Programmatic Modeling and Generation of Real-time Strategic Soccer Environments for Reinforcement Learning

Abdus Salam Azad*, **Edward Kim***, **Mark Wu, Kimin Lee,**
Ion Stoica, Pieter Abbeel, Alberto Sangiovanni-Vincentelli, Sanjit A. Seshia

University of California, Berkeley

Abstract

The capability of a reinforcement learning (RL) agent heavily depends on the diversity of the learning scenarios generated by the environment. Generation of diverse realistic scenarios is challenging for real-time strategy (RTS) environments. The RTS environments are characterized by intelligent entities/non-RL agents cooperating and competing with the RL agents with large state and action spaces over a long period of time, resulting in an infinite space of feasible, but not necessarily realistic, scenarios involving complex interaction among different RL and non-RL agents. Yet, most of the existing simulators rely on randomly generating the environments based on predefined settings/layouts and offer limited flexibility and control over the environment dynamics for researchers to generate diverse, realistic scenarios as per their demand. To address this issue, for the first time, we formally introduce the benefits of adopting an existing formal scenario specification language, SCENIC, to assist researchers to *model* and *generate* diverse scenarios in an RTS environment in a flexible, systematic, and programmatic manner. To showcase the benefits, we interfaced SCENIC to an existing RTS environment Google Research Football(GRF) simulator and introduced a benchmark consisting of 32 realistic scenarios, encoded in SCENIC, to train RL agents and testing their generalization capabilities. We also show how researchers/RL practitioners can incorporate their domain knowledge to expedite the training process by intuitively modeling stochastic programmatic policies with SCENIC.

1 Introduction

Deep reinforcement learning (RL) has emerged as a powerful method to solve a variety of sequential decision-making problems, including board games (Silver et al. 2017, 2018), video games (Mnih et al. 2015; Vinyals et al. 2019), and robotic manipulation (Kalashnikov et al. 2018). These successes rely heavily on widely-used simulation environments (Bellemare et al. 2013; Brockman et al. 2016) and benchmarks (Cobbe et al. 2020; Duan et al. 2016; Tassa et al. 2018). However, regardless of a long history of RL benchmarks, the existing RL environments/simulators are insufficient to properly train, test, and benchmark RL algorithms for real-time strategy (RTS) environments such as Starcraft (Vinyals et al. 2017), Dota2 (OpenAI et al. 2019), and soccer (Kurach et al. 2020), due to their lack of support for modeling diverse scenarios involving sophisticated interactive behaviors.

These RTS environments are characterized by unique characteristics that require special support for modeling. The environments involve intelligent entities/non-RL agents cooperating and competing with the RL agents with large state and action spaces over a long horizon. This opens up extremely diverse strategies consisting of numerous interactive behaviors. Yet, most of the existing simulators rely on randomly generating the environments based on predefined settings/layouts and offer limited flexibility and control to the researchers over the environment dynamics to generate diverse realistic scenarios. As a result RL research face at least two fundamental challenges: (i) the lack of diverse and realistic training data often leads to lack of generalization (Cobbe et al. 2019, 2020; Lee et al. 2020a,b), and (ii) the lack of flexibility and control over the environment dynamics makes it hard to generate realistic evaluation scenarios to comprehensively test generalization in these complex RTS environments.

To address this issue, for the first time to the best of our knowledge, we introduce the benefits of adopting an existing formal scenario specification language, SCENIC, to assist researchers to *model* and *generate* diverse realistic scenarios in an RTS environment in a flexible, systematic, and programmatic manner. Each SCENIC program represents a Markov Decision Process (MDP) and provides high-level syntax and semantics, backed by its own compiler, to intuitively and quickly model diverse and complex interactive scenarios to train RL agents and test their generalization capabilities. Furthermore, it allows researchers/RL practitioners to incorporate their domain knowledge into the training process by generating offline data with stochastic programmatic policies written in high-level intuitive syntax of SCENIC. To demonstrate the benefits, we interfaced SCENIC to an existing RTS environment, Google Research Football (GRF) (Kurach et al. 2020).

Our contributions are as follows:

- For the first time, we introduce the benefits of adopting a scenario specification language to (1) flexibly model interactive scenarios to train RL agents, (2) test their generalization capability, and (3) program stochastic RL poli-

*These authors contributed equally.

cies to generate demonstration data.

- We open-sourced our SCENIC's interface to GRF environment along with our 32 scenarios, 5 stochastic policies, and libraries encoded in SCENIC to assist researchers to build upon them to easily model diverse and sophisticated scenarios.

2 Related work

Environment Generation in RL: In literature, several techniques have been adopted to generate a rich variation of learning scenarios, primarily to promote or, ensure generalization. Techniques such as changing background with natural videos (Zhang, Wu, and Pineau 2018), introducing sticky actions (Machado et al. 2018) have been attempted, but are not robust enough. To ensure generalization, (Lee et al. 2020b) and (Seo et al. 2020) generated training and testing scenarios by randomly sampling from different regions of parameter space. Similar to supervised learning, the use of separate train and test sets have also been adopted (Nichol et al. 2018; Cobbe et al. 2020, 2019; Justesen et al. 2018), typically using techniques such as Procedural Content Generation (Hendrikx and et al 2013), which has traditionally been used to automatically generate levels in video games. However, most of these focus on discrete domain, typically the dataset generation process is opaque, and it can be difficult to quantify or, reason about how different (or, similar) these train and test sets are, because the generation process often use random numbers to generate different configurations.

On the contrary, a few manually scripted scenario benchmarks are proposed with respect to a few RTS RL environments with limitations. For StarCraft (Vinyals et al. 2017), only two benchmark scenarios (Uriarte and Ontañón 2021; Samvelyan and et al 2019) have been proposed. Both of these model different initial states but leave the behavior generation to either a learned RL agent or AI bots that are provided by the StarCraft environment, which are considered as blackbox agents. As a result, a sophisticated modeling and control over the behaviors of non-RL agents to create specific types of scenarios is not possible, severely restricting the diversity of the scenarios. For soccer domain, Stone and et al (2006) presented one benchmark scenario on keepaway tactical scenario and later extended to more general half-field offense scenario (Hausknecht and et al 2016) and provided a library of APIs relating to behaviors (e.g. mark player, defend goal) of players, which helps users to model scenarios. However, SCENIC provides further benefits that are not covered in this work. SCENIC provides high-level syntax and semantics to (i) easily write spatial relations for intutively modeling initial states, (ii) assign distributions over both initial states and behaviors to generate variations of environments for robust training and testing generalization, and (iii) specify priorities over interaction conditions over behaviors to model more sophisticated types of higher level behavior (for more detail refer to Section *Modeling Scenarios with* SCENIC).

Formal Scenario Specification Languages for Environment Modeling and Generation A few scenario specification languages have been proposed in the autonomous driving domain including SCENIC. Paracosm language (Majumdar et al. 2019) models dynamic scenarios with reactive and synchronous model of computation. The Measurable Scenario Description Language (M-SDL) (Foretellix 2020) shares common features as SCENIC to model interactive scenarios. In contrast, however, SCENIC provides a much higher-level, probabilistic, declarative way of modeling. Furthermore, unlike other scenario specification languages, SCENIC has demonstrated its generality over different domains such as autonomous driving, robotics, and aviation (Fremont, Kim, and et al 2020). For these reasons, we chose SCENIC in this paper for demonstration of benefits that a scenario specification language can provide to RL.

3 Background

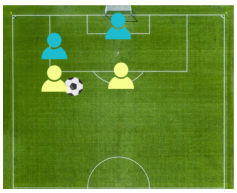
3.1 Google Research Football Simulator

The Google Research Football (GRF) simulator (Kurach et al. 2020) provides a realistic soccer environment to train and test RL agents. The setting, the rules, and the objective of the environment are the same as defined by Fédération Internationale de Football Association (FIFA 2021). The environment setup is as the following. All the players on the field are controlled by (1) GRF's built-in, rule-based AI bots and (2) RL agents. The simulator dynamically determines which of the RL team players are to be controlled by RL agents based on their vicinity to the ball. GRF provides 11 offense scenarios to train and test RL agent performance and it provides trained RL agent checkpoints for a subset of its scenarios.

3.2 Scenario Specification Language: SCENIC

SCENIC (Fremont, Dreossi, and et al 2019; Fremont, Kim, and et al 2020) is an object-oriented, probabilistic programming language whose syntax and semantics are designed to intuitively *model* and *generate* scenarios. A SCENIC program represents an abstract scenario, which models a *distribution* over initial states and behaviors of players in the scenario. For each scenario generation, an initial state is sampled from the program at the beginning of a simulation and interactive behaviors are sampled during simulation runtime. Therefore, with a single SCENIC program, users can generate a distribution of concrete scenarios.

SCENIC requires action and model libraries, which are imported and compiled with a user's SCENIC program for execution. The action library defines the action space which is determined by the simulator. The model library defines objects and their attributes (e.g. position, heading). We can assign prior distributions over these attributes. For example, a goalkeeper's position can be uniformly randomly distributed over the penalty box region. If a user simply instantiates a goalkeeper in a SCENIC program but does not specify any condition over its attributes, then they are sampled from the prior distributions by default. These prior distributions can be overwritten in the user's SCENIC program.



(a) a bird-eye view of the scenario

(b) a snapshot of GRF environment

```
builder.SetBallPosition(0.7, -0.28)
builder.SetTeam(Team.e_Left)
builder.AddPlayer(-1.0, 0.0, e_PlayerRole_GK)
builder.AddPlayer(0.7, 0.0, e_PlayerRole_CB)
builder.AddPlayer(0.7, -0.3, e_PlayerRole_CB)
builder.SetTeam(Team.e_Right)
builder.AddPlayer(-1.0, 0.0, e_PlayerRole_GK)
builder.AddPlayer(-0.75, 0.1, e_PlayerRole_CB)
```

(c) GRF's scenario program

```
behavior egoBehavior(player, destinationPoint):
        do Uniform(ShortPassTo(player), dribbleToAndShoot(destinationPoint))

behavior attackMidBehavior()
  try:
    do HoldPosition()
  interrupt when self.owns_ball is True:
    do AimGoalCornerAndShoot()
  interrupt when (distance to nearestOpponentPlayer(self)) < 5:
    do dribble_evasive_zigzag()

LeftGK
attack_midfielder = LeftAM on left_penalty_arc_region,
                facing north,
                with behavior attackMidBehavior()
ego = LeftLM ahead of attack_midfielder by Range(5, 10),
    facing toward attack_midfielder,
    with behavior egoBehavior(attack_midfielder, left_penaltyBox_center)

Ball ahead of ego by 0.1
RightRB left of ego by Range(3,5)
RightGK
```

(d) SCENIC program of generalized pass-and-shoot scenario with distribution over players' initial condition and behaviors

Figure 1: Programs encoding the Google Research Football's (GRF) pass-and-shoot scenario

4 Scenario Specification Language for RL

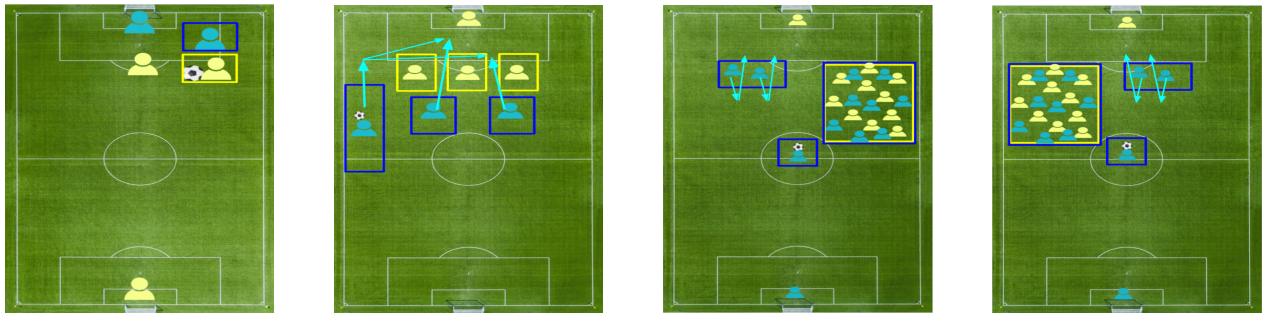
4.1 Benefits of Scenario Specification Language for RL

The objective of this paper is to introduce the benefits of the use of scenario specification language for modeling and generating scenarios, specifically for RTS environments for RL. Using a scenario specification language whose syntax and semantics are carefully designed to intuitively model scenarios have the following benefits:

1. **Easily Model Interactive Environments on *User-demand* to Train and Test RL Agents:** The intuitive syntax and semantics, which abstracts away the implementation details and allows users to reason solely at high-level semantics, makes it easy to model complex spatial relations among multiple agents, their behaviors and conditions on how these behaviors should interact. It should be noted that, it requires a considerable amount of research and engineering effort to design and implement a formal scenario modeling language and its compiler from scratch.
2. **Program Stochastic Policies:** These programmed agents can serve two purposes: (i) allow developers to incorporate domain knowledge, e.g., generate demonstration data for offline training and (ii) provide performance baseline for trained RL agents.
3. **Interpretability and Transparency**: The intuitive syntax and semantics make scenario programs interpretable and transparent. Therefore, users can reason about the difference/similarity of train and test environments by comparing their scenario programs.
4. **Reusability of Existing Scenarios**: The interpretability of scenario programs facilitates easy modification or reuse of existing SCENIC programs, models, and behaviors to quickly model new scenarios. This facilitates building a community around designing and sharing scenario programs, by building upon each other's scenarios.

4.2 Modeling Scenarios with SCENIC

Formally, a scenario is a Markov Decision Process (MDPs) (Sutton and Barto 2018) defined as a tuple $(\mathcal{S}, \mathcal{A}, p, r, \rho_0)$, with $\mathcal{S}$ denoting 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. Given the



(a) generalization test scenario for the scenario in Fig. 1a

(b) 3 vs 3 left mid-fielder crosses to either player in penalty box

(c) 11 vs 11 open player scenario

(d) mirrored Fig. 2c scenario

Figure 2: Examples of a new defense scenarios with specific assigned behaviors (a), a test scenario to assess generalization (b), and two full game scenarios (c,d) we used for training and testing. The RL team is yellow and the opponent, blue. The assigned opponent behaviors are highlighted with light blue arrows. Uniformly random distribution is assigned over a specific region for each player. These regions are highlighted boxes.

```
1  behavior receiveCrossAndShoot(destinationPoint):
2      do MoveToPosition(destinationPoint)
3      do HoldPosition() until self.owns_ball
4      do dribbleToAndShoot(yellowPenaltyBox_centerPoint)
5      do HoldPosition()
6
7  behavior crossToPlayers(list_of_players):
8      destinationPoint = Point on region_in_penaltyBox
9      do MoveToPosition(destinationPoint)
10     do HighPassTo(Uniform(*list_of_players))
11     do HoldPosition()
```

Figure 3: A snippet of a SCENIC program specifying behaviors for players Fig. 2b

state and action spaces as defined by the GRF environment, a SCENIC program defines (i) the initial state distribution, (ii) the transition dynamics (specifically players' behaviors), and (iii) the reward function. Hence, users can exercise extensive control over the environment with SCENIC.

Modeling Initial State Distribution Users can intuitively specify initial state distributions with SCENIC's high-level syntax that resembles natural English. For example, refer to the full SCENIC program in Fig. 1(d) which describes a more generalized version of GRF's Pass and Shoot scenario as visualized in Fig. 1(a,b). In line 12-22, the initial state distribution is specified. The SCENIC syntax for modeling spatial relations among players are highlighted in yellow. In addition, SCENIC supports about 20 different syntax to support modeling complex spatial relations (Fremont, Kim, and et al 2020). Rather than having to hand-code positions for a concrete scenario as in the GRF's scenario 1(c), users can much more intuitively and concisely model a distribution of initial states. Here, `Left` represents the yellow team, `Right` the blue, and the two following abbreviated capital letters indicate the player role.

Modeling Transition Dynamics One can flexibly modify transition dynamics of the environment by specifying the behaviors of non-RL players using SCENIC. Take the same example SCENIC program in Fig. 1(d) as above. Line 1-10 models two new behaviors. A behavior can invoke another behavior(s) with syntax `do`, succinctly modeling a behavior in a hierarchical manner. Users can assign distribution over behaviors as in line 2. The interactive conditions are specified using try/interrupt block as in line 5-10. Semantically, the behavior specified in the try block is executed by default. However, if any interrupt condition is satisfied, then the default behavior is paused and the behavior in the interrupt block is executed until completion and then the default behavior resumes. These interrupts can be nested with interrupt below has higher priority. In such case, the same semantics is consistently applied.

Rewards SCENIC has a construct called `monitor`, which can be used to specify reward functions. The reward conditions in the `monitor` is checked at every simulation step and updates the reward accordingly.

Termination Conditions Users can also specify termination conditions which are monitored at every simulation time step.

4.3 On Interfacing SCENIC to a Simulator

Interfacing SCENIC to other simulators is straight-forward. In fact, SCENIC is already interfaced with five other simulators (Daniel Fremont 2021) in domains such as autonomous driving, aviation, and robotics. To interface SCENIC with a simulator, one needs define the model, action, and behavior libraries. These libraries expedites modeling complex scenarios by helping users re-use the set of models, actions, and behaviors in the libraries, rather than having to write a scenario from scratch.

The *model* library defines the state space. It defines players with distribution over their initial state according to their roles and GRF's AI bot is assigned by default to all player behavior. These prior distribution over the initial state and behavior can be overwritten in the SCENIC program. The model library also defines region objects such as goal and penalty box regions as well as directional objects in compass directions. The *action* library defines the action space as determined by the GRF simulator. These action space

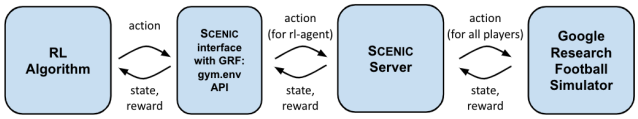


Figure 4: Interface Architecture between SCENIC and GRF

consists of movement actions in eight compass directions, long/short/high pass, shoot, slide, dribble, and sprint.

The *behavior* library consists of behaviors and helper functions that represent widely used basic skills in soccer. These behaviors include give-and-go, evasive zigzag dribble to avoid an opponent's ball interception, dribbling to a designated point and shooting, shooting towards the left or right corner of the goal, etc. Additionally, the behavior library also include useful helper functions such as identifying nearest opponent or teammate, whether there is an opponent near the running direction of a dribbler, etc. Please refer to our open-sourced repository for more details.

Interface Architecture Figure 4 shows an overview of our overall architecture. The architecture can be divided into two parts: i) RL interface, through which the RL algorithms interact with SCENIC and ii) the SCENIC Server, which executes a SCENIC program and governs the simulation by interacting with the underlying simulator. We follow the widely used OpenAI Gym API (Brockman et al. 2016) as our interface, which allows our interface to be used seamlessly with all the existing standard RL frameworks.

For each simulation/episode, the SCENIC server first samples an initial state from the SCENIC program to start a new scenario in the GRF simulator and updates its internal model of the world (e.g., player and ball positions). From then on, a round of communication occur between the RL algorithm and SCENIC server, with the RL interface at the middle. At each timestep, the gym interface takes in the action(s) for the RL agent and passes them to the SCENIC server. The SCENIC server in turn computes actions for all the remaining non-RL players—the players not controlled by the RL agent—and then executes all these actions (of both the RL and non-RL players) in the simulator. The SCENIC server then receives the observation and reward from the simulator, updates the internal world state, and then passes them back to the RL algorithm. This interaction goes on till any terminating conditions as specified in the scenario script is satisfied.

5 Evaluation

In this section, we demonstrate four use cases of SCENIC in RL. First, we present and benchmark a set of 13 realistic mini-game scenarios encoded in SCENIC with a varying level of difficulty. Second, we test the generalization capabilities of the trained RL agents on unseen, yet intuitively similar scenarios. Next, we show how developers can "debug" their agents for failure scenarios of interest. At last, we show how probabilistic SCENIC policies can be used to generate offline data and endow domain knowledge into the learning process for faster training, which we believe to be very important for applying RL in practice.

