On the Verge of Solving Rocket League using Deep Reinforcement Learning and Sim-to-sim Transfer

Marco Pleines, Konstantin Ramthun, Yannik Wegener, Hendrik Meyer, Matthias Pallasch, Sebastian Prior
Jannik Drögemüller, Leon Büttinghaus, Thilo Röthemeyer, Alexander Kaschwig,
Oliver Chmurzynski, Frederik Rohkrähmer, Roman Kalkreuth, Frank Zimmer*, Mike Preuss†
Department of Computer Science, *TU Dortmund University*, Dortmund, Germany
**Department of Communication and Environment, Rhine-Waal University of Applied Sciences*, Kamp-Linfort, Germany
†*LIACS Universiteit Leiden*, Leiden, Netherlands

***Abstract*—Autonomously trained agents that are supposed to play video games reasonably well rely either on fast simulation speeds or heavy parallelization across thousands of machines running concurrently. This work explores a third way that is established in robotics, namely sim-to-real transfer, or if the game is considered a simulation itself, sim-to-sim transfer. In the case of Rocket League, we demonstrate that single behaviors of goalies and strikers can be successfully learned using Deep Reinforcement Learning in the simulation environment and transferred back to the original game. Although the implemented training simulation is to some extent inaccurate, the goalkeeping agent saves nearly 100% of its faced shots once transferred, while the striking agent scores in about 75% of cases. Therefore, the trained agent is robust enough and able to generalize to the target domain of Rocket League.**

***Index Terms*—rocket league, sim-to-sim transfer, deep reinforcement learning, proximal policy optimization**

I. Introduction

The spectacular successes of agents playing considerably difficult games, such as StarCraft II [1] and DotA 2 [2], have been possible only because the employed algorithms were able to train on huge numbers of games on the order of billions or more. Unfortunately, and despite many improvements achieved in AI in recent years, the utilized Deep Learning methods are still relatively sample inefficient. To deal with this problem, fast running environments or high amounts of computing resources are vital. OpenAI Five for DotA 2 [2] is an example of the utilization of hundreds of thousands of computing cores in order to achieve high throughput in terms of played games. However, this way is closed for games that run only on specific platforms and are thus very hard to parallelize. Moreover, not many research groups have such resources at their disposal. Video games that suffer from not being able to be sped up significantly, risk minimal running times and hence repeatability. Therefore it makes sense to look for alternative ways to tackle difficult problems.

Sim-to-real transfer offers such an alternative way and is well established in robotics, and it follows the general idea that robot behavior can be learned in a very simplified simulation environment and the trained agents can then be successfully transferred to the original environment. If the target platform is a game as well, we may speak of sim-to-sim transfer because the original game is also virtual, just computationally much more costly. This approach is applicable to current games, even if they are not parallelizable, and makes them available for modern Deep Reinforcement Learning (DRL) methods. There is of course a downside of this approach, namely that it may be difficult or even infeasible to establish a simulation that is similar enough to enable transfer later on, but still simple enough to speed up learning significantly. A considerable amount of effort has to be invested in establishing this simulation environment before we can make any progress on the learning task.

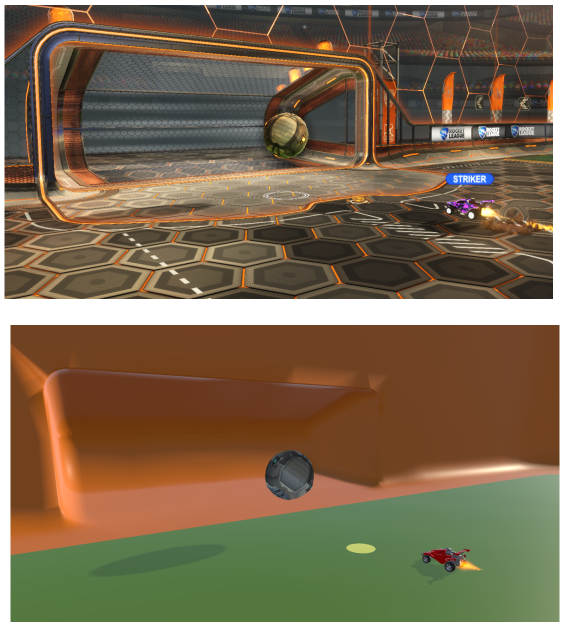

Fig. 1. The game of Rocket League (top) and the contributed simulation (bottom), which notably advances its ancestor project *RoboLeague* [3].

To our knowledge, the sim-to-sim approach has not yet been applied to train agents for a recent game. Therefore we aim to explore the possibilities of this direction in order to detect how simple the simulation can be, and how good the transfer to the original game works.

The game we choose as a test case of the sim-to-sim approach is Rocket League (Figure 1), which basically resembles indoor football with cars and teams of 3. Rocket league is freely available for Windows and Mac, possesses a bot API

(RLBot [4]) and a community of bot developers next to a large human player base. As the 3 members of each team control car avatars with physical properties different from human runners, the overall tactics are the one of rotation without fixed roles. Thereby, large parts of the current speed can be conserved and players do not have to accelerate from zero when ball possession changes [5]. Next to basic abilities attempting to shoot towards the goal and to move the goalie in order to prevent a goal, Rocket League is a minimal *team AI* setting [6] where layers of team tactics and strategy can be learned.

The first step of our work re-implements not all, but multiple physical gameplay mechanics of Rocket League using the game engine Unity, which results in a slightly inaccurate simulation. We then train an agent in a relatively easy goalie and striker environment using Proximal Policy Optimization (PPO) [7]. The learned behaviors are then transferred to Rocket League for evaluation. Even though the training simulation is imperfect, the transferred behaviors are robust enough to succeed at their tasks by generalizing to the domain of Rocket League. The goalkeeping agent saves nearly 100% of the shots faced, while the striking agent scores about 75% of its shots. The sim-to-sim transfer is further examined by ablating physical adaptations that were added to the training simulation.

This paper proceeds with elaborating on related work. Then, the physical gameplay mechanics of Rocket League are shown. After illustrating the trained goalie and striker environment, PPO and algorithmic details are presented. Section V examines the sim-to-sim transfer. Before concluding our work, a discussion is provided.

II. Related Work

Sim-to-sim transfer on a popular multiplayer team video game touches majorly on two different areas, namely multi-agent and sim-to-real transfer. DotA 2 and StarCraft II are the already mentioned prominent examples in the field of multi-agent environments. As this work focuses on single-agent environments, namely the goalkeeper and striker environments, related work on sim-to-real transfer is focused next.

Given the real world, a considered prime example for multi-agent scenarios is *RoboCup*. RoboCup is an annual international competition [8] that offers a publicly effective open challenge for the intersection of robotics and AI research. The competition is known for the robot soccer cup but also includes other challenges. Reinforcement Learning (RL) has been successfully applied to simulated robot soccer in the past [9] and has been found a powerful method for tackling robot soccer. A recent survey [10] provides insights into robot soccer and highlights significant trends, which briefly mention the transfer from simulation to the real world.

In general, sim-to-real transfer is a well-established method for robot learning and is widely used in combination with RL. It allows the transition of an RL agent's behavior, which has been trained in simulations, to real-world environments. Sim-to-real transfer has been predominantly applied to RL-based robotics [11] where the robotic agent has been trained with state-of-the-art RL techniques like PPO [7]. Popular applications for sim-to-real transfer in robotics have been autonomous racing [12], Robot Soccer [13], navigation [14], and control tasks [15]. To address the inability to exactly match the real-world environment, a challenge commonly known as sim-to-real gap, steps have also been taken towards generalized sim-to-real transfer for robot learning [16], [17]. The translation of synthetic images to realistic ones at the pixel level is employed by a method called GraspGAN [18] which utilizes a generative adversarial network (GAN) [19]. GANs are able to generate synthetic data with good generalization ability. This property can be used for image synthesis to model the transformation between simulated and real images. GraspGAN provides a method called *pixel-level domain adaptation*, which translates synthetic images to realistic ones at the pixel level. The synthesized pseudo-real images correct the sim-to-real gap to some extent. Overall, it has been found that the respective policies learned with simulations execute more successfully on real robots when GraspGAN is used [18].

Another approach to narrow the sim-to-real gap is domain randomization [20]. Its goal is to train the agent in plenty of randomized domains to generalize to the real domain. By randomizing all physical properties and visual appearances during training in the simulation, a trained behavior was successfully transferred to the real world to solve the Rubik's cube [21].

III. Rocket League Environment

This section starts out by providing an overview of vital components of Rocket League's physical gameplay mechanics, which are implemented in the training simulation based on the game engine Unity and the ML-Agents Toolkit [22]. RLBot [4] provides the interface to Rocket League where the training situations can be reproduced. Afterward, the DRL environments, designated for training, and their properties are detailed. The code is open source[1].

A. Implementation of the Training Simulation

The implementation of the Unity simulation originates from the so called *RoboLeague* repository [3]. As this version of the simulation is by far incomplete and inaccurate, multiple fundamental aspects and concepts are implemented, which are essentially based on the physical specifications of Rocket League. These comprise, for example, the velocity and acceleration of the ball and the car, as well as the concept of boosting. Jumps, double jumps as well as dodge rolls are now possible, and also collisions and interactions. There is friction caused by the interaction of a car with the ground, but also friction caused by the air is taken into account.

However, further adjustments are necessary. Therefore, table I provides an overview of all the material that was considered during implementing essential physical components, while highlighting distinct adjustments that differ from the information provided by the references. It has to be noted

[1]https://github.com/PG642

TABLE I

OVERVIEW ON ESSENTIAL PHYSICAL GAMEPLAY MECHANICS PRESENT IN ROCKET LEAGUE, WHICH ARE ADDED TO THE TRAINING SIMULATION.

Physics Component	**Sources**	**Additional Information and Different Parameters**
Entity Measures (e.g. Arena)	[3], [4]	Car model Octane and its collision mesh is used Radius of the ball is set to 93.15uu (value in Rocket League 92.75uu)
Car: Velocity, Acceleration, Boost	[23]	No modifications done
Car: Jumps, Double Jumps, Dodge Rolls	[4], [24]	Raise max. angular velocity during dodge from $5.5\frac{\text{rad}}{\text{s}}$ to $7.3\frac{\text{rad}}{\text{s}}$
Car: Air Control	[23]	Adjust drag coefficients for roll to -4.75 and pitch to -2.85
Bullet and Psyonix Impulse	[23], [25]	Used for the ball-to-car interaction and car-to-car interaction. The impulse by the bullet engine replaces the Unity one. Psyonix impulse is an additional impulse on the center of the ball, which allows a better prediction and control of collisions.
Ball Bouncing	[23]	Within the bounce's computation a ball radius of 91.25uu is considered.
Friction (Air, Ground) and Drifting	[25]	A drag of $-525\frac{uu}{s^2}$ is used, which is reduced by more than half when the car is upside down.
Ground Stabilization	[4], [25]	The stabilization torque is denoted by an acceleration of $50\frac{\text{rad}}{s^2}$.
Wall Stabilization	[4]	Raise sticky forces for wall stabilization to an acceleration of $500\frac{uu}{s^2}$
Suspension	[26] [27]	Stiffness of front wheels: $163.9\frac{1}{s^2}$ and of back wheels: $275.4\frac{1}{s^2}$ Damper front and back is set to $30\frac{1}{s}$. The equations used are inspired by [27], which may differ to the approach taken in Rocket League that remains unclear.
Car-to-car interaction		Implemented using the Bullet and Psyonix impulses, but not thoroughly tested
Demolitions	[28]	Implemented, but not thoroughly tested and hence not considered in this paper

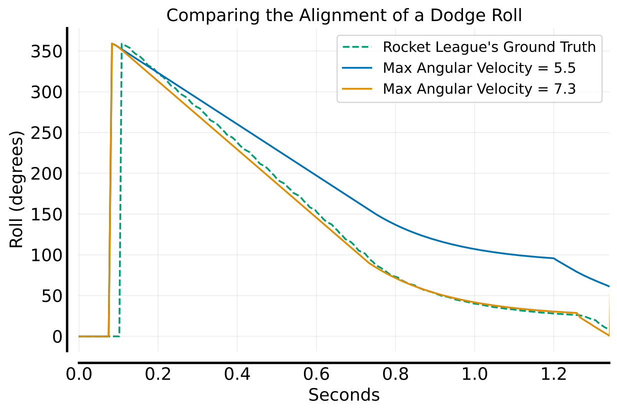

Fig. 2. The physical maneuver of a dodge roll is executed to exemplary show the alignment of the Unity simulation to the ground truth by using different max angular velocities.

that most measures are given in *unreal units* (uu). To convert them to Unity's scale, these have to be divided by 100.

Some adjustments are based on empirical findings by comparing the outcome of distinct physical maneuvers inside the implemented training simulation and the ground truth provided by Rocket League. A physical maneuver simulates several player inputs over time, such as applying throttle and steering left or right. While the simulation is conducted in both simulations, multiple relevant game state variables like positions, rotations, and velocities are monitored for later evaluation. Figure 2 is an example where the physical maneuver orders the car to execute a dodge roll. Whereas the original max angular velocity of $5.5\frac{\text{rad}}{\text{s}}$ does not compare well to the ground truth, a more suitable value of $7.3\frac{\text{rad}}{\text{s}}$ is found by analyzing the observed data.

The speed of the training simulation is about 950 $steps/second$, while RLBot is constrained to the real-time, where only 120 $steps/second$ are possible. This simulation performance is measured on a Windows Desktop

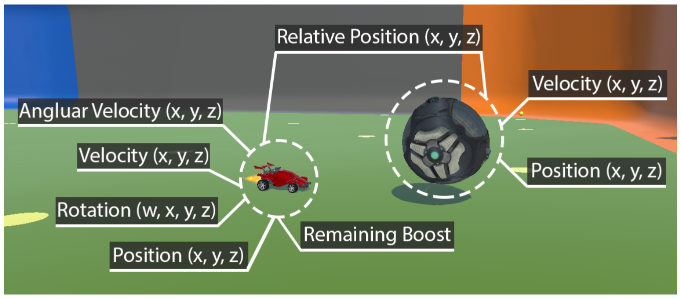

Fig. 3. The contents of the agent's observation.

utilizing a GTX 1080 and a AMD Ryzen 7 3700X.

B. Goalie Environment

In the goalie environment, the agent is asked to save shots. 1000 different samples of shots, which uniformly vary in speed, direction, and origin, are faced by the agent during training. In every episode, one shot is fired towards the agent's goal. The agent's position is reset to the center of the goal at the start of each episode. Every save rewards the agent with $+1$. A goalkeeping episode terminates if the ball hits the goal or is deflected by the agent.

C. Striker Environment

To score a goal is the agent's task inside the striker environment. The ball moves bouncy, slowly, close, and in parallel to the goal. Its speed and origin are sampled uniformly from 1000 samples during the agent's training. The agent's position is farther away from the goal while being varied as well. $+1$ is the only reward signal that the agent receives upon scoring. Once the ball hits the goal or a time limit is reached, the episode terminates and the environment is reset.

D. Observation and Action Space

Both environments share the same observation and action space. The agent perceives 23 normalized game state variables to fully observe its environment as illustrated by figure 3. The agent's action space is multi-discrete and contains the following 8 dimensions:

- Throttle (5 actions)
- Steer (5 actions)
- Yaw (5 actions)
- Pitch (5 actions)
- Roll (3 actions)
- Boost (2 actions)
- Drift or Air Roll (2 actions)
- Jump (2 actions)

Rocket League is usually played by humans using a gamepad as input device. Some of the inputs (e.g. thumbstick) are thus continuous and not discrete. To simplify the action space, the continuous actions throttle, steer, yaw, and pitch are discretized using buckets as suggested by Pleines et al. [29]. By this means, the agent picks one value from a bucket containing the values -1, -0.5, 0, 0.5 and 1. The roll action is also discretized using the values -1, 0 and 1. All other actions determine whether the concerned discrete action is executed or not. The action dimension that is in charge of drifting and air rolling is another special case. Both actions can be boiled down to one because drifting is limited to being on the ground, whereas air rolling can be done in the air only. Moreover, multi-discrete action spaces allow the execution of concurrent actions. One discrete action dimension could achieve the same behavior. This would require defining actions that feature every permutation of the available actions. As a consequence, the already high-dimensional action space of Rocket League would be much larger and therefore harder to train.

IV. Deep Reinforcement Learning

The actor-critic, on-policy algorithm PPO [7] and its clipped surrogate objective (Equation 1) is used to train the agent's policy π, with respect to its model parameters θ, inside the Unity simulation. PPO, algorithmic details, and the model architecture are presented next.

