment complexity and makes training more difficult. Therefore, to allow for rapid development and testing, most scenarios comprise no more than four active players to represent the situation without creating additional challenges.

We propose defensive scenarios where the opponent player spawns with the ball close to the RL Team's goal. The scenario scripts define the opponents' behaviors to ensure their attack is effective. The objective of RL policies is to utilize all defense players and the goalkeeper to defend the aggression and prevent the opponent from scoring. Therefore, we recommend using the "all" control mode for all defensive scenarios to enable RL agents to organize defense with all available players.

The set also includes offensive scenarios where the RL players must maintain ball possession and score. Most scenarios require team coordination to bypass opponent defenders. Therefore, we recommend using the "allNonGK" control mode to maximize teamwork potential.

We benchmark the proposed mini-game scenarios by training RL agents with PPO for 5M timesteps and repeated for 5 random seeds. We use the "all" and "allNonGK" control modes for defensive and offensive scenarios, respectively, except for 11vsGK scenario where we

repeat experiments with 2 and 3 controlled players with fixed mapping. Figure 3.1 compares the mean agent performance on benchmark scenarios. Table 3.1 details the scenario settings and the average goal difference of the RL team. Because we end episodes if either team scores, the goal difference is between -1 and 1. For offensive scenarios, an optimal policy should achieve a goal difference of 1 (consistently scores), and for defensive scenarios, it should be 0 (always prevent the opponent from scoring).

From the graph, we can see the scenarios in the benchmark have varying difficulties. While RL agents can achieve near-perfect performance on the pass and shoot with keeper scenario, they struggle to make progress in scenarios such as hard crossing and run, pass and shoot with keeper. Defensive scenarios also pose challenges to the agent in the multi-agent setting. The RL agents can only reduce the goal difference to around -0.4 on moderately-difficult 2v1 scenarios such as Defender vs. Opponent with Zigzag Dribble and Defender vs. Opponent with Hesitant Dribble.

Pretraining with Domain Knowledge

While it is challenging to train RL policies from the stretch in the multi-agent environment with many controlled players, researchers[14] have used offline RL training methods such as imitation learning to train RL agents to solve a simpler version of the 11v11 full game[1]. However, the offline RL algorithm generally requires a large dataset of trajectories of successful runs. These trajectories are hard to get because they require a working model capable of playing the game in multi-agent settings, which the RL community currently lacks. Alternatively, researchers have to rely on human experts to manually define multiple controlled players' trajectories, which is often impractical and tedious.

Researchers can use Scenic4RL to easily generate offline training data from Scenic programs. Similar to the single-agent environment, Scenic allows the researcher to directly encode expert strategies for all controlled players using Scenic's extensive behavior and action library. In the experiments, we define distributions of behaviors instead of deterministic actions to ensure the diversity of the generated datasets. To facilitate development, Scenic4RL provides a utility script to automatically generate trajectories from Scenic programs and record them in an RLlib-compatible file to be used for future training.

Therefore, Scenic4RL enables researchers to endow domain knowledge to facilitate training. Before training the multi-agent policies using PPO, researchers can pretrain the policies on the generated demonstration data through behavior cloning. To demonstrate, we composed a semi-expert policy in Scenic for each of the four multi-agent scenarios and used it to generate 8k sample trajectories. Then, we run the behavior cloning algorithm on the generated dataset for 2M timesteps and then further fine-tune the policies using PPO for additional 5M timesteps. Figure 3.2 compares the performance of the PPO agents with pretraining with that of agents without pretraining. Table 3.2 details the average goal dif-

[1]In this setting, RL policies are allowed to take the built-in AI action. However, we don't allow the policy to take this action in our experiments.

Scenario	Number of Left Player	Number of Controlled Player	Avg Goal Diff	95% C.I.
Offensive Scenarios				
pass and shoot with keeper	3	2	0.7013	0.2773
avoid pass and shoot	3	2	0.5856	0.5482
easy crossing	3	2	0.54029	0.2772
11 vs GK (2)	11	2	0.12	0.0399
11 vs GK (3)	11	3	0.11	0.0498
hard crossing	4	3	-0.024	0.0613
run pass and shoot with keeper	3	2	-0.034	0.0423
3v1	4	3	0.591	0.592
Defensive Scenarios				
Defender vs Opponent with Zigzag Dribble	2	2	-0.41	0.1199
Defender vs Opponent with Hesitant Dribble	2	2	-0.41	0.333
3vs3 with cross from side	4	4	-0.534	0.2572
3vs3 side buildup play	4	4	-0.5573	0.3267
2v2	3	3	-0.7372	0.1801
3vs2 Counterattack	4	4	-0.7517	0.1769
2v2 CounterAttack	3	3	-0.794	0.1749

Table 3.1: Table of PPO agent performances on Multi-agent Mini-game Benchmark.

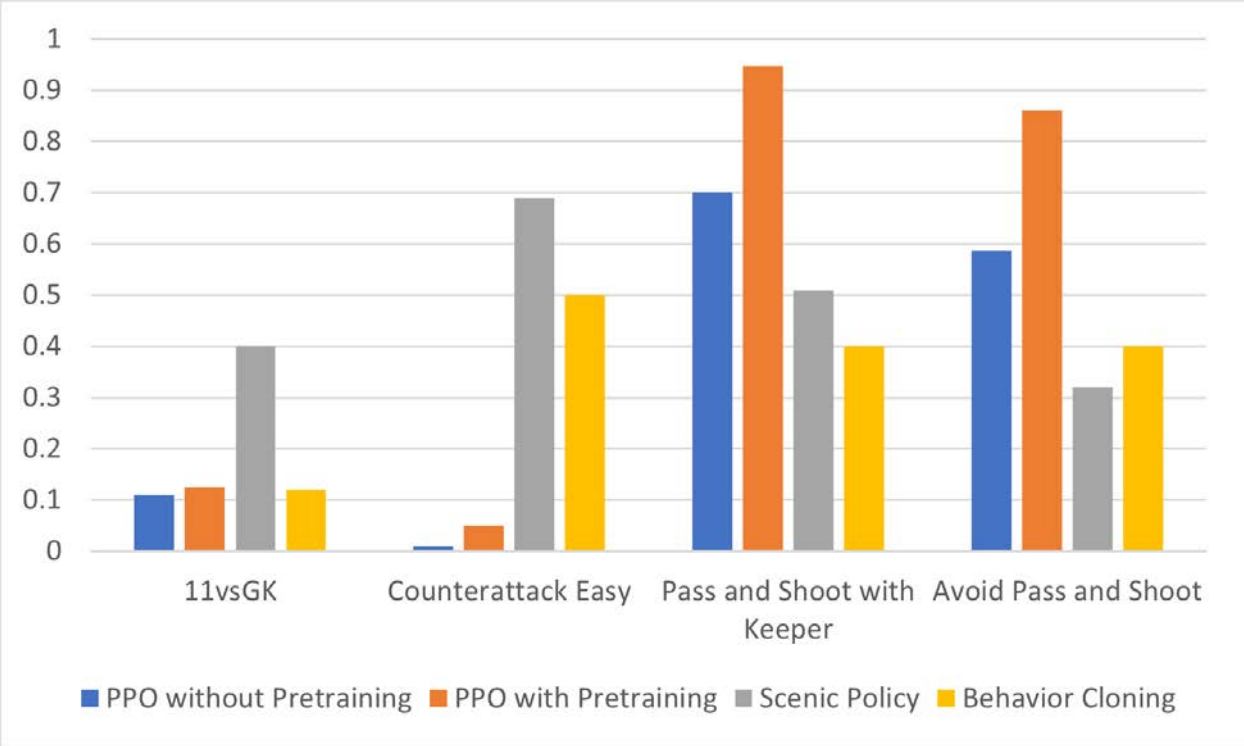


Figure 3.2: Bar chart comparing the performance of agents trained with and without demonstration data. The first two columns represent the average goal difference of RL agents without pretraining and with pretraining. The third and fourth columns show the performance of the semi-expert Scenic policy and the behavior-cloned agents. The first two scenarios, 11vsGK and Counterattack Easy, had environment-controlled teammates, while the last two scenarios, Pass and Shoot with Keeper and Avoid Pass and Shoot, had all relevant players controlled by RL agents.

ference of PPO agents with and without pretraining, the semi-expect Scenic Policy, and the intermediate behavior cloning agent.

The experiment results show that behavior cloning agents were relatively successful in fitting the semi-expert behavior in most scenarios. However, the effectiveness of further PPO fine-tuning depends on whether there were uncontrolled teammates in the scenario. If the RL agent controls all the relevant players in the scenario, e.g., Pass and Shoot with Keeper and Avoid Pass and Shoot, then we see a significant improvement in agent performance after pretraining with demonstration data similar to the result in the single-agent setting. However, for scenarios 11vsGK and Counterattack Easy, which consist of 3 RL-controlled players and 7 RL-controlled teammates (excluding the goalkeeper), further PPO training gradually reduces the performance of the behavior cloned agent close to the level of PPO agents without pretraining. The decrease in performance after fine-tuning may be due to

several reasons. First, the network model we used may be too simple to handle environments with many players, as evident by the poor performance on the 11vGK scenario. Second, the presence of non-RL-controlled players adds significant variance to the environment, which adds difficulty for pretrained agents to generalize to these unseen situations.

Scenario	Number of left player	Number of controlled player	PPO without Pretraining	PPO with Pretraining	Scenic Policy	Behavior Cloning
11vsGK	11	3	0.11	0.125	0.4	0.12
Counterattack Easy	11	3	0.01	0.05	0.69	0.5
Pass and Shoot with Keeper	3	2	0.7013	0.948	0.51	0.4
Avoid Pass and Shoot	3	2	0.5856	0.861	0.32	0.4

Table 3.2: Table showing the average goal difference of the agents trained with and without demonstration data, and the performance of semi-expert Scenic policy and the behavior-cloned agents.

Chapter 4

Automatic Curriculum Learning via Environment Generation

Previous chapters introduce Scenic4RL's capability to allow RL researchers to model and specify distributions of environments. Yet, manually defining environment distributions is time-consuming, error-prone, and insufficient to facilitate many RL algorithms such as transfer learning [35] and emergent complexity methods [33] that require extensive sets of diverse scenarios. For example, in generated curriculum based curriculum learning method, RL agents are trained on a curriculum of scenarios that are simple at first but get progressively more difficult as the learning progresses. The idea is to let the RL agent start easy, gradually picking up skills in each marginally more challenging scenario until it can solve the target task. Clearly, it is infeasible to create such a curriculum for each target task manually: it is impossible to cover all variations and edge cases, and humans can not accurately build scenarios suitable to the agent's learning progress since we as humans can not accurately determine scenario difficulty for RL agents.

The solution is to automate the environment generation process. This chapter describes an alternative environment definition paradigm called Unsupervised Environment Design (UED)[8]. In this setting, instead of directly specifying the environment in the Scenic program, researchers provide an under-specified environment with free parameters that are used to automatically create a distribution over feasible environments. Each Scenic program now contains free environment parameters that an external agent can set to generate a new distribution of environments, from which the Scenic server can sample fully-specified RL environments. In the evaluation section, we modify and apply the state-of-the-art automatic curriculum learning algorithm, PAIRED, and evaluate its performance on custom Scenic scenarios in GRF.

4.1 Problem Definition

Automated Curriculum Learning

Automated curriculum learning methods aim to train the RL agent to solve previously unsolvable tasks with 1) high sample efficiency and 2) good generalizability to diverse environments. It achieves the goal by training the RL agent on automatically generated environments (curriculums) suitable for the RL agent's learning progress to facilitate incremental learning. These methods usually use a teacher network to create environments, and the RL agent is referred to as the student. At every iteration, the teacher network creates new scenarios based on the student's performance, and then the student is trained on these scenarios. The process repeats until the student can solve the target tasks. The student agent is then evaluated in similar but unseen environments to test its generalization ability.

Under-Specified POMDP

Sampled scenes from Scenic are fully-specified environments because they don't contain any non-determinism. Therefore, consistent with previous chapters, we can formulate a fully-specified environment as a POMDP. On the other hand, an under-specified environment is an environment with free environment parameters that control its features and behavior. Formally, it is formulated as an Under-specified Partially Observable Markov Decision Process (UPOMDP), which can be represented as a tuple $M = (S^M, A, O, T^M, \varepsilon^M, r^M, \theta)$. The difference between a UPOMDP and POMDP is the addition of θ, the free environment parameters that can be set at any timestep. Incorporating the environment parameters, $T^M : S \times A \times \theta \to S$ is the transition function and the rest are the same as those in POMDP. A setting of environment parameter θ can be combined with an under-specified environment M to produce a fully-specified POMDP M_θ.

Unsupervised Environment Design

Unsupervised Environment Design (UED) is the task of generating a distribution of full-specified environments from a given under-specified environment automatically. Ideally, we want the generated distribution to suit the RL agent's current ability to facilitate learning. Formally, the solution to UED can be characterized as an environment policy $\Lambda : \Pi \to \theta^T$ where Π is the set of possible RL policies and θ^T is the set of possible sequence of environment parameters. In plain words, an environment policy produces trajectories of environment parameters to curate the environment distribution to facilitate RL agent's learning.

Common UED policies include domain randomization, where the environment parameters are sampled randomly from the distribution, and minimax adversary, where an adversary agent sets the environment parameters to minimize the reward of the RL agent. However, as we describe in detail in the PAIRED algorithm section, both methods fail to generate a viable curriculum of environments in most contexts.

```
spawn_area = get_reg_from_edges(0, 80, -25, 25)

ego = LeftCM on spawn_area
opponent = RightGK on spawn_area

MyGK
ball = Ball ahead of ego by 2
```

Figure 4.1: Example Scenic script that defines a distribution of environments.

4.2 Approach

SCENIC Program as Under Specified POMDP

Scenic allows the user to define a scenario expressing a distribution of concrete scenes. Since each concrete scene is a fully-specified environment, such a scenario, encoded in a Scenic program, can be seen as defining an under-specified environment, with the environment variables being the random variables in the program.

