Appendix: Strategy and Skill Learning for Physics-based Table Tennis Animation

Jiashun Wang
jiashunw@cs.cmu.edu
Carnegie Mellon University
USA

Jessica Hodgins
jkh@cmu.edu
Carnegie Mellon University
and The AI Institute
USA

Jungdam Won
jungdam@imo.snu.ac.kr
Seoul National University
South Korea

We describe the details of the states and actions used in our controllers, the training, and the implementation of the network architecture and hyper-parameters. We also provide more details of the skill and strategy evaluation.

A STATES AND ACTIONS

We describe the states and actions for both skill-level and strategy-level controllers.

Skill-level controller. We follow [Peng et al. 2022] to use the agent's local coordinate frame for the skill-level controller. The agent's local coordinate frame is defined with the origin located at the root, the x-axis oriented along the root link's facing direction, and the y-axis aligned with the global up vector. Agent's state $s \in \mathbb{R}^{226}$ consists of the height of the root, the rotation and position of the root in the local coordinate frame, the rotation and position of each joint in the local coordinate frame, the linear and angular velocity of each joint in the local coordinate frame, and the position of the paddle in the local coordinate frame. Ball state's $b \in \mathbb{R}^9$ includes the ball velocity, the distance between the ball and root, and the distance between the ball and paddle, computed in the agent's local coordinate frame. Target $y \in \mathbb{R}^3$ includes the distance between the ball and target in the agent's local coordinate frame. δ is a one-hot vector determining the skill to use. Agent's action $a \in \mathbb{R}^{31}$ is the target joint angles except for the root for PD controllers and the blending weights $\varphi \in \mathbb{R}^{31}$ is used to mix the skill actions in a joint-wise manner.

Strategy-level controller. Agent's state $s \in \mathbb{R}^6$ consists of the root position and paddle position in the table frame. Opponent's state $\tilde{s} \in \mathbb{R}^6$ consists of the root position and paddle position of the opponent in the table frame. Ball's state $b \in \mathbb{R}^6$ includes the ball position and velocity in the table frame. The table's longer edge is the x-axis, its shorter edge is the y-axis, and the z-axis is the gravity axis. Agent's strategy action $c \in \mathbb{R}^6$ includes the $\delta \in \mathbb{R}^5$ determining the skill to use and $y \in \mathbb{R}^2$ the target landing location on the table in the table frame.

B TRAINING DETAILS

We utilize IsaacGym [Makoviychuk et al. 2021] to train the agents with a simulation frequency of 120 Hz. For control policies, imitation policies run at 30Hz. Mixer and ball control policies run at 15Hz. The RL policies ω and π are trained with proximal policy optimization (PPO) [Schulman et al. 2017]. We collect 18 minutes of reference motion of a high-ranking player with the Vicon Motion Capture system. We perform a broad categorization including drive, push, and smash.

All the networks are trained with Pytorch [Paszke et al. 2019] and all training processes are performed on NVIDIA RTX 4090. For the VR experiments, we finetune the policies to run with a control frequency of 60 Hz to enhance the interaction experience. Each imitation policy is trained with 2 billion samples, taking 12 hours. Each ball control policy and mixer policy is trained with 4 billion samples, taking about 1 day. The strategy learning takes about 3 minutes to collect data and 5 minutes to train for one iteration. We apply 5 iterations for agent-agent experiments and 2 iterations for human-agent experiments. We split the reference motion into five subsets: Forehand Drive, Forehand Push, Forehand Smash, Backhand Drive and Backhand Push. A drive is a stroke that is primarily used to keep the ball in play with a fast and flat shot. A push is a stroke that requires the player to strike downwards on the back and underneath the ball to create a backspin. A smash is a fast, hard and powerful stroke that drives the ball downward. We train imitation and ball control policies with each skill separately. We train the universal imitation policy and mixer policy with all the data.

C NETWORK ARCHITECTURE AND HYPER-PARAMETERS

The imitation policies are modeled by a neural network to map to a Gaussian Distribution $\pi^i(a^i|s,z^i) = \mathcal{N}(\mu_{\pi^i}(s,z^i),\Sigma_{\pi^i})$, where $i \in \{1,2,3,4,u\}$. Specifically, the mean μ_{π^i} is predicted by an MLP with three hidden layers of [1024, 1024, 512] units followed by a linear output layer, and the diagonal covariance matrix Σ_{π^i} is set to 0.0025 on each diagonal element. The value network is a similar architecture but the final output is a scalar. The encoder $q^i(z^i|s,s')$ and discriminator $D^i(s,s')$ are modeled by a single network which outputs both the mean of the encoder $\mu_{q^i}(s,s')$ and the discriminator value. The ball control policies $\omega^i(z^i|s,b,y) = \mathcal{N}(\mu_{\omega^i}(s,b,y),\Sigma_{\omega^i})$ are also modeled by a neural network to map to a Gaussian Distribution. Specifically, the mean $\mu_{\omega^i}(s,b,y)$ is predicted by an MLP with two hidden layers of [1024,512] units followed by a linear output layer, and the diagonal covariance matrix Σ_{ω^i} is set to 0.01 on each diagonal element. The output z^i is normalized with its norm before sending to the imitation policies. The mixer policy is similar to the ball control policy except it takes s, b, y and δ as inputs and outputs $\mu_{\omega^m}(s,b,\delta,y)$ and blending weight φ. f takes the strategy observation $o = (s,\tilde{s},b)$ as input and outputs the strategy action $c = (\delta, y)$. It is modeled by an MLP with three hidden layers of [1024, 1024, 512] followed by a linear output layer. We use ReLU for all the activations except the outputs of the discriminator value and blend weight φ, which use a Sigmoid. Table 1 reports the hyper-parameters used in our experiments.

Table 1: Hyper-parameters.

Parameter	Value
Discount factor γ	0.99
GAE and TD λ	0.95
Episode length	500
Learning rate	$1.0e^{-5}$
# tuples per update	258944
Policy Batch Size	16384
Discriminator Batch Size γ	4096
PPO Clip Threshold	0.2
Latent Space Dimension	64
Gradient Penalty Weight λ_{gp}	5
Diversity Objective Weight λ_D	0.01
CVAE KL Divergence Weight β_{KL}	0.01
Skill Discovery Objective Weight β	0.5
Paddle Reward Weight w_p	0.8
Ball Reward Weight w_b	0.8
Style Reward Weight w_r	0.2

D SKILL EVALUATION

ASE [Peng et al. 2022], CASE [Dou et al. 2023], and ET are trained under the same setting as our mixer policy. ET learns a high-level policy to output the latent action for the universal imitation policy. For ET, we train the five skills first and then we fix the individual skill controllers and let the ET controller take control during the transition (when the ball passes the net until it is returned to the agent). For other time steps, we directly let each individual controller control the agent. For training the CASE method, we modify their code by applying the δ as their conditions and keep the same dimension of the latent z as ours because there are only five skill categories in our setting. To train each evaluation discriminator D^i_{test}, we utilize the reference motions of the i-th skill as positive data and all the other reference motions as negative data. We use the same hyperparameters as those used in training the imitation policies; however, the discriminators for evaluation are independently trained for fair evaluation. During the skill evaluation, we collect 1 hour of ball tracking data of a match between two high-ranking players using the SPINSIGHT software[1] including the speed, including the position and speed of the ball when it contacts the paddle, touches the table, and passes the net. We use this ball tracking data to test the skill controller's ability with more challenging cases.

E STRATEGY EVALUATION

For the opponent used in strategy evaluation, the *random op* utilizes a strategy that uniformly samples the skill to use and the target land location of the ball. For the *video op*, we collect 20 minutes of high-ranking player broadcast video and followed the annotation process of [Zhang et al. 2023] to get the video expert demonstration $\{(o_k^{\text{video}}, c_k^{\text{video}})\}_{k=1}^K$, by which the video strategy is trained.

We utilize reinforcement learning (RL) trained with PPO [Schulman et al. 2017] as a baseline. We collect state action pairs $(s, \tilde{s}, b, c)$ to update the policy. For the competition setting, the reward is defined as r_g, which is the task goal reward applied to the final step, $r_g = 10$ if the agent wins and $r_g = -10$ if it loses. Empirically, we find this sparse goal reward is actually better than the combination of r_g and the continuous reward r described in Section 4.2. For the cooperation setting, the reward is 1 for each time step and we set the episode length to 1000.

We also report the results of our strategy learning approach at each iteration. We report the winning rate for each iteration in Table 2 and average rounds for each iteration in Table 3. We observe that the improvements in the winning rate at the first iteration are the most significant and then it converges gradually. For example, the winning rate increases by about 8% in the first iteration and then only increases by about 10% in the next four iterations. For the cooperation setting, the average rounds at the first iteration are close to that of the last iteration. We think the cooperation task is easier to learn compared to the competition match.

The video op incorporates real-world demonstrations, making them initially more challenging to defeat (48% winning rate). However, with more iterations, the random op can sample a broader and more diverse set of strategy decisions. We can observe the final winning rates are similar. In addition, the more predictable nature of the video op makes cooperation easier. On the other hand, as an opponent, video op provides a relatively fixed challenge while random op provides more diverse and unpredictable challenges, which leads to our method being stronger when trained against a random op, as shown in Table 4 in the main paper.

Table 2: Winning rate for each iteration.

Iteration	0	1	2	3	4	5
Random op	0.500	0.582	0.639	0.640	0.659	0.687
Video op	0.482	0.549	0.599	0.628	0.662	0.681

Table 3: Average rounds for each iteration.

Iteration	0	1	2	3	4	5
Random op	10.9	13.4	14.2	14.9	15.7	16.4
Video op	12.8	14.3	16.9	17.2	17.8	18.2

[1] https://spinsight.com/

REFERENCES

Zhiyang Dou, Xuelin Chen, Qingnan Fan, Taku Komura, and Wenping Wang. 2023. C·ASE: Learning Conditional Adversarial Skill Embeddings for Physics-based Characters. In *SIGGRAPH Asia 2023 Conference Papers, SA 2023*. ACM, 2:1–2:11. https://doi.org/10.1145/3610548.3618205

Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, and Gavriel State. 2021. Isaac Gym: High Performance GPU Based Physics Simulation For Robot Learning. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*. https://openreview.net/forum?id=fgFBtYgJQX_

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Z. Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019*. 8024–8035. https://proceedings.neurips.cc/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html

Xue Bin Peng, Yunrong Guo, Lina Halper, Sergey Levine, and Sanja Fidler. 2022. ASE: large-scale reusable adversarial skill embeddings for physically simulated characters. *ACM Trans. Graph.* 41, 4 (2022), 94:1–94:17. https://doi.org/10.1145/3528223.3530110

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. *arXiv preprint arXiv:1707.06347* (2017). arXiv:1707.06347 http://arxiv.org/abs/1707.06347

Haotian Zhang, Ye Yuan, Viktor Makoviychuk, Yunrong Guo, Sanja Fidler, Xue Bin Peng, and Kayvon Fatahalian. 2023. Learning Physically Simulated Tennis Skills from Broadcast Videos. *ACM Trans. Graph.* 42, 4 (2023), 95:1–95:14. https://doi.org/10.1145/3592408