A. Proximal Policy Optimization

$L_t^C(\theta)$ denotes the policy objective, which optimizes the probability ratio of the current policy π_θ and the old one $\pi_{\theta old}$:

$$L_t^C(\theta) = \hat{\mathbb{E}}_t[min(q_t(\theta)\hat{A}_t, clip(q_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)] \quad (1)$$

$$\text{with the surrogate objective } q_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta\text{old}}(a_t|s_t)}$$

s_t is the environment's state at step t. a_t is an action tuple, which is executed by the agent, while being in s_t. The clipping range is stated by ϵ and $\hat{A}_t$ is the advantage, which is computed using generalized advantage estimation [30]. While computing the squared error loss L_t^V of the value function, the maximum between the default and the clipped error loss is determined.

$$V_t^C = V_{\theta old}(s_t) + clip(V_\theta(s_t) - V_{\theta old}(s_t), -\epsilon, \epsilon) \quad (2)$$

$$L_t^V = max((V_\theta(s_t) - G_t)^2, (V_t^C - G_t)^2) \quad (3)$$

$$\text{with the sampled return } G_t = V_{\theta old}(s_t) + \hat{A}_t$$

The final objective is established by $L_t^{CVH}(\theta)$:

$$L_t^{CVH}(\theta) = \hat{\mathbb{E}}_t[L_t^C(\theta) - c_1 L_t^V(\theta) + c_2\mathcal{H}[\pi_\theta](s_t)] \quad (4)$$

To encourage exploration, the entropy bonus $\mathcal{H}[\pi_\theta](s_t)$ is added and weighted by the coefficient c_2. Weighting is also applied to the value loss using c_1.

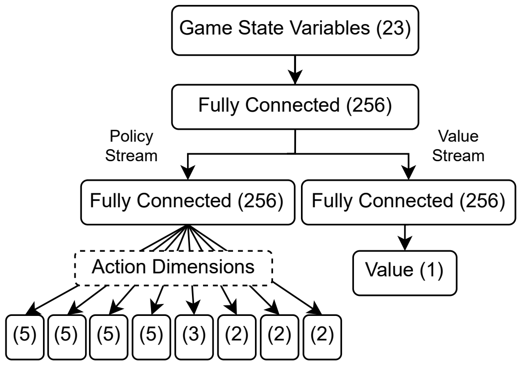

Fig. 4. The policy and the value function share gradients and several parameters. After feeding 23 game states variables as input to the model and processing a shared fully connected layer, the network is split into a policy and value stream starting with their own fully connected layer. The policy stream outputs action probabilities for each available action dimension, whereas the value stream exposes its estimated state-value.

B. Algorithmic Details and Model Architecture

PPO starts out by sampling multiple trajectories of experiences, which may contain multiple completed and truncated episodes, from a constant number of concurrent environments (i.e. workers). The model parameters are then optimized by conducting stochastic gradient descent for several epochs of mini-batches, which are sampled from the collected data. Before computing the loss function, advantages are normalized across each mini-batch. The computed gradients are clipped based on their norm.

A relatively shallow neural net (model) is shared by the value function and the policy (Figure 4). To support multi-discrete actions, the policy head of the model outputs 8 categorical action probability distributions. During action selection, each distribution is used to sample actions, which are provided to the agent as a tuple. The only adjustment to the policy's loss computation is that the probabilities of the selected actions are concatenated. Concerning the entropy bonus, the mean of the action distributions' entropies is used.

V. Sim-to-sim Transfer

Two major approaches are considered to examine learned behaviors inside the Unity simulation and its transfer to Rocket League. The first one runs various handcrafted scenarios (like seen in section III-A) in both simulations to directly compare their alignment. This way, it can be determined whether the car or the ball behave similarly or identically concerning their positions and velocities. The second approach trains the agent in Unity given the goalie and the striker environment, while all implemented physics components are included. We further conduct an ablation study on the implemented physics where each experiment turns off one or all components. Turning off may also refer to use the default physics of Unity.

If not stated otherwise, each training run is repeated 5 times and undergoes a thorough evaluation. Each model checkpoint is evaluated in Unity and Rocket League by 10 training and 10 novel shots, which are repeated 3 times. Therefore, each data point aggregates 150 episodes featuring one shot. Result plots

TABLE II
THE RESULTED ERROR FOR EACH RUN PHYSICAL MANEUVER SCENARIO. THE CAR'S POSITION IS CONSIDERED BY THE GREEN SHADED DATA POINTS, WHILE THE BLUE ONES ARE RELATED TO THE BALL'S POSITION.

	1) Acceleration			**2) Air Control**			**3) Drift**			**4) Jump**			**5) Ball Bounce**			**6) Shot**		
Mean	0.69	3.72	1.67	2.32	3.07	5.24	3.19	0.84	4.87	0.07	0.22	1.37	0.01	0.05	0.03	28.45	23.79	28.31
Std	0.48	3.05	1.92	2.61	1.81	4.58	4.73	1.04	6.80	0.06	0.15	0.69	0.01	0.03	0.02	25.20	25.01	22.49
Max	1.21	8.04	5.96	8.40	8.12	12.97	16.06	5.12	21.08	0.24	0.41	2.02	0.02	0.12	0.07	58.16	59.00	58.19

TABLE III
THE HYPERPARAMETERS USED TO CONDUCT THE TRAINING WITH PPO. THE LEARNING RATE α AND c_2 DECAY LINEARLY OVER TIME.

Hyperparameter	Value	Hyperparameter	Value
Discount Factor γ	0.99	Clip Range ϵ	0.2
λ (GAE)	0.95	c_1	0.25
Number of Workers	16	Initial α	0.0003
Worker Steps	512	Min α	0.000003
Batch Size	8192	Initial c_2	0.0005
Epochs	3	Min c_2	0.00001
Mini Batches	8	Optimizer	AdamW
Max Gradient Norm	0.5	Activations	ReLU

show the interquartile mean of the cumulative reward and a confidence interval of 95% as recommended by Agarwal et al. [31]. The hyperparameters are detailed in Table III. At last, we describe some of the learned behaviors that are also retrieved from training in a more difficult striker environment.

A. Alignment Comparison using Handcrafted Scenarios

To directly compare the alignment between both simulations, six physical maneuvers are assessed by 3 different handcrafted scenarios:

1) Acceleration
   - Car drives forward and steers left and right
   - Car drives backward and steers left and right
   - Car uses boost and steers left and right
2) Air Control
   - Car starts up in the air, looks straight up, boosts shortly and boosts while rolling in the air
   - Car starts up in the air, has an angle of $45°$, boosts shortly and boosts while rolling in the air
   - Car starts up in the air, looks straight up and concurrently boosts, yaws, and air rolls
3) Drift
   - Car drives forward for a bit and then starts turning and drifting while moving forward
   - Car drives backward for a bit and then starts turning and drifting while moving forward
   - Car uses boost and then starts turning and drifting while using boost
4) Jump
   - Car makes a short jump, then a long one and at last a double jump
   - Car makes a front flip, a back flip and a dodge roll
   - Car drives forward, does a diagonal front flip and at last a back flip
5) Ball Bounce
   - Ball falls straight down
   - Ball falls down with an initial force applied on its x-axis
   - Ball falls down with an initial force applied on its x-axis and an angular velocity
6) Shot
   - Car drives forward and hits the motionless ball
   - Car drives forward and the ball rolls to the car
   - Ball jumps, the car jumps while boosting and hits the ball using a front flip

Each scenario tracks the position of the ball and the car during each frame. As both simulations end up monitoring the incoming data with slight time differences, the final data is interpolated to match in shape. Afterward, the error for each data point between both simulations is measured. The final results are described by Table II, which comprises the mean, max, and standard deviation (Std) error across each run scenario. Letting the ball bounce for some time shows the least error, while a significant one is observed when examining the scenarios where the car shoots the ball. Note that slight inaccuracies during acceleration may cause a strongly summed error when considering a different hit location on the ball. The other scenarios, where the error is based on the car's position, also indicate that the Unity simulation suffers from inaccuracies.

B. Physics Ablation Study based on PPO Training

The previously shown imperfections of the Unity simulation may lead to the impression that successfully transferring a trained behavior is rather unlikely. This assumption can be negated by considering the results retrieved from training the agent in the goalie environment (Figure 5). Even though each experiment ablates all, single or no physical adaptations, the agent is still capable of saving nearly every ball once transferred to Rocket League. A drawback of the goalie environment lies in its simplicity because the agent only has to somehow hit the ball to effectively deflect it. The next step of complexity is posed by the striker environment, where the agent has to land a more accurate hit on the ball to score a goal. Figure 6 illustrates the results of the striker training. Notably, when all physical adaptations are present, the transferred behavior manages to score in about 75% of the played episodes. Catastrophic performances emerge in Rocket League once single physical adaptations are turned off.

C. Learned Policies

During the performed experiments, several intriguing agent behaviors emerged[2]. When trained as a goalkeeper, the agent

[2]https://www.youtube.com/watch?v=WXMHJszkz6M&list=PL2KGNY2Ei3ix7Vr_vA-ZgCyVfOCfhbX0C

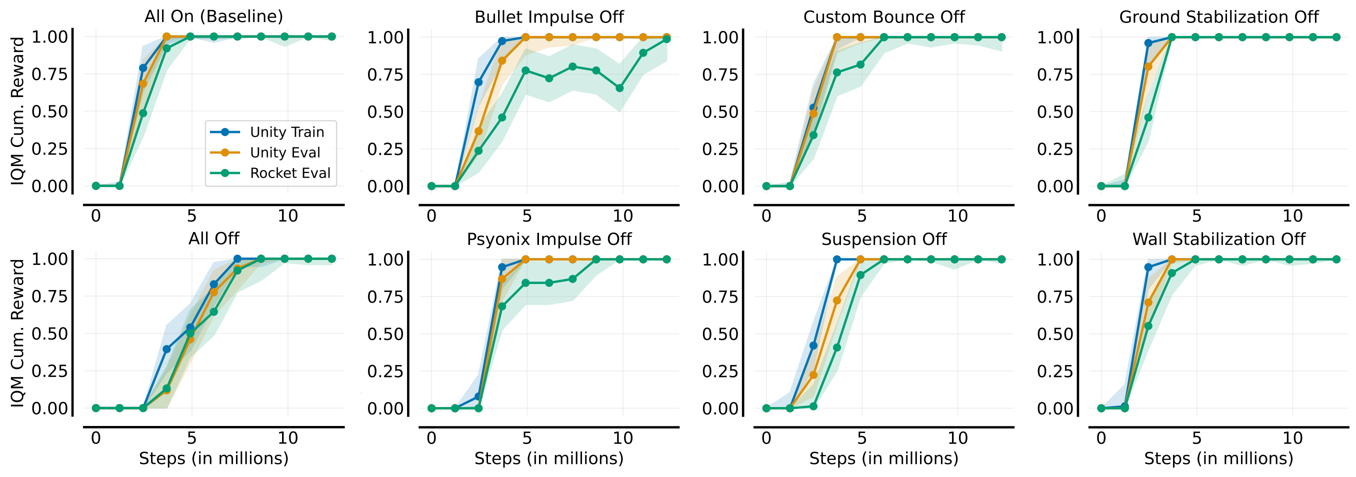

Fig. 5. Results of training the goalie environment under different ablations and transferring it to Rocket League. The agent is evaluated on training shots and ones, which were not seen during training. The agent easily solves the goalie task under all circumstances. Both, training and unseen shots, behave identically in Rocket League.

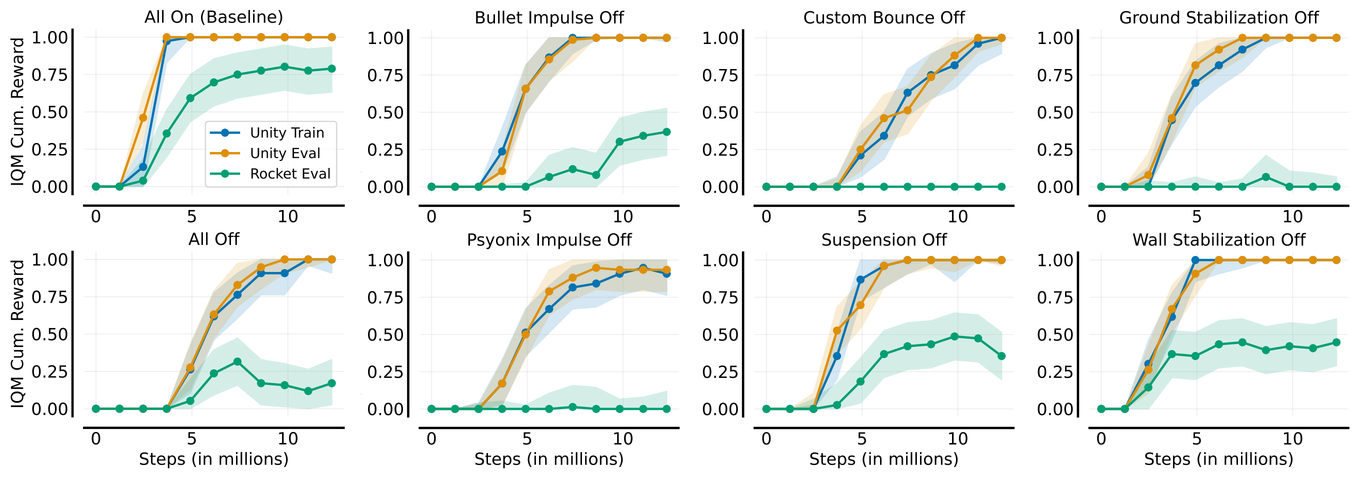

Fig. 6. Results of training the striker environment under different ablations and transferring it to Rocket League. The agent is evaluated on training situations and ones, which were not seen during training. The agent scores in about 75% of the played episodes given all physical adaptations, while any ablation turns out catastrophic. Both, training and unseen situations, behave identically in Rocket League.

tries to hit the ball very early, while making its body as big as possible towards the ball. This is achieved by simultaneously jumping and rolling forward or executing a forward flip. Concerning the striker environment, the agent usually approaches the ball using its boost. To get a better angle to the ball, the agent steers left and right or vice versa. Drifting is sometimes used to aid this purpose. Jumping is always used when needed. This is usually the case if the agent is close to the ball, which is located above the agent. Otherwise, the agent's preference is to stay on the ground. Further training experiments were conducted in a more difficult striker environment. The ball is not anymore simply passed in parallel and close to the goal. Instead, the ball bounces higher and farther away from the goal, which increases the challenge of making a good touch on the ball to score. Given this setting, two different policies were achieved. One policy approaches the ball as fast as possible while using a diagonal dodge roll to make the final touch to score. However, this behavior fails a few shots. The other emerged behavior can be considered as the opposite. Depending on the distance and the height of the ball, the agent waits some time or even backs up to ensure that it will hit the ball while being on the ground. Therefore, the agent avoids jumping. This is surprising because the agent should maximize its discounted cumulative reward and therefore finish the episode faster. Although the increased difficulty led to different behaviors, the agent may struggle a lot to get there. Usually, 2 out of 5 training runs succeeded, while the other ones utterly failed.

VI. Discussion

In this work, the agent is trained on isolated tasks, which are quite apart from a complete match of Rocket League. To train multiple cooperative and competitive agents, the first obstacle that comes to mind is the tremendously high computational complexity, which might be infeasible for smaller research groups. But before going this far, several aspects need to be considered that can be treated in isolation as well. At last, the difficulties of training the more difficult striker environments are discussed.

A. On Improving the Sim-to-sim Transfer

At first, the Unity simulation is still lacking the implementation of physical concepts like the car-to-car interaction and suffers from the reported (Section V-A) inaccuracies. These can be further improved by putting more work into the simulation, but also other approaches are promising. At the cost of more computational resources, domain randomization [20] could achieve a more robust agent, potentially comprising an improved ability to generalize to the domain of Rocket League. As the ground truth is provided by Rocket League, approaches from the field of supervised learning can be considered as well.

B. Training under Human Conditions

Once the physical domain gap is narrowed, the Unity simulation still does not consider training under human conditions. Notably, the current observation space provides perfect information on the current state of the environment, whereas players in Rocket League have to cope with imperfect information due to solely perceiving the rendered image of the game. Thus, the Unity simulation has to implement Rocket League's camera behavior as well. However, one critical concern is that the RLBot API does not reveal the rendered image of Rocket League and therefore makes a transfer impossible as of now. However, even if that information is made available by Psyonix, both simulations' visual appearances are very different. The Unity simulation's aesthetics are very abstract, whereas Rocket League impresses with multiple arenas featuring many details concerning lighting, geometry, shaders, textures, particle effects, etc.. To overcome this gap of visual appearance, approaches of the previously described related work, like GraspGAN [18], can be considered.

Another challenge arises once the environment is partially observable. It should be considered that the agent will probably need memory to be able to compete with human players. Otherwise, the agent might not be able to capture the current affairs of its teammates and opponents. For this purpose, multiple memory-based approaches might be suitable, like using a recurrent neural network or a transformer architecture.

Moreover, the multi-discrete action space used in this paper is a simplification of the original action space that features concurrent continuous and discrete actions. Initially, the training was done using the PPO implementation of the ML-Agents toolkit [22], which supports mixed (or hybrid) concurrent action spaces. However, these experiments were quite unstable and hindered progress. Therefore, Rocket League presents an interesting challenge for exploring such action spaces, of which other video games or applications are likely to take advantage.

C. Difficulties of Training the harder Striker Environment

While the goalie and the striker environment are relatively easy, the slightly more difficult striker one poses a much greater challenge due to multiple reasons:

- Episodes are longer, leading to an even more delayed reward signal and more challenging credit assignment
- More states have to be explored by the agent
- Even more accurate touches on the ball have to be made to score

To overcome these problems, curriculum learning [32] and reward shaping [33] can be considered. In curriculum learning, the agent could face easier scenarios first and once success kicks in, the next level of difficulty can be trained. However, catastrophic forgetting may occur and therefore a curriculum should sample from a distribution of scenarios to mitigate this issue.

Concerning reward shaping, multiple variants were casually tried without improving training results:

- Reward the first touch on the ball
- Reward or penalize the distance between the ball and the agent
- Reward or penalize the dot product between the car's velocity and the direction from the car to the ball

Adding more reward signals along the agent's task introduces bias and is likely task-irrelevant. For example, the agent could exploit such signals to cuddle with the ball at a close distance or to slowly approach the ball to maximize the cumulative return of the episode. If those signals are turned off once the ball is touched, the value function might struggle to make further good estimates on the value of the current state of the environment, which ultimately may lead to misleading training experiences and hence an unstable learning process. In spite of the results of these first explorative tests, future work shall examine whether these points shall be reconsidered.

VII. Conclusion

Towards solving Rocket League by the means of Deep Reinforcement Learning, a fast simulation is crucial, because the original game cannot be sped up and neither parallelized on Linux-based clusters. Therefore, we advanced the implementation of a Unity project that mimics the physical gameplay mechanics of Rocket League. Although the implemented simulation is not perfectly accurate, we remarkably demonstrate that transferring a trained behavior from Unity to Rocket League is robust and generalizes when dealing with a goalkeeper and striker task. Hence, the sim-to-sim transfer is a suitable approach for learning agent behaviors in complex game environments. After all, Rocket League still poses further challenges when targeting a complete match under human circumstances. Based on our findings, we believe that Rocket League and its Unity counterpart will be valuable to various research fields and aspects, comprising: sim-to-sim transfer, partial observability, mixed action-spaces, curriculum learning, competitive and cooperative multi-agent settings.

References

[1] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, J. Oh, D. Horgan, M. Kroiss, I. Danihelka, A. Huang, L. Sifre, T. Cai, J. P. Agapiou, M. Jaderberg, A. S. Vezhnevets, R. Leblond, T. Pohlen, V. Dalibard, D. Budden, Y. Sulsky, J. Molloy, T. L. Paine, Ç. Gülçehre, Z. Wang, T. Pfaff, Y. Wu, R. Ring, D. Yogatama, D. Wünsch, K. McKinney, O. Smith, T. Schaul, T. P. Lillicrap, K. Kavukcuoglu, D. Hassabis, C. Apps, and D. Silver, "Grandmaster level in starcraft II using multi-agent reinforcement learning," *Nat.*, vol. 575, no. 7782, pp. 350–354, 2019.

[2] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Debiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, R. Józefowicz, S. Gray, C. Olsson, J. Pachocki, M. Petrov, H. P. de Oliveira Pinto, J. Raiman, T. Salimans, J. Schlatter, J. Schneider, S. Sidor, I. Sutskever, J. Tang, F. Wolski, and S. Zhang, "Dota 2 with large scale deep reinforcement learning," *CoRR*, vol. abs/1912.06680, 2019.

[3] RoboLeague, "Roboleague," 2021, available at https://github.com/roboserg/RoboLeague retrieved February 28, 2022.

[4] RLBot, "Rlbot wiki: Getting started," 2021, available at https://github.com/RLBot/RLBot/wiki retrieved February 28, 2022.

[5] Y. Verhoeven and M. Preuss, "On the potential of rocket league for driving team ai development," in *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, 2020, pp. 2335–2342.

[6] M. Mozgovoy, M. Preuss, and R. Bidarra, "Guest editorial special issue on team AI in games," *IEEE Trans. Games*, vol. 13, no. 4, pp. 327–329, 2021.

[7] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv:1707.06347*, 2017.

[8] H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, and E. Osawa, "Robocup: The robot world cup initiative," in *Proceedings of the First International Conference on Autonomous Agents, AGENTS 1997, Marina del Rey, California, USA, February 5-8, 1997*, W. L. Johnson, Ed. ACM, 1997, pp. 340–347.

[9] M. J. Hausknecht and P. Stone, "Deep reinforcement learning in parameterized action space," in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2016.

[10] E. Antonioni, V. Suriani, F. Riccio, and D. Nardi, "Game strategies for physical robot soccer players: A survey," *IEEE Trans. Games*, vol. 13, no. 4, pp. 342–357, 2021.

[11] W. Zhao, J. P. Queralta, and T. Westerlund, "Sim-to-real transfer in deep reinforcement learning for robotics: a survey," in *2020 IEEE Symposium Series on Computational Intelligence, SSCI 2020, Canberra, Australia, December 1-4, 2020*. IEEE, 2020, pp. 737–744.

[12] B. Balaji, S. Mallya, S. Genc, S. Gupta, L. Dirac, V. Khare, G. Roy, T. Sun, Y. Tao, B. Townsend, E. Calleja, S. Muralidhara, and D. Karuppasamy, "Deepracer: Autonomous racing platform for experimentation with sim2real reinforcement learning," in *2020 IEEE International Conference on Robotics and Automation, ICRA 2020, Paris, France, May 31 - August 31, 2020*. IEEE, 2020, pp. 2746–2754.

[13] J. Blumenkamp, A. Baude, and T. Laue, "Closing the reality gap with unsupervised sim-to-real image translation for semantic segmentation in robot soccer," *CoRR*, vol. abs/1911.01529, 2019.

[14] R. Traoré, H. Caselles-Dupré, T. Lesort, T. Sun, N. D. Rodríguez, and D. Filliat, "Continual reinforcement learning deployed in real-life using policy distillation and sim2real transfer," *CoRR*, vol. abs/1906.04452, 2019.

[15] O. Pedersen, E. Misimi, and F. Chaumette, "Grasping unknown objects by coupling deep reinforcement learning, generative adversarial networks, and visual servoing," in *2020 IEEE International Conference on Robotics and Automation, ICRA 2020, Paris, France, May 31 - August 31, 2020*. IEEE, 2020, pp. 5655–5662.

[16] K. Rao, C. Harris, A. Irpan, S. Levine, J. Ibarz, and M. Khansari, "Rl-cyclegan: Reinforcement learning aware simulation-to-real," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*. Computer Vision Foundation / IEEE, 2020, pp. 11 154–11 163.

[17] D. Ho, K. Rao, Z. Xu, E. Jang, M. Khansari, and Y. Bai, "Retinagan: An object-aware approach to sim-to-real transfer," in *IEEE International Conference on Robotics and Automation, ICRA 2021, Xi'an, China, May 30 - June 5, 2021*. IEEE, 2021, pp. 10 920–10 926.

[18] K. Bousmalis, A. Irpan, P. Wohlhart, Y. Bai, M. Kelcey, M. Kalakrishnan, L. Downs, J. Ibarz, P. Pastor, K. Konolige, S. Levine, and V. Vanhoucke, "Using simulation and domain adaptation to improve efficiency of deep robotic grasping," in *2018 IEEE International Conference on Robotics and Automation, ICRA 2018, Brisbane, Australia, May 21-25, 2018*. IEEE, 2018, pp. 4243–4250.

[19] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. C. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds., 2014, pp. 2672–2680.

[20] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2017, Vancouver, BC, Canada, September 24-28, 2017*. IEEE, 2017, pp. 23–30.

[21] OpenAI, I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas, J. Schneider, N. Tezak, J. Tworek, P. Welinder, L. Weng, Q. Yuan, W. Zaremba, and L. Zhang, "Solving rubik's cube with a robot hand," *CoRR*, vol. abs/1910.07113, 2019.

[22] A. Juliani, A. Khalifa, V. Berges, J. Harper, E. Teng, H. Henry, A. Crespi, J. Togelius, and D. Lange, "Obstacle tower: A generalization challenge in vision, control, and planning," in *Proceedings of the 28th International Joint Conference on Artificial Intelligence, IJCAI 2019*, 2019, pp. 2684–2691.

[23] S. Mish, "Rocket league notes," 2019, available at https://samuelpmish.github.io/notes/RocketLeague/ retrieved February 28, 2022.

[24] Timo Huth, "Dodges explained. power & more - rocket science #14," 2018, available at https://www.youtube.com/watch?v=pX950bhGhJE retrieved February 28, 2022.

[25] J. Cone, "It is rocket science! the physics of 'rocket league' detailed," 2018, available at https://www.gdcvault.com/play/1024972/It-IS-Rocket-Science-The retrieved February 28, 2022.

[26] Timo Huth, "Why are wheel hits so odd? - rocket science #10," 2017, available at https://www.youtube.com/watch?v=pTAVP00xwF4 retrieved February 28, 2022.

[27] Vehicle Physics Pro, "How simple suspensions work," available at https://vehiclephysics.com/advanced/how-suspensions-work/ retrieved February 28, 2022.

[28] Rocket Sledge, "How demos actually work in rocket league," 2020, available at https://www.gamersrdy.com/blog/2020/06/11/how-demos-actually-work-in-rocket-league/#:~:text=Demos%20were%20not%20intended%20to,broken%20for%20a%20long%20time retrieved February 28, 2022.

[29] M. Pleines, F. Zimmer, and V. Berges, "Action spaces in deep reinforcement learning to mimic human input devices," in *IEEE Conference on Games, CoG 2019, London, United Kingdom, August 20-23, 2019*. IEEE, 2019, pp. 1–8.

[30] J. Schulman, P. Moritz, S. Levine, M. I. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2016.

[31] R. Agarwal, M. Schwarzer, P. S. Castro, A. Courville, and M. G. Bellemare, "Deep reinforcement learning at the edge of the statistical precipice," in *Thirty-Fifth Conference on Neural Information Processing Systems*, 2021.

[32] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *Proceedings of the 26th Annual International Conference on Machine Learning, ICML 2009, Montreal, Quebec, Canada, June 14-18, 2009*, ser. ACM International Conference Proceeding Series, A. P. Danyluk, L. Bottou, and M. L. Littman, Eds., vol. 382. ACM, 2009, pp. 41–48.

[33] V. Gullapalli and A. Barto, "Shaping as a method for accelerating reinforcement learning," in *Proceedings of the 1992 IEEE International Symposium on Intelligent Control*, 1992, pp. 554–559.