For example, in the following simple Scenic script defining a 1v1 scenario between an RL-controlled attacker and an opponent goalkeeper, we define a distribution over the area where both players could spawn. If we train the RL agent on this scenario, at every reset() step, the Scenic server will randomly sample two points in the spawn area to determine the exact spawning locations to create a fully-specified scene. Therefore, Scenic implicitly implements an environment policy of domain randomization, where all the environment parameters are sampled randomly from the defined distributions.

Figure 4.2: Illustration of the 1v1 scenario defined in 4.1. The yellow rectangle represents the spawn area of both players.

However, the domain randomization policy is not optimal for creating a curriculum on which to train an RL agent to master the 1v1 skill. Since we randomly sample the spawning locations for the controlled player and the opponent from the spawn area, it will generate scenes that are not helpful in training. For example, it can generate trivially easy scenes if the controlled player with the ball spawns in front of the opponent goalkeeper. It can also generate impossible scenes where the RL agent can't score, e.g., if the opponent spawns closer to the ball than the RL player. While it is

```
1  X1 = Range (0, 80)
2  Y1 = Range (-25, 25)
3  X2 = Range (0, 80)
4  Y2 = Range (-25, 25)
5
6  ego = LeftCM at (X1 @ Y1)
7  opponent = RightGK at (X2 @ Y2)
8
9  MyGK
10 ball = Ball ahead of ego by 2
```

Figure 4.3: Equivalent Scenic script to Example 4.1

possible to create rules to prevent the generation of such scenes, it is time-consuming and infeasible to manually tune the distribution in complex scenarios with a large environment parameter space. Domain randomization also does not consider the RL agent's learning progress to create an adaptive curriculum.

We can expose all environment parameters in a Scenic scenario to let an external agent control the distribution of the environment. Therefore we can use an external agent to determine the environment parameters to shape the distribution to support the RL agent's learning. In this setting, all the distributions defined in the Scenic scenario are translated into a vector of environment parameters. While Range objects are mapped directly as an entry of the vector, area objects are translated into two variables representing the x and y coordinates. Figure 3 shows the equivalent scripts to the previous example with environment variables explicitly written as Range objects. The environment variable of the example script is therefore $\theta = [X_1, Y_1, X_2, Y_2]$.

To model a Scenic program as a UPOMDP, we provided an environment wrapper on Scenic4RL's Gym environment that exposes the environment parameters as a list. Furthermore, we rescale the environment parameters from -1 to 1 to allow easy integration with neural net models. The interface will scale back the input environment parameters according to the defined ranges when sampling scenes. The wrapped gym environment consists of two additional functions.

reset_random(): Randomly reset all environment parameters and reset the environment.

step_adversary(env_params): Set the environment parameters.

Using the UPOMDP wrapper, we can train another RL agent as the environment policy. For example, we can train a minimax adversary RL agent to set the environment parameters with the goal of minimizing the player agent's reward. However, this approach may not

work in most scenarios since the adversary agent can always create impossible environments if such configuration exists to prevent the player agent from scoring.

PAIRED Algorithm

PAIRED [8], or Protagonist Antagonist Induced Regret Environment Design, is an environment policy capable of generating challenging yet feasible environments suitable to the RL agent's capability. It trains an adversary environment agent whose goal is to minimize the difference between the reward of the player RL agent (protagonist) and the reward of a rival RL agent (antagonist). Because the environment agent's reward is based on the difference in learning progress between the two RL agents, the environment agent is encouraged to generate environments that are challenging to the protagonist but are solvable by the RL agents[1].

In the PAIRED method, we train three RL agents:

1. The protagonist, which is the agent we want to train to solve the task.
2. The antagonist, identical to the protagonist, acts as the protagonist's rival.
3. The adversarial environment agent, which generates the environment parameters to create a distribution of environment that is feasible and incrementally challenging for both the protagonist and the antagonist.

To calculate the reward for these agents during training, we define regret as the difference between the reward of the protagonist and the reward of the antagonist under a POMDP created from a UPOMDP under environment parameters $\vec{\theta}$.

$$REGRET(\pi^P, \pi^A) = \max U^{\vec{\theta}}(\pi^A) - E[U^{\vec{\theta}}(\pi^P)]$$

where $U = U^{\vec{\theta}}(\pi) = \sum_{i=0}^{T} r_t \gamma^t$ is the discounted cumulative rewards of an agent, and γ is the discount factor.

The three agents are randomly initialized at the start. In each training iteration, the adversarial environment agent will generate and set the environment parameters to tailor the distribution of environments to suit the protagonist's learning progress. Once the scenario is set, we run the protagonist and antagonist on the sample scenes from the scenario to estimate the regret. Then we update all agents based on the regret and repeat. The algorithm can be summarized as follows.

[1] If the training environment is trivial, then both the protagonist and the antagonist will do well, so the difference is low. If the environment is unsolvable, then the difference is zero. Therefore, the environment adversary agent has to devise the easiest scenario protagonist can't solve but antagonist can to maximize its rewards.

Algorithm 1: PAIRED Algorithm

Input: A SCENIC program P defining an UPOMDP M;
Randomly initialize Protagonist π^P, Antagonist π^A, and Environment Adversary $\tilde{\Lambda}$;
while *not converged* **do**
  Use adversary to generate environment parameters: $\vec{\theta} \sim \tilde{\Lambda}$. Use to create POMDP $M_{\vec{\theta}}$;
  Collect Protagonist trajectory τ^P in $M_{\vec{\theta}}$. Compute: $U^{\vec{\theta}}(\pi^P) = \sum_{i=0}^{T} r_t \gamma^t$;
  Collect Antagonist trajectory τ^A in $M_{\vec{\theta}}$. Compute: $U^{\vec{\theta}}(\pi^A) = \sum_{i=0}^{T} r_t \gamma^t$;
  Compute: $\textsc{Regret}(\pi^P, \pi^A) = \max_{\tau^A} U^{\vec{\theta}}(\tau^A) - \mathbb{E}_{\tau^P}[U^{\vec{\theta}}(\tau^P)]$;
  Train Protagonist policy π^P with RL update and reward $R(\tau^P) = -\textsc{Regret}$;
  Train Antagonist policy π^A with RL update and reward $R(\tau^A) = \textsc{Regret}$;
  Train Adversary policy $\tilde{\Lambda}$ with RL update and reward $R(\tau^{\tilde{\Lambda}}) = \textsc{Regret}$;
end

4.3 Evaluation

The experiments in this section aim to compare the learning performance of agents trained with and without an unsupervised curriculum learning algorithm (PAIRED [8]). We build 4 mini-game programs encoded in SCENIC, each containing a distribution of environment resembling a common real-world soccer situation. Each mini-game consists of 1) a training scenario that contains samplable environment parameters that define a broad distribution of the initial players and ball positions and/or opponent behaviors, 2) 2 to 4 testing scenarios each defining a concrete environment with all environment parameters set within the training distribution. The testing scenario ranges in difficulties as defined by human knowledge. The learning performance is measured by the average return of the agent evaluated on the training and all testing scenarios. Due to computation resource limitations, the mini-game contains fewer players (no more than 6) than the 22-player real soccer game.

Experiment Setup

For each mini-game, we train two agents using PPO [24] with identical network structure and hyperparameters, one with the PAIRED algorithm and the other with the domain randomization algorithm, on the training scenario for 5M timesteps and repeated for 2 seeds. The agent is evaluated on each scenario for 500 episodes.

For Google Research Football settings, the observation type is stacked super minimap representation: a 72*96 image with 16 channels representing the location of players, balls, and the currently controlled player over the last 4 timestamps. An episode of the game terminates on scoring, ball possession change, or upon reaching 400 timesteps. The reward is 1 if the agent scores, -1 if the opponent scores, and 0 otherwise. All non-controlled players

are controlled by the built-in AI, except players with pre-defined behaviors encoded in the SCENIC program.

Scenic Scenarios

Every scenario is a SCENIC program. All samplable environment parameters are in the form of Range(low, high), denoting a continuous range from low to high. If a scenario contains samplable environment parameters, during each reset, their values are updated by the adversarial environment agent if using PAIRED [8], or chosen uniformly random if using domain randomization.

1v1 (Restricted) In this scenario, the agent controls a striker who spawns in a narrow rectangular area in front of the opponent's goal trying to score. The opponent goalkeeper spawns in a rectangular area between the striker and the goal. The difficulties and strategies vary depending on the spawn locations. If the initial position of the goalkeeper doesn't block the striker's shot, the agent can simply shoot at once to score. Otherwise, the agent has to learn to pass the goalkeeper before shooting.

There are 3 test scenarios ranging from easy to hard. In test scenario 0 (easy), the opponent goalkeeper spawns on the opposite side of the striker and hence it is easy for the striker to score. In scenario 1 (medium), the goalkeeper blocks half of the goal, but it is still feasible to score by shooting immediately. In scenario 3 (hard), the striker spawns further away from the goal and has the goalkeeper directly in front; therefore the agent has to pass the goalkeeper, or move to the side to create an opening.

1v1 (Free) This scenario has the same setting as 1v1 (Restricted) except that both the striker and the goalkeeper can spawn anywhere on the right half-field. Compared to 1v1 (Restricted), the training scenario contains a broader range of samplable environment parameters enabling the goalkeeper to spawn behind the striker (easy) or next to the ball (hard). The testing scenarios are the training and testing scenarios in 1v1 (Restricted).

Choose Passing The initial player is placed with the ball in the middle, and two teammates spawn near the two corners of the goal ready to shoot once they received the ball. An opponent defender spawns within a rectangular area between the two teammates, and the opponent goalkeeper spawns in front of the initial player to block direct shooting. The difficulty varies based on the position of the opponent defender since the agent has to learn to pass the ball to the teammate away from the opponent to have the best scoring chance.

There are 2 testing scenarios of equal difficulty. The first scenario has the opponent defender blocking the top teammates while the second scenario has the defender blocking the bottom teammates.

3v2 4 behaviors This scenario resembles an attacking scenario in which 3 attackers are spawned within the opponent's penalty area, facing an opponent defender and the goalkeeper in front of the goal. The attackers controlled by the agent are ready to perform a pincer movement to score. The training scenario not only defines a distribution of players' initial position but also defines 4 different defense strategies encoded as SCENIC behaviors for the opponent defender. A samplable environment parameter (Range(0,1)) controls which

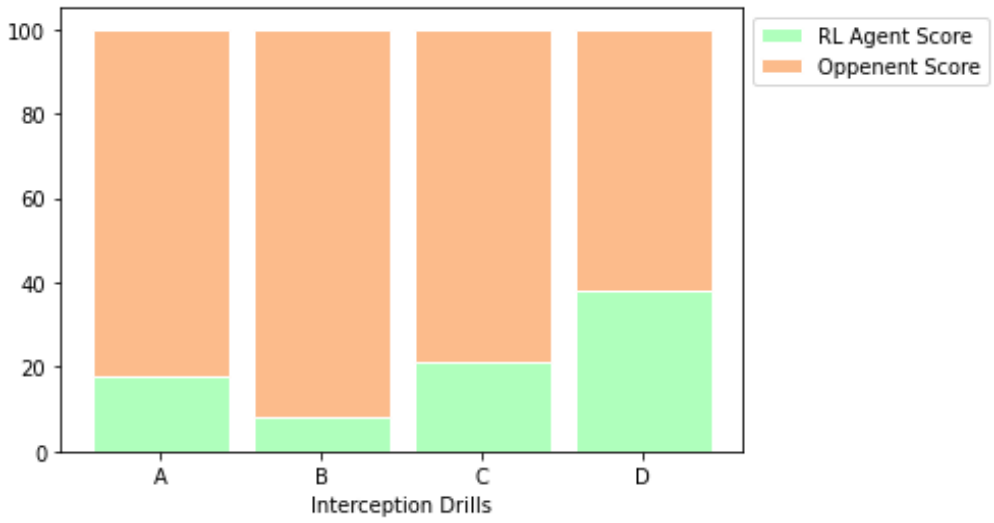


Figure 4.4: Bar chart showing the performance of the four scenic policies facing the built-in AI.

strategy to use in the game. The 4 testing scenarios test the agent's performance under each of the 4 defense strategies.

The 4 defense strategies are summarized as follows. The performance of the strategies against the built-in AI bot is shown in figure 4.4.

A The opponent will keep moving towards the ball and intercept the ball.

B The opponent will move towards the player who owns the ball.

C The opponent will move towards the midpoint of the closest left player and the left player closest to the ball owner before trying to intercept the ball.

D The opponent will help the goal keeper guard the door against the RL agent.

Domain Randomization Experiments

While the Impala-CNN-based [10] model used in the Google Research Football paper [16] can handle complex 11v11 full game scenarios, despite our best effort, we can not replicate their result after translating the model to our codebase in Pytorch. Therefore we develop a lightweight CNN model inspired by the Nature CNN model [18] as our feature extractor that is then used by the actor and critic. Figure 4.10 shows the network architecture, and Table 4.2 lists the hyperparameters used in the experiments. To ensure our model is capable of learning mini-games scenarios, we first trained and tested the agent network on some soccer scenarios defined in the single agent mini-game benchmark. We train the agent with the

Scenario	Our Avg Return	GRF Avg Return
EasyCrossing	**0.87**	0.5
Pass and Shoot	**0.85**	0.62
AvoidPassShoot (10M)	0.66	**0.72**
Defense2V2HighPassForward	-0.32	**-0.14**
Defense3V3CrossFromSide	-0.53	**-0.16**
DefenseDefenderOpponentZigzag	**0**	-0.17

Table 4.1: Comparison of average returns of our agents and returns of GRF agents on mini-games with domain randomization.

domain randomization algorithm [31] with the best-found hyperparameters for 5M timesteps. In each episode, we uniformly randomly sample values for environment parameters in the SCENIC program. The results are summarized in table 4.1. Compared to the GRF model, we find that our model achieves good performance in most offensive scenarios, but falls short in defensive scenarios. Therefore in this section, we focus on offensive mini-game scenarios.

PAIRED Experiments

For each PAIRED experiment, we train 3 RL agents in total. We train 2 player agents learning to play the minigame: the protagonist and the antagonist, and an adversarial environment agent that builds environments based on the player agents' abilities. The protagonist and the antagonist are identical to the RL agent in domain randomization experiments and are trained using the same hyperparameters. The adversarial environment agent has the same observation space as the player agents but has a continuous action space with the dimension equal to the number of samplable environment parameters in the training scenario. Since the parameters have different ranges, to facilitate training, we squash the model output to [-1,1] using the hyperbolic tangent function and scale them back in the environment abstraction layer. One downside of this approach is that we no longer have an analytical form for entropy and have to estimate from samples instead.

There is a small possibility that the adversarial environment agent outputs a set of parameters that will result in an invalid scenario. For example, the initial position of two players may be too close so they overlap. In this case, we randomly sample a valid set of parameters, apply it to build the environment, and relabel the agent action in the storage with these valid parameters.

Figure 4.5, 4.6, 4.7, 4.8 show the average returns of the protagonist agent in PAIRED and the returns of domain randomization agent on the testing scenarios of each minigame.

The learning curves are included in Figure 4.9.

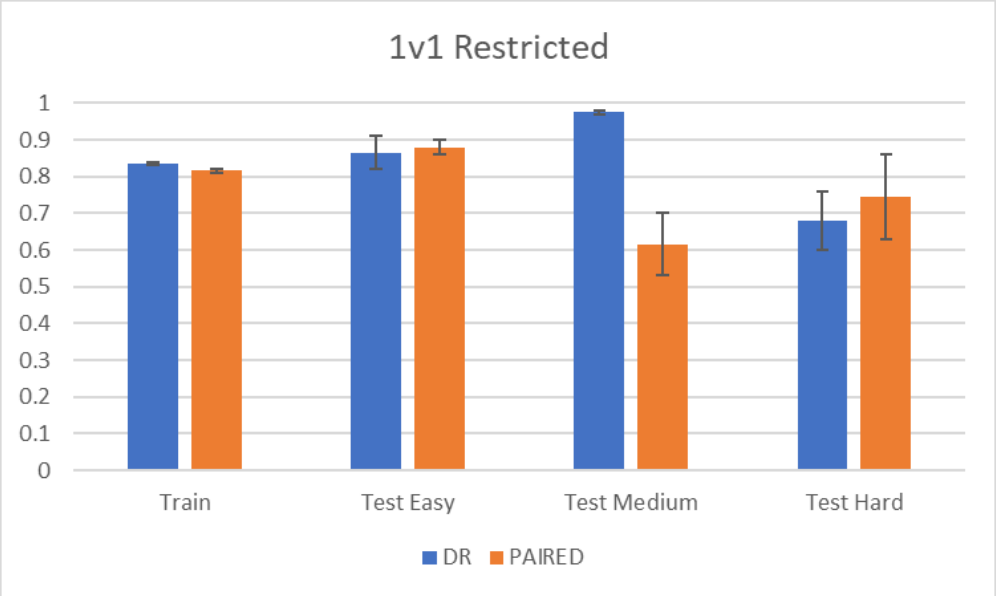


Figure 4.5: Evaluations Results of trained RL agents on 1v1 Restricted

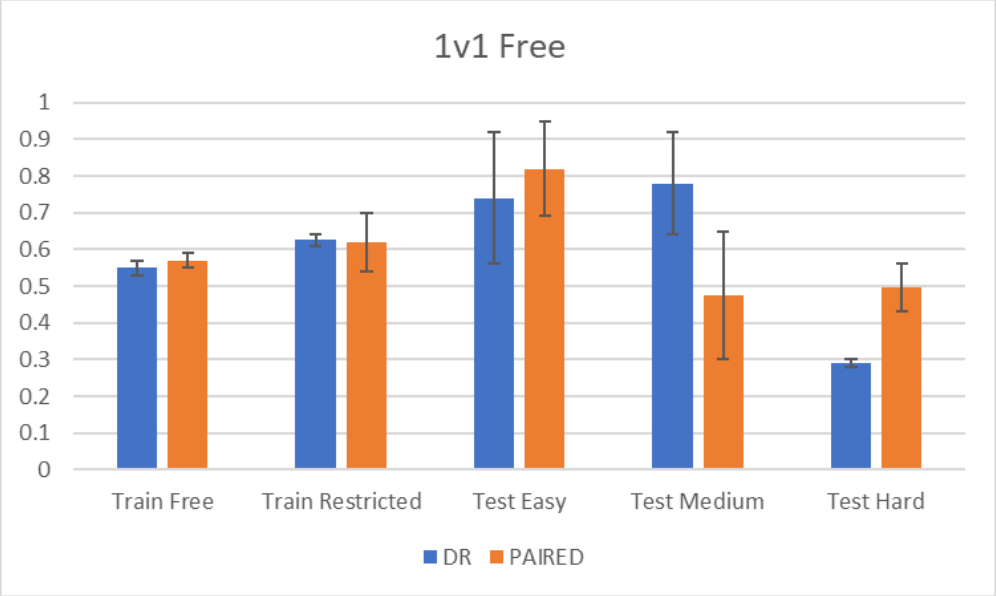


Figure 4.6: Evaluations Results of trained RL agents on 1v1 Free

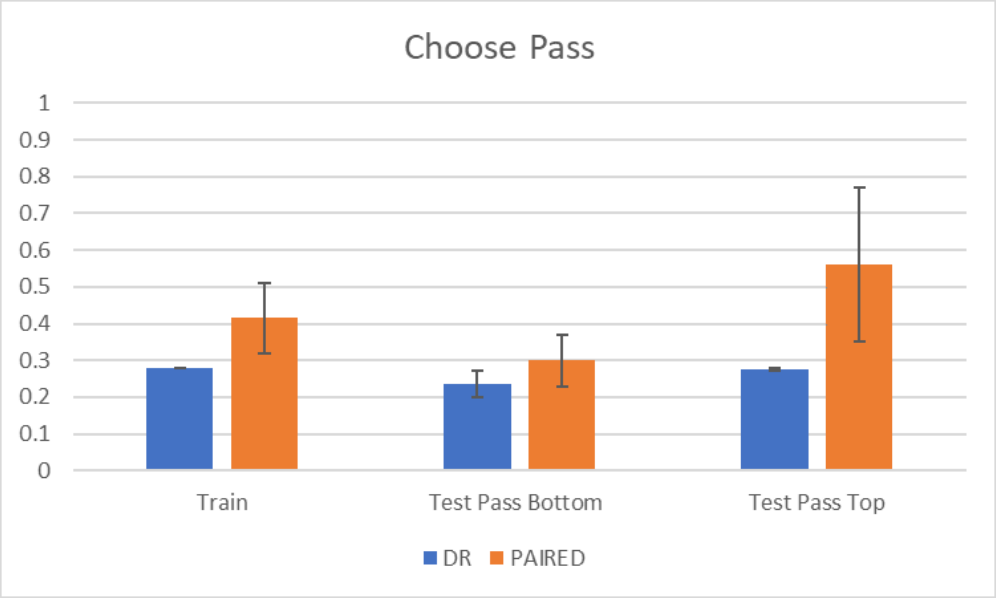


Figure 4.7: Evaluations Results of trained RL agents on Choose Pass

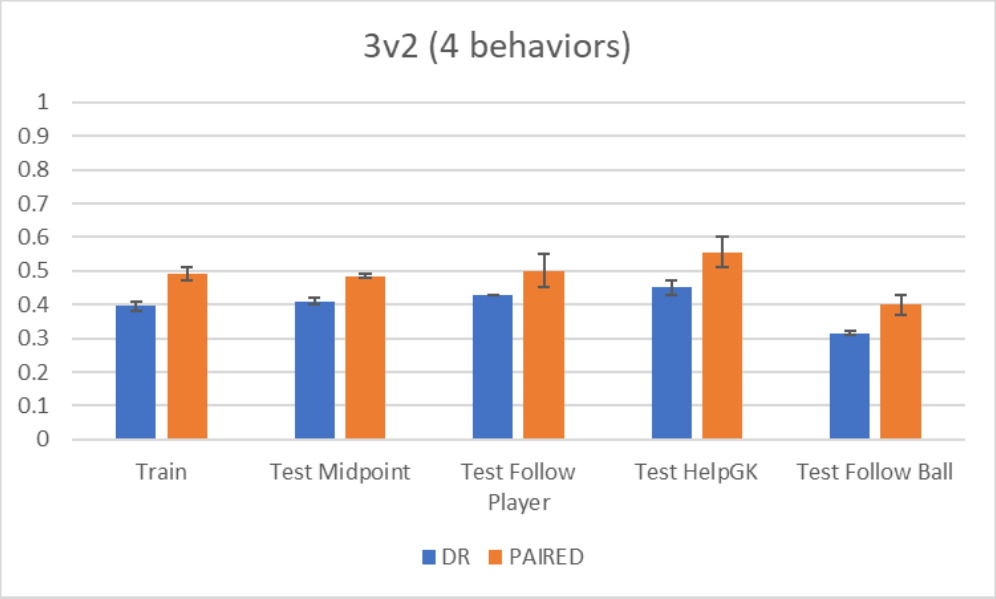


Figure 4.8: Evaluations Results on 3v2 4 behaviors

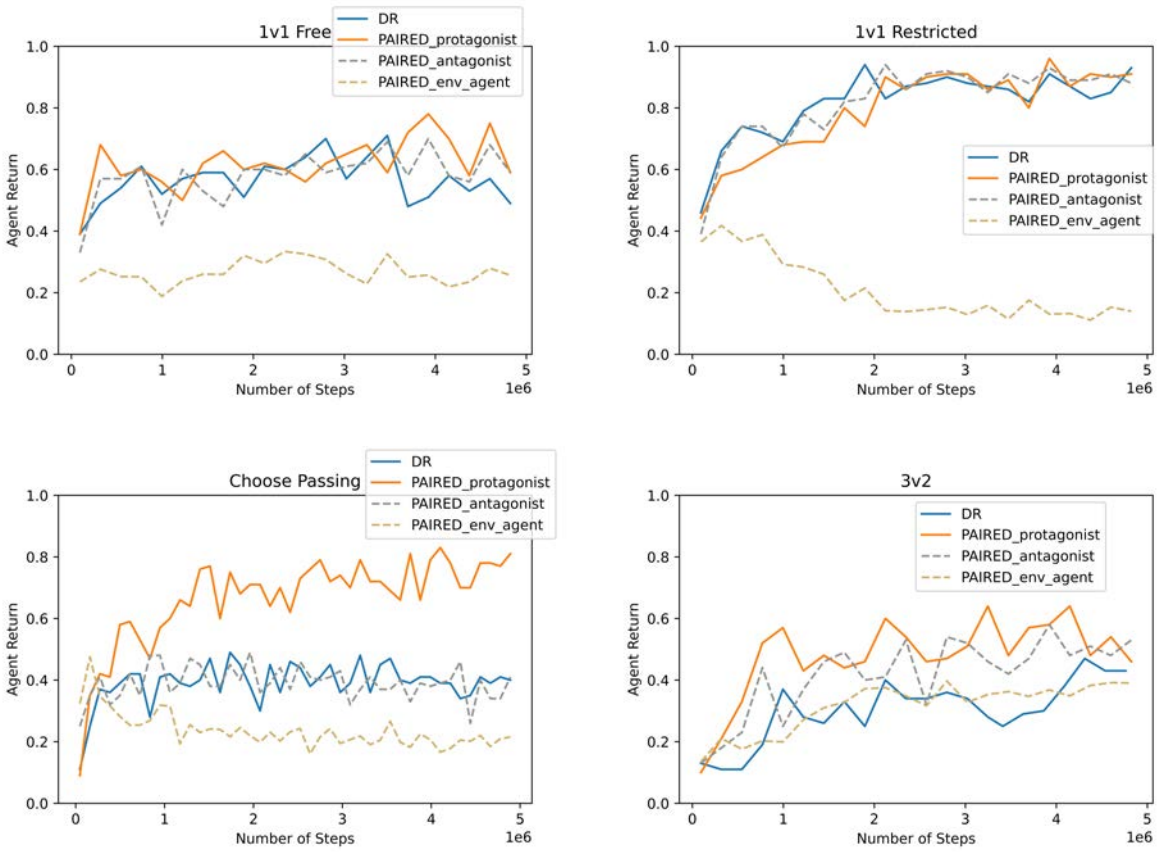


Figure 4.9: Comparing mean agent return, mean adversary agent return, and mean environment return of PAIRED with domain randomization in the four scenarios

Analysis

In this section, we define PAIRED agent as the protagonist agent in the PAIRED algorithm, and the DR agent as the RL agent training using the domain randomization algorithm. From the evaluation results shown in Figure 4.9, we find that PAIRED agent achieves similar or better returns on all training scenarios compared to DR agent, with PAIRED agent generally performing better on more difficult test scenarios. However, PAIRED agent's lack of performance on the 1v1 Medium testing in both the free and restricted training setting suggests that the agent might forget how to play easy scenarios as the training progresses.

The data confirm the hypothesis that PAIRED agents' ability to perform well in difficult environments is due to the increasingly challenging curriculum built by the adversarial environment agent over time. As the player agents improve their skills as the training continues, the adversarial environment agent also learns to output environment parameters that produce challenging scenarios relative to the player agents' current abilities, implicitly creating

a curriculum of training environment of increasing difficulties from the given distribution of training scenarios. This is evident comparing the testing results of PAIRED and DR agents in 1v1 Restricted and 1v1 Free mini-games. The 1v1 free training scenario defines a much broader environment distribution (16M pixel area) compared with the 1v1 restricted scenario (0.4M pixel area), so with domain randomization, the RL agent gets less chance of seeing the difficult environment during training to develop skills to avoid or pass the opponent goalkeeper, while the PAIRED agents are more likely to replay these hard environments to pick up the skill. Therefore the DR agent performs significantly poorly on the hard 1v1 testing scenario if trained in 1v1 free than the DR agent trained in 1v1 restricted, which achieves similar results as that of PAIRED agent.

However, this regret-driven curriculum generation may cause the PAIRED agent to play less challenging scenarios and forget the skills to solve easy environments, as evident by the PAIRED agent's low returns on 1v1 test medium for both 1v1 restricted and 1v1 free mini-games. As the adversarial environment agent learns to choose challenging scenarios for the player agents, it slowly converges those scenarios that require more complex actions to solve since these scenarios usually lead to higher regret. (Why? Consider a trivial 1v1 scenario of an empty goal, both the protagonist and the antagonist would easily learn to shoot immediately and score, achieving near-perfect returns every time so the regret is almost always 0. However for a challenging scenario such as a close standoff between the striker and the goalkeeper, even an adept agent who knows how to pass the goalkeeper may fail to score due to the stochasticity of the game, and thus the regret would often be high). Therefore as the training progresses, it is easy for the adversarial environment agent to overfit to a specific region of parameters representing just one challenging configuration. In the "choose pass" mini-game, the two testing scenarios are almost mirrored images of each other: one requires the agent to pass to the open teammate at the bottom and the other requires passing to the top player, yet PAIRED agent only performs well on one of the two scenarios despite that these two scenarios test the same skill.

It is also interesting to note that PAIRED agent overperforms the DR agent on a 3v2 mini-game on all scenarios, including the training scenarios. The 4 testing scenarios of this minigame corresponds to the 4 defense strategies used by the opponent defender, and the training scenario consists of a single continuous random variable to select strategies and a distribution of the player's initial position within the opponent penalty area. This suggests that PAIRED performs well in building curriculums from behaviors distributions in addition to initial-position distributions, and is a promising result for future works such as using the adversarial environment agent to select compose scenarios for training.

The performance of the PAIRED algorithm depends heavily on the adversarial environment agent's ability to output diverse, yet progressively more difficult training environments. Therefore, we need to be aware of the risk of overfitting the environment agent, especially in complex minigames with multiple configurations of difficult scenarios. One potential remedy is to employ an exploration strategy encouraging the environment agent to test previously-unseen parameter spaces to avoid getting mired in one difficult environment configuration and causing the player agents to also overfit.

Hyperparameters	Value
adv-entropy-coef	0.01
adv-max-grad-norm	0.5
adv-num-mini-batch	1
adv-ppo-epoch	5
algo	ppo
alpha	0.99
clip-param	0.27
entropy-coef	0.01
eps	0.00001
gae-lambda	0.95
gamma	0.993
lr	0.00008
max-grad-norm	0.5
num-mini-batch	8
num-steps	128
ppo-epoch	4
value-loss-coef	0.5

Table 4.2: Hyperparameters used in PAIRED experiments

```
GFootballNetwork(
  (cnn_layer): feature_extractor(
    (conv1): Conv2d(16, 32, kernel_size=(8, 8), stride=(4, 4))
    (conv2): Conv2d(32, 64, kernel_size=(4, 4), stride=(2, 2))
    (conv3): Conv2d(64, 32, kernel_size=(3, 3), stride=(1, 1))
    (fc1): Linear(in_features=1280, out_features=512, bias=True)
  )
  (critic): Linear(in_features=512, out_features=1, bias=True)
  (actor): Linear(in_features=512, out_features=19, bias=True)
)

GFootballAdversaryNetwork(
  (cnn_layer): feature_extractor(
    (conv1): Conv2d(16, 32, kernel_size=(8, 8), stride=(4, 4))
    (conv2): Conv2d(32, 64, kernel_size=(4, 4), stride=(2, 2))
    (conv3): Conv2d(64, 32, kernel_size=(3, 3), stride=(1, 1))
    (fc1): Linear(in_features=1280, out_features=512, bias=True)
  )
  (critic): Linear(in_features=512, out_features=1, bias=True)
  (mean_net): Linear(in_features=512, out_features=6, bias=True)
)
```

Figure 4.10: Network structure of our lightweight RL agent.

Chapter 5

Conclusion

In this work, we introduced the benefit of adopting a formal scenario specification language, Scenic, to assist RL researchers in flexibly modeling and generating the environment to facilitate training. We released an open-source interface, Scenic4RL, that allows seamless integration of Scenic with the RL framework through a Gym environment corresponding to the scene sampled from the Scenic program. As a result, RL researchers can model, specify, and generate distributions of training environments in both single-agent and multi-agent settings. We interfaced Scenic with the Google Research Football environment and released a benchmark of minigames to assist training and evaluation of RL agents. In addition, we demonstrated multiple methods researchers could use the interface to facilitate training, debugging, and evaluating RL policies. Moreover, researchers can manually define expert strategies as probabilistic behaviors in the Scenic program and generate demonstration data using the interface to pre-train RL policies. Chapter 4 showed that researchers can transform a Scenic program into an under-specific POMDP to allow an external environment policy to control the environment distribution. We adopted the PAIRED algorithm as the environment policy to automatically generate a set of training scenarios to form a curriculum to support learning. Last, we compared the performance of agents trained using PAIRED and agents trained with domain randomization.

5.1 Future Works

Integration with Other Environments

We interfaced Scenic with Google Research Football to demonstrate the benefits of adopting a scenario specification language in RL development. Both Scenic and the Scenic4RL interface can easily be integrated with other RL environments and simulators. There are two methods to adapt our framework to a new environment. First, if Scenic already models the environment, then the integration is relatively straightforward. The user has to define the desired Gym environment format, such as the reward function and the format of observation

space and action space, in the Scenic4RL interface. Second, if Scenic does not yet support the environment, the user needs to integrate it with the Scenic language before defining the RL environment specifications in the Scenic4RL interface.

Scenario Composition

In this work, RL agents are trained on one Scenic scenario to learn a single skill. Therefore, we did not explore Scenic's scenario composition functionality that enables scenes to be generated from multiple Scenic scenarios. Researchers can write reusable modular scenarios representing parts of the scene and design a method to compose them to create dynamic, complex environments. In addition, we can extend the UPOMDP wrapper of Scenic4RL to allow an external policy to control the composition logic. For example, to achieve automatic curriculum learning of the 11vs11 full game, researchers can provide a set of training scenarios for each skill and train a teacher agent to use scenario composition to create training environments that facilitate the learning of multiples skills.

Bibliography

[1] Abdus Salam Azad, Edward Kim, Qiancheng Wu, Kimin Lee, Ion Stoica, Pieter Abbeel, and Sanjit A. Seshia. *Scenic4RL: Programmatic Modeling and Generation of Reinforcement Learning Environments*. 2021. DOI: 10.48550/ARXIV.2106.10365. URL: https://arxiv.org/abs/2106.10365.

[2] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. "The arcade learning environment: An evaluation platform for general agents". In: *Journal of Artificial Intelligence Research* 47 (2013), pp. 253–279.

[3] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. "Openai gym". In: *arXiv preprint arXiv:1606.01540* (2016).

[4] Han Cha and Seong-Lyun Kim. "A Reinforcement Learning Approach to Dynamic Spectrum Access in Internet-of-Things Networks". In: *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*. 2019, pp. 1–6. DOI: 10.1109/ICC.2019.8762091.

[5] Karl Cobbe, Chris Hesse, Jacob Hilton, and John Schulman. "Leveraging procedural generation to benchmark reinforcement learning". In: *International conference on machine learning*. 2020.

[6] Karl Cobbe, Oleg Klimov, Chris Hesse, Taehoon Kim, and John Schulman. "Quantifying generalization in reinforcement learning". In: *International Conference on Machine Learning*. 2019.

[7] Edward Kim Daniel Fremont. *Scenic interfaced simulators*. https://scenic-lang.readthedocs.io/en/latest/simulators.html. 2021.

[8] Michael Dennis, Natasha Jaques, Eugene Vinitsky, Alexandre Bayen, Stuart Russell, Andrew Critch, and Sergey Levine. "Emergent Complexity and Zero-shot Transfer via Unsupervised Environment Design". In: *Neural Information Processing Systems (NeurIPS)*. 2020.

[9] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. "Benchmarking deep reinforcement learning for continuous control". In: *International conference on machine learning*. 2016.

[10] Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymir Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu. *IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures*. 2018. arXiv: 1802.01561 [cs.LG].

[11] FIFA. *Laws of the Game*. https://ussoccer.app.box.com/s/xx3byxqgodqtl1h15865/file/850765570638. 2021.

[12] Daniel J. Fremont, Tommaso Dreossi, Shromona Ghosh, Xiangyu Yue, Alberto L. Sangiovanni-Vincentelli, and Sanjit A. Seshia. "Scenic: a language for scenario specification and scene generation". In: *PLDI*. ACM, 2019, pp. 63–78.

[13] Daniel J. Fremont, Edward Kim, Tommaso Dreossi, Shromona Ghosh, Xiangyu Yue, Alberto L. Sangiovanni-Vincentelli, and Sanjit A. Seshia. "Scenic: A Language for Scenario Specification and Data Generation". In: *CoRR* abs/2010.06580 (2020). arXiv: 2010.06580. URL: https://arxiv.org/abs/2010.06580.

[14] Shiyu Huang, Wenze Chen, Longfei Zhang, Shizhen Xu, Ziyang Li, Fengming Zhu, Deheng Ye, Ting Chen, and Jun Zhu. *TiKick: Towards Playing Multi-agent Football Full Games from Single-agent Demonstrations*. 2021. DOI: 10.48550/ARXIV.2110.04507. URL: https://arxiv.org/abs/2110.04507.

[15] Hiroaki Kitano, Minoru Asada, Yasuo Kuniyoshi, Itsuki Noda, and Eiichi Osawa. "RoboCup: The Robot World Cup Initiative". In: *Proceedings of the First International Conference on Autonomous Agents*. AGENTS '97. Marina del Rey, California, USA: Association for Computing Machinery, 1997, pp. 340–347. ISBN: 0897918770. DOI: 10.1145/267658.267738. URL: https://doi.org/10.1145/267658.267738.

[16] Karol Kurach, Anton Raichuk, Piotr Stańczyk, Michał Zajac, Olivier Bachem, Lasse Espeholt, Carlos Riquelme, Damien Vincent, Marcin Michalski, Olivier Bousquet, et al. "Google research football: A novel reinforcement learning environment". In: *AAAI Conference on Artificial Intelligence*. 2020.

[17] Xi Lan, Yuansong Qiao, and Brian Lee. "Towards Pick and Place Multi Robot Coordination Using Multi-agent Deep Reinforcement Learning". In: Feb. 2021, pp. 85–89. DOI: 10.1109/ICARA51699.2021.9376433.

[18] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. *Deep learning*. May 2015. URL: https://www.nature.com/articles/nature14539#citeas.

[19] Kimin Lee, Kibok Lee, Jinwoo Shin, and Honglak Lee. "Network randomization: A simple technique for generalization in deep reinforcement learning". In: *International Conference on Learning Representations*. 2020.

[20] Kimin Lee, Younggyo Seo, Seunghyun Lee, Honglak Lee, and Jinwoo Shin. "Context-aware dynamics model for generalization in model-based reinforcement learning". In: *International Conference on Machine Learning*. 2020.

[21] Siqi Liu, Guy Lever, Josh Merel, Saran Tunyasuvunakool, Nicolas Heess, and Thore Graepel. "Emergent Coordination Through Competition". In: (2019). DOI: 10.48550/ARXIV.1902.07151. URL: https://arxiv.org/abs/1902.07151.

[22] Frans A. Oliehoek and Christopher Amato. *A Concise Introduction to Decentralized POMDPs*. 1st. Springer Publishing Company, Incorporated, 2016. ISBN: 3319289276.

[23] OpenAI, Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique P. d. O. Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. *Dota 2 with Large Scale Deep Reinforcement Learning*. 2019. arXiv: 1912.06680 [cs.LG].

[24] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. *Proximal Policy Optimization Algorithms*. 2017. arXiv: 1707.06347 [cs.LG].

[25] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. "Proximal policy optimization algorithms". In: *arXiv preprint arXiv:1707.06347* (2017).

[26] Sanjit A. Seshia, Dorsa Sadigh, and S. Shankar Sastry. "Towards Verified Artificial Intelligence". In: *ArXiv e-prints* (July 2016). arXiv: 1606.08514.

[27] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Artfhur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play". In: *Science* 362.6419 (2018), pp. 1140–1144.

[28] Arambam James Singh, Akshat Kumar, and Hoong Chuin Lau. "Hierarchical Multiagent Reinforcement Learning for Maritime Traffic Management". In: *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*. AAMAS '20. Auckland, New Zealand: International Foundation for Autonomous Agents and Multiagent Systems, 2020, pp. 1278–1286. ISBN: 9781450375184.

[29] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT Press, 2018.

[30] Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, Timothy Lillicrap, and Martin Riedmiller. "Deepmind control suite". In: *arXiv preprint arXiv:1801.00690* (2018).

[31] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. *Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World*. 2017. arXiv: 1703.06907 [cs.RO].

[32] Oriol Vinyals, Timo Ewalds, Sergey Bartunov, Petko Georgiev, Alexander Sasha Vezhnevets, Michelle Yeo, Alireza Makhzani, Heinrich Küttler, John Agapiou, Julian Schrittwieser, John Quan, Stephen Gaffney, Stig Petersen, Karen Simonyan, Tom Schaul, Hado van Hasselt, David Silver, Timothy Lillicrap, Kevin Calderone, Paul Keet, Anthony Brunasso, David Lawrence, Anders Ekermo, Jacob Repp, and Rodney Tsing. *StarCraft II: A New Challenge for Reinforcement Learning*. 2017. arXiv: 1708.04782 [cs.LG].

[33] Rui Wang, Joel Lehman, Aditya Rawal, Jiale Zhi, Yulun Li, Jeff Clune, and Kenneth O. Stanley. *Enhanced POET: Open-Ended Reinforcement Learning through Unbounded Invention of Learning Challenges and their Solutions*. 2020. DOI: 10.48550/ARXIV.2003.08536. URL: https://arxiv.org/abs/2003.08536.

[34] Weirui Ye, Shaohuai Liu, Thanard Kurutach, Pieter Abbeel, and Yang Gao. *Mastering Atari Games with Limited Data*. 2021. DOI: 10.48550/ARXIV.2111.00210. URL: https://arxiv.org/abs/2111.00210.

[35] Wenhao Yu, Jie Tan, C. Karen Liu, and Greg Turk. *Preparing for the Unknown: Learning a Universal Policy with Online System Identification*. 2017. DOI: 10.48550/ARXIV.1702.02453. URL: https://arxiv.org/abs/1702.02453.

[36] Won Yun, Sungwon Yi, and Joongheon Kim. "Multi-Agent Deep Reinforcement Learning using Attentive Graph Neural Architectures for Real-Time Strategy Games". In: Oct. 2021, pp. 2967–2972. DOI: 10.1109/SMC52423.2021.9658625.