Experimental Setup

We run PPO (Schulman et al. 2017) on a single GPU machine (NVIDIA T4) with 16 parallel workers on Amazon AWS. Unless otherwise specified, all the PPO training are run for 5M timesteps and repeated for 10 different seeds. All the evaluation has been done for 10000 timesteps. For all the experiments, we use the stacked Super Mini Map representation for observations —a 4x72x96 binary matrix representing positions of players from both team, the ball, and the active player—and the scores as rewards, i.e., $+1$ when scoring a goal and -1 upon conceding, from (Kurach et al. 2020). Similar to the academy scenarios from (Kurach et al. 2020), we also terminate a game when one of the following happens: either of the team scores, ball goes out of the field, or, the ball possession changes. For further details, including hyperparameters and network architecture, we refer readers to the Supplementary Materials (Section *Details on Experimental Setup and Training*).

5.1 Mini-game Scenario Benchmark

Training an RL agent to solve a full soccer game involving 22 players is very challenging and may take days even with distributed algorithms. For example, Kurach et al. (2020) showed even the easy version of GRF's 11 vs 11 game cannot be solved with 50M samples. To allow researchers to iterate their ideas with a reasonable amount of time and compute, we present a set of 13 mini-game scenarios. All these scenarios are inspired from common situations occurring in real soccer games but involves fewer number of players to make them amenable to be faster training.

Nine of our proposed mini-game scenarios are defense scenarios, which are nice complement to GRF's offense-only scenarios (refer to Sec. 3.1), along with four new offense scenarios. Most of these scenarios are initialized from a distribution, rather than fixed locations. By default all the opponent players are controlled by GRF's built-in AI bot (refer to Sec. 3.1). However, for the scenarios where the AI bot does not exhibit our desired behavior, we model the opponent behaviors using SCENIC. For example, in the 3vs3 cross scenario as shown in Fig. 2b, the opponent AI bots tried to pass the ball around instead of crossing. Therefore, we modelled and assigned behaviors such that the blue player on the leftmost side of the field would run up the field and cross the ball. Meanwhile, the two blue players in the center run into the penalty box area to receive the cross and shoot. These modelled behaviors are shown in Fig. 3.

We benchmark our mini-game scenarios by training agents with PPO. Figure 5 shows the average goal differences for all the scenarios. For these mini-game scenarios, we end the game if one of the teams score. Hence, the goal difference can range between -1 to +1. For the offense scenarios, a well trained agent is supposed to score consistently achieving an average goal difference close to +1. On the other hand, a well-trained agent should achieve a goal difference close to 0 for successfully defending the opponents in the defense scenarios. From the graph it can be seen that

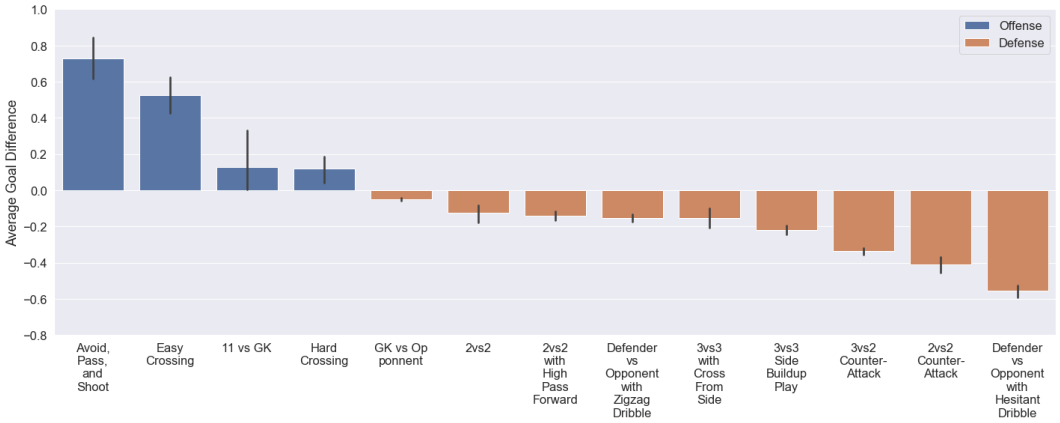


Figure 5: Average Goal Difference of PPO agents on the proposed mini-game scenario benchmark. The error bars represent 95% bootstrapped confidence intervals

the proposed scenarios offer a varied levels of difficulties. For example, PPO consistently achieves goal difference of around 0.5 for the EASY CROSSING scenario, but barely learns anything for HARD CROSSING. In case of the defense scenarios, the results also show a varied range of difficulty, GK VS OPPONENT scenario being be easiest.

5.2 Testing for Generalization

We provide scripts to test generalization of all of our 13 new benchmark scenarios along with 5 scenarios provided by GRF. We changed the distribution over the initial state while keeping the formation of players and their behaviors in each scenario intact. For example, for testing generalization of an RL agent trained in the Pass and Shoot scenario (Fig. 1a), we instantiated the yellow and the blue players on the symmetric right side of the field instead of the left and kept the other initial state distribution the same (Figure 2a).

Fig. 6 compares the trained agents' performance in training and test scenarios. As expected, we observe a noticeable drop of performance in most of the GRF's academy and offense scenarios (Fig. 6a). For example, the Pass and Shoot scenario (Figure 1a), which achieved around 0.6 in training, failed to generalize for the test scenario. However, for the defense scenarios, the drop in performance was not as noticeable. We conjecture that this distinction comes from the differences in the offense and defense training scenarios, where the defense scenarios tend to contain larger distribution over the initial state than those of the offense scenarios (refer to Supplement). Consequently, larger variations of scenarios introduced during training may have contributed to better generalization for defense scenarios.

5.3 Debugging Agents on 11v11 Failure Scenario

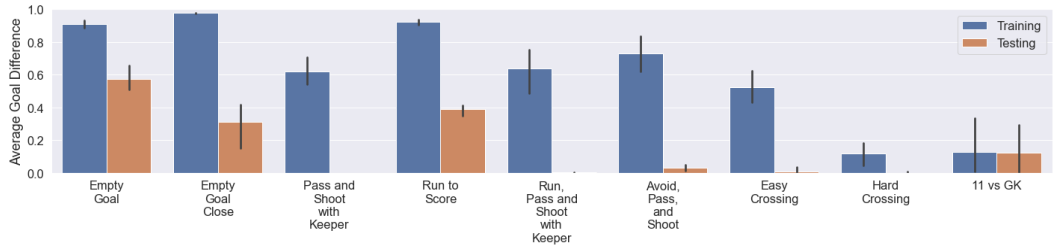
For this experiment, we evaluate and debug an RL checkpoint provided by GRF, which was trained on their 11 vs 11 easy stochastic scenario, i.e., easy version of their full-game scenario. This agent achieves an impressive average goal difference of 6.99 per full-game[1], scoring up to 14 goals in the training scenario during our experiments. We modelled a scenario, as visualized in Figure 2c, to test the agent's ability to quickly perceive open teammates near the opponent goal to advance the ball forward and score—a crucial skill for soccer. When we assigned GRF's built-in AI bots to control the open players on the left side of the field, the players ran straight toward the ball, instead of taking advantage of the closeness to the opponent goal without being marked. Hence, we modelled a behavior for open players in SCENIC so that they would stay close to the goal while abiding by the offside rule.

Although obvious to humans, the trained checkpoint performs poorly in this scenario with an average goal difference of 0.1. To 'debug' the agent, we then fine-tune the agent on a 'mirrored' scenario, as shown in Figure 2d, with PPO for 5M timesteps. The fine-tuned agent improved noticeably on the original scenario, achieving an average goal difference of 0.67. This showcases the usefulness of SCENIC to easily model and generate scenarios of interest using one's domain knowledge, which may have been difficult with blackbox agents (e.g. built-in AI bots, or trained RL agents), to test and debug certain capabilities of an RL agent.

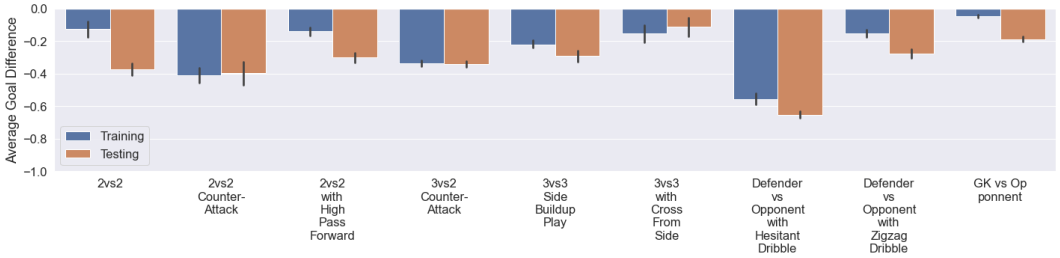
5.4 Facilitating Training with Probabilistic SCENIC Policies

In the section, we show how RL practitioners can incorporate their domain knowledge by writing probabilistic SCENIC policies for faster training. We wrote simple semi-expert RL policies for five different scenarios, where the agent suffers to learn, and generated 8K samples of demonstration data for per scenario. To facilitate training on those scenarios, we first pre-train an agent via behavior cloning with the generated offline data and then fine-tune the agent using PPO for 5M timesteps. All the experiments were repeated for three different seeds. Figure 7 compares the training performance of these agents against the agents that were trained with PPO only. We notice that, even with such a low volume of demonstration data, we can train much better agents and can solve scenarios which were otherwise un-

[1]Evaluated on 100K timesteps



(a) Offense and select GRF academy scenarios



(b) Defense scenarios

Figure 6: Evaluation of PPO agents' generalization against varying initial conditions. For most of the academy and offense scenarios we observe a significant drop in performance. However, for several defense scenarios the difference in train and test scenarios is not that significant.

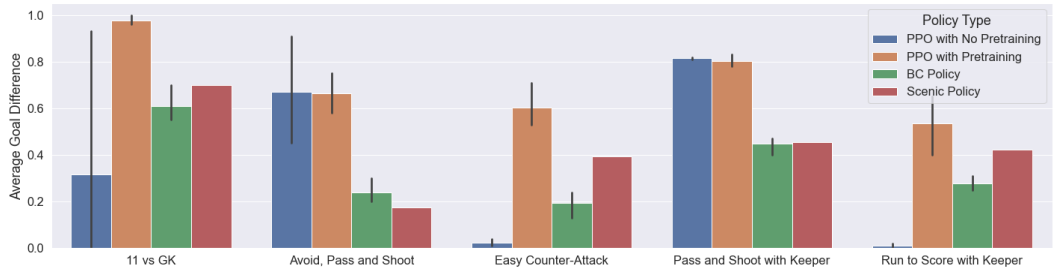


Figure 7: Performance of PPO agents trained with and without any demonstration data, along with the performance of corresponding behavior-cloned and SCENIC policies. We see significantly better performance on three of the scenarios, while the rest two achieves comparable performance, highlighting the usefulness of the proposed SCENIC policies.

solved. The experimental results thus suggests, with stochastic SCENIC policies we can generate rich quality demonstration data to substantially enhance training performance, which can be particularly useful in practice for environments like GRF which requires a heavy compute resource.

6 Conclusion & Future works

We introduced and demonstrated the benefits of adopting a scenario specification language to train RL agents and test their generalization capabilities in various realistic scenarios – spanning from mini-games to full games – generated by SCENIC programs, which succinctly capture distributions of initial states and behaviors. We also showcased modeling domain knowledge via stochastic SCENIC policies by generating demonstration data to facilitate training in GRF, a complex real-time strategy environment. We hope our work could gather an interest to support systematic modeling of RTS environments.

Limitations and future directions. There are a number of interesting future directions that we plan to pursue.

- Interfacing other simulators: Currently, our platform only supports the Google Soccer Environment for RL. In future we plan to support more simulators.
- Our addition of a layer over the base GRF simulator also adds an overhead, which we plan to improve in future. Please refer to the Supplementary Materials for detailed experiment on performance.
- Multi-Agent Learning: While our platform does not restrict us to only single-agent RL settings, our current training experiments/benchmark do not include multi-agent training. In future, we would like to introduce scenarios and benchmarks for multi-agent setting.
- Multi-Task Learning: Currently we only train RL agents on a single scenario. However, as many soccer skills are transferable, in future we want to explore simultaneously solving multiple-scenarios in a multi-task setting.

Acknowledgements

This work was supported in part by National Science Foundation grants CNS-1730628, CNS-1545126 (VeHICaL), CNS-1739816, and CCF-1837132, by DARPA contracts FA8750-16-C0043 (Assured Autonomy) and FA8750-20-C-0156 (Symbiotic Design of Cyber-Physical Systems), by Berkeley Deep Drive, by Toyota through the iCyPhy center, by the Toyota Research Institute, and by the Berkeley Artificial Intelligence Research (BAIR) Commons program.

References

Bellemare, M. G.; Naddaf, Y.; Veness, J.; and Bowling, M. 2013. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47: 253–279.

Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; and Zaremba, W. 2016. Openai gym. *arXiv preprint arXiv:1606.01540*.

Cobbe, K.; Hesse, C.; Hilton, J.; and Schulman, J. 2020. Leveraging procedural generation to benchmark reinforcement learning. In *International conference on machine learning*.

Cobbe, K.; Klimov, O.; Hesse, C.; Kim, T.; and Schulman, J. 2019. Quantifying generalization in reinforcement learning. In *International Conference on Machine Learning*.

Daniel Fremont, E. K. 2021. SCENIC interfaced simulators. https://scenic-lang.readthedocs.io/en/latest/simulators.html.

Duan, Y.; Chen, X.; Houthooft, R.; Schulman, J.; and Abbeel, P. 2016. Benchmarking deep reinforcement learning for continuous control. In *International conference on machine learning*.

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

Foretellix. 2020. Measurable Scenario Description Language. https://www.foretellix.com/wp-content/uploads/2020/07/M-SDL_LRM_OS.pdf.

Fremont, D.; Dreossi, T.; and et al. 2019. Scenic: a language for scenario specification and scene generation. In *PLDI*, 63–78. ACM.

Fremont, D.; Kim, E.; and et al. 2020. Scenic: A Language for Scenario Specification and Data Generation. *CoRR*, abs/2010.06580.

Hausknecht, M.; and et al. 2016. Half Field Offense: An Environment for Multiagent Learning and Ad Hoc Teamwork. In *AAMAS Adaptive Learning Agents (ALA) Workshop*.

Hendrikx, M.; and et al. 2013. Procedural content generation for games: A survey. In *ACM Transactions on Multimedia Computing, Communications, and Applications*, volume 9.

Justesen, N.; Torrado, R. R.; Bontrager, P.; Khalifa, A.; Togelius, J.; and Risi, S. 2018. Illuminating generalization in deep reinforcement learning through procedural level generation. *arXiv preprint arXiv:1806.10729*.

Kalashnikov, D.; Irpan, A.; Pastor, P.; Ibarz, J.; Herzog, A.; Jang, E.; Quillen, D.; Holly, E.; Kalakrishnan, M.; Vanhoucke, V.; et al. 2018. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. In *Conference on Robot Learning*.

Kurach, K.; Raichuk, A.; Stańczyk, P.; Zajac, M.; Bachem, O.; Espeholt, L.; Riquelme, C.; Vincent, D.; Michalski, M.; Bousquet, O.; et al. 2020. Google research football: A novel reinforcement learning environment. In *AAAI Conference on Artificial Intelligence*.

Lee, K.; Lee, K.; Shin, J.; and Lee, H. 2020a. Network randomization: A simple technique for generalization in deep reinforcement learning. In *International Conference on Learning Representations*.

Lee, K.; Seo, Y.; Lee, S.; Lee, H.; and Shin, J. 2020b. Context-aware dynamics model for generalization in model-based reinforcement learning. In *International Conference on Machine Learning*.

Machado, M. C.; Bellemare, M. G.; Talvitie, E.; Veness, J.; Hausknecht, M.; and Bowling, M. 2018. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, 61: 523–562.

Majumdar, R.; Mathur, A.; Pirron, M.; Stegner, L.; and Zufferey, D. 2019. Paracosm: A Language and Tool for Testing Autonomous Driving Systems. arXiv:1902.01084.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *Nature*, 518(7540): 529.

Nichol, A.; Pfau, V.; Hesse, C.; Klimov, O.; and Schulman, J. 2018. Gotta learn fast: A new benchmark for generalization in rl. *arXiv preprint arXiv:1804.03720*.

OpenAI; Berner, C.; Brockman, G.; Chan, B.; Cheung, V.; Debiak, P.; Dennison, C.; Farhi, D.; Fischer, Q.; Hashme, S.; Hesse, C.; Józefowicz, R.; Gray, S.; Olsson, C.; Pachocki, J.; Petrov, M.; d. O. Pinto, H. P.; Raiman, J.; Salimans, T.; Schlatter, J.; Schneider, J.; Sidor, S.; Sutskever, I.; Tang, J.; Wolski, F.; and Zhang, S. 2019. Dota 2 with Large Scale Deep Reinforcement Learning. arXiv:1912.06680.

Samvelyan, M.; and et al. 2019. The StarCraft Multi-Agent Challenge. In *Workshop on Deep Reinforcement Learning at the 33rd Conference on Neural Information Processing Systems*.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Seo, Y.; Lee, K.; Clavera, I.; Kurutach, T.; Shin, J.; and Abbeel, P. 2020. Trajectory-wise Multiple Choice Learning for Dynamics Generalization in Reinforcement Learning. *arXiv preprint arXiv:2010.13303*.

Silver, D.; Hubert, T.; Schrittwieser, J.; Antonoglou, I.; Lai, M.; Guez, A.; Lanctot, M.; Sifre, L.; Kumaran, D.; Graepel, T.; et al. 2018. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362(6419): 1140–1144.

Silver, D.; Schrittwieser, J.; Simonyan, K.; Antonoglou, I.; Huang, A.; Guez, A.; Hubert, T.; Baker, L.; Lai, M.; Bolton,

A.; et al. 2017. Mastering the game of go without human knowledge. *Nature*, 550(7676): 354.

Stone, P.; and et al. 2006. Keepaway Soccer: From Machine Learning Testbed to Benchmark. In *RoboCup 2005: Robot Soccer World Cup IX*, 93–105. Berlin, Heidelberg: Springer Berlin Heidelberg.

Sutton, R. S.; and Barto, A. G. 2018. *Reinforcement learning: An introduction*. MIT Press.

Tassa, Y.; Doron, Y.; Muldal, A.; Erez, T.; Li, Y.; Casas, D. d. L.; Budden, D.; Abdolmaleki, A.; Merel, J.; Lefrancq, A.; et al. 2018. Deepmind control suite. *arXiv preprint arXiv:1801.00690*.

Uriarte, A.; and Ontañón, S. 2021. A Benchmark for StarCraft Intelligent Agents. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 11, 22–28.

Vinyals, O.; Babuschkin, I.; Czarnecki, W. M.; Mathieu, M.; Dudzik, A.; Chung, J.; Choi, D. H.; Powell, R.; Ewalds, T.; Georgiev, P.; et al. 2019. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575(7782): 350–354.

Vinyals, O.; Ewalds, T.; Bartunov, S.; Georgiev, P.; Vezhnevets, A. S.; Yeo, M.; Makhzani, A.; Küttler, H.; Agapiou, J.; Schrittwieser, J.; Quan, J.; Gaffney, S.; Petersen, S.; Simonyan, K.; Schaul, T.; van Hasselt, H.; Silver, D.; Lillicrap, T.; Calderone, K.; Keet, P.; Brunasso, A.; Lawrence, D.; Ekermo, A.; Repp, J.; and Tsing, R. 2017. StarCraft II: A New Challenge for Reinforcement Learning. arXiv:1708.04782.

Zhang, A.; Wu, Y.; and Pineau, J. 2018. Natural environment benchmarks for reinforcement learning. *arXiv preprint arXiv:1811.06032*.