FoosPong DQN: A Multi-Agent Game for Emergent Skill Acquisition through Reinforcement Learning based Autocurricula

Alec Engl
Department of Applied Physics
Stanford University
Stanford, USA
aengl@stanford.edu

Evan Hedding
Department of Aeronautics and Astronautics
Stanford University
Stanford, USA
eth8180@stanford.edu

Nitish Gudapati
Department of Mechanical Engineering
Stanford University
Stanford, USA
gnitish9@stanford.edu

Shashvat Jayakrishnan
Department of Electrical Engineering
Stanford University
Stanford, USA
shashvat@stanford.edu

***Abstract*—Multi-Agent Reinforcement Learning continues to be an active field of research with many challenges, however, success has been found in recent research making use of Deep Reinforcement Learning techniques to overcome these challenges. In this work we focus on using a Deep Q-Network for training teams to play a multi-player version of the classic game Pong, which we call FoosPong. We experiment with various team sizes and ball amounts, as well as differing training pipelines and report on the results. Even with a relatively small amount of training time, when compared to past research in reinforcement learning, we observed distinct multi-agent learning occurring, as well as various emergent behaviors relevant to gameplay.**

***Index Terms*—multi-agent, reinforcement learning, cooperation, competition, curiosity, intrinsic motivation, game theory**

I. Introduction

Multi-agent interaction is at the core of every system or process that surrounds us or that is even within us. Entities of a living cell that are each independently non-living and serve different purposes come together (cooperate) to form the very unit of life. Our ability to reflect upon this comes from cognition which too emerges from a multi-agent cooperation of neurons. The cells and entities not only cooperate but also compete for resources for their individual survival. The very fabric of existence is encoded with a balance between cooperation and competition among various agents involved, and each of these agents has an individualism along with a communal identity. While the ideas of cooperation and competition arise in the context of the communal identity of agents, individualism gives agents traits of curiosity and abilities to hold independent beliefs [1]–[3]. In this project we abstract social behaviors like cooperation and competition [4] in a Reinforcement Learning (RL) context through a multi-agent game environment - *FoosPong*. The goal of this work is to observe and achieve interesting emergent behavior by allowing social interactions of individualistic agents and using multi-agent self-play for co-evolution based skill acquisition.

The next section details upon some previous work in the field of multi-agent RL. This is followed by the description of our *FoosPong* game environment, and the state and action spaces for our game setup. Following the details of the game in section III, we detail upon the DQN algorithm used in our project in section IV. Further, we delve into the details of the DQN architecture in section V and discuss the reward structure in section VI that was used for training our agents to play *FoosPong*. Section VII elaborates on various training paradigms and game-plays that we experimented with throughout the project. Section VIII is focused on the results of our training and details upon an evaluation of our agents' performances. In this section we also uncover the various emergent behaviors demonstrated by agents during heuristic-based single team training as well as while executing self-play based decentralized multi-team training. Lastly, we conclude our results in section IX and discuss some future scope and possible extensions of our work.

II. Background

The field of multi-agent reinforcement learning (MARL) has been growing for years and has a large amount of research conducted. Markov Decision Processes (MDP) have traditionally been the standard framework for single agent decision making. The disadvantage with this approach is that the states are not entirely deterministic in practical scenarios, which call for Partially Observable Markov Decision Processes (POMDP). In a multi-agent setting, the MDPs are extended to stochastic games or Markov Games (MG) and the POMDPs are generalized to Partially Observable Markov Games (POMG). The

decentralized POMDP (Dec-POMDP) focuses on cooperative multi-agent system with shared rewards among agents [5].

The learning approaches for these models are usually classified into independent learning, joint action learning and gradient based methods. Under independent learning, the agents mutually ignore each other in the process of learning, and hence the stochasticity of the environment is not taken into account, thereby not guaranteeing convergence. In modern literature, independent learning is usually not referred under multi-agent learning. The joint-action learning addresses this issue by learning in the space of joint actions and joint states. Due to the joint representation of the states, actions and policies, the complexity of these algorithms grows exponentially with the number of agents. Gradient Ascent is an optimization technique, by following the gradient of a differentiable objective function to a local optimum [6].

One of the difficulties with MARL techniques, as opposed to traditional single-agent reinforcement learning, is that the state space of the joint system grows exponentially as the number of agents grow. This issue is compounded even further if the action space of each agent is continuous and/or the environment itself is stochastic and unpredictable [7]. Additionally, the environment is often too complex to leverage game theoretic approaches, as these often assume that actions are "one-shot" [8].

Much of the work into solving MARL generally involves a function approximation technique that can compensate for the large state space issues. Multiple algorithms have been proposed and proven successful. [7], [9], [10] utilize a flavor of Q-learning where the methodology of finding the optimal joint action strategy varies, from solving the MDP with linear programming [5] to using a deep neural network to find a solution.

In the case of Deep Reinforcement Learning (DRL), much success early on has been shown with creating superhuman gameplay with single agents [7], and then more recently with wildfire and flood monitoring by teams of drones [9], [10]. The use of DRL has shown quite promising results in these studies and is a continued field of research since it offers a solution to multi-agent problems in complex environments and offers the advantage of faster computation when compared to linear programming. Even within the field of DRL, multiple algorithms have been proposed. The Deep reinforcement algorithms are extended into three classes, namely temporal difference learning like Deep Q Networks (DQN), policy gradient like Trust Region Policy Optimization (TRPO), and Actor-Critic methods like Deep Deterministic Policy Gradients (DDPG) and Asynchronous Advantage Actor Critic (A3C).

DQN methods use a neural network to approximate the state-action value function, and relies on an experience replay dataset that stores the agent's experiences. The DQN, has been popularized in recent research, but brings about certain limitations in that the action space of each agent must be discrete and well defined [7]. TRPO is a policy gradient method that updates policies to improve performance while satisfying constraints on the closeness of the old and new policies [11]. Actor-critic methods have an actor and a critic, where the former represents a parameterized policy while the latter provides an estimate of the value function. DDPG combines actor-critic and DQN approaches for continuous action spaces. In A3C the agents interact asynchronously with their environments, learning with each interaction by predicting both the value function and optimal policy function [7].

Important to modelling MARL experiments is a consideration of three properties: degree of centralization in control, the agents' degree of observability of the environment, and the cooperative/competitive aspects of the environment. The experiments we have studies involve primarily decentralized, distributed control in a fully-observed environment [8]. These experiments are intended to be more general, and thus are helpful across a range of applications as they are more hardware-agnostic than partially-observed models (which require defining sensor arrays in each agent). We are also very interested in distributed control models for their usefulness in real world multi-agent scenarios.

We find this body of research most interesting because it focuses on a solution strategy that allows each agent to learn independently from the group, based upon individual partial observations of the environment and a reward structure that incentives coordination. This follows more along with the classical interests of reinforcement learning and allows agents to coordinate with each other toward a common goal without making use of complex communication structures or distributed control systems.

III. FoosPong Description

FoosPong (derived from an amalgamation of Foosball and Pong) is a multi-agent version of the classic Pong game. A pygame model of the environment and code was borrowed from 'PongAIvsAI' [12], as it provided flexibility in terms the position and number of paddles on each side and the number of balls. The physics of the game environment can also be modified by altering deflections angles, the paddle and ball speeds. The number of paddles were increases to two on each side to convert it to a multi-agent setting.

Other common implementations from literature use Open AI Gym with Atari Pong, which provides only a single paddle on each side and they are trained against a random agent. Hence, it cannot be utilized for training multi agent algorithms. Algorithms are also commonly implemented on Stable Baselines. In this paper, we present an implementation of the algorithm using native tensorflow.

The state space is continuous as there are many possible coordinates and velocities the paddles and balls can take, while the action space is discrete Other methods include using the frames of the game window as the state space and deep neural networks (convolution) are used to train them.

1) States (continuous):

- X, Y coordinates of each paddle - $x_p^i, y_p^i \ \ \forall i = 1, \cdots, P$
- X, Y coordinates of each ball - $x_b^j, y_b^j \ \ \forall j = 1, \cdots, B$
- X, Y velocities of the balls - $\dot{x}_b^j, \dot{y}_b^j \ \ \forall j = 1, \cdots, B$

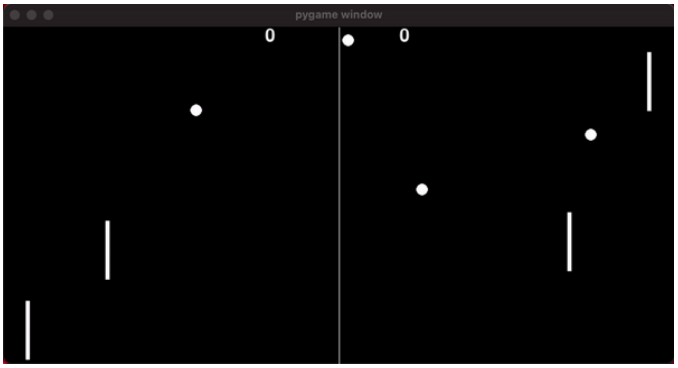

Fig. 1: FoosPong environment using Pygame

So, for a case of two paddles on each team ($P = 4$) with four balls ($B = 4$), the size of the joint states turns out to be 24 — 2 position coordinates for each paddle ($\times 4$), 2 position coordinates and corresponding velocities in each direction for each ball ($\times 4$). The x coordinates of each paddle are constant throughout the game, but they are still considered to distinguish the paddle indices from each other while training. The number of states is however infinite due to a continuous state space.

2) Actions (discrete):

- Move *up*
- Move *down*

Each paddle has two actions, and they can either move up or down. During centralized training, the agents on one side are trained against a predefined models like random, heuristic or hybrid. Hence, the action space only consists of the actions associated with each paddle only on one side. So, for the two paddles case, the number of actions is 4. During self-play, the paddles on both sides are simultaneously trained, thereby doubling the action space. An additional action of '*staying still*' is also include in some articles in literature [13], however we have not incorporated this in our model. Interestingly, this choice led to an emergent behavior demonstrated by our agents that we will discuss further in section VIII.

IV. Multi-Agent Deep Q-Network

As mentioned in the Introduction and Background sections, there are a plethora of algorithmic solutions to the multi-agent reinforcement learning problem. The method that we decided to apply to our game is that of a Deep Q-Network (DQN). We chose this method because our action space is discrete, which has been shown to work well with DQN [7], and the algorithm has achieved much success in previous work [7], [9], [10].

The objective of the DQN algorithm is to find the optimal policy π that will map the current state of the system to the optimal actions for each agent on a given team. This means establishing a function that can calculate a Q-value for any given state-action pair, and the maximum of those values will be the optimal action for that state. With DQN, we train a neural network that can be used to approximate that function. Training is performed by minimizing the mean squared error loss of the Bellman error:

$$L_i(\theta_i) = E_{(s,a,r,s')}[r(s,a) + \gamma * maxQ(s', a', \theta^-) - Q(s,a,\theta)] \quad (1)$$

where θ^- corresponds to the parameters of a previously trained target network, θ to those of the current network being trained, and γ is the discount factor. These parameters are trained over a randomly sampled collection from the experience replay, $E_{s,a,r,s'}$. This is a collection of states s and actions a taken at each time step, rewards r resulting from each action, and the resulting next states s'. These are collected at each time step and added to the replay memory. We don't start training until the replay memory has collected enough samples, at least 50,000, and we systematically delete the oldest samples as new ones are added to maintain the size of the replay memory. This method of experience replay has been shown to compensate for issues of instability inherent with simply training DRL agents on the most recent replay sample. [7].

The complete algorithm of Multi-Agent DQN is detailed in Algorithm 1, where the loss function $L(\theta)$ is that shown in Equation 1. The greedy-epsilon model θ^+ is a hard-coded function that either moves directly towards the closest ball or chooses an action at random. Then ϵ decides the proportion at which the team chooses actions based upon the training model θ or θ^+, and ϵ is incrementally reduced so that eventually all actions are being decided by the model. We found that our model benefited from a higher epsilon in the early stages of training so that heuristic/random decision making could expedite the exploratory phase.

Algorithm 1 Multi-Agent Deep Q-Network

Input ϵ, n, memory capacity L, greedy-epsilon model θ^+
Output θ

```
Initialize replay memory E
Initialize θ, θ⁻
for each episode do
    Start game loop
    for each time step t_i do
        Gather state s_i
        Based upon ε, use θ or θ⁺ to obtain action a_i
        Add s_i, a_i, r_i, s_{i+1} to replay memory
    end for
    if size E > L then
        Set θ⁻ = θ
        Randomly sample n time steps from E
        Train θ using Mean Squared Error of L(θ)
    end if
    Update θ
    Reduce ε
end for
return θ
```

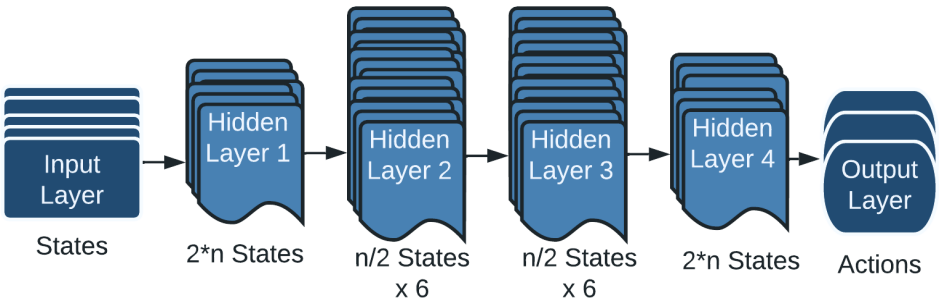

Fig. 2: DQN Network

V. Network Architecture

For training the model, we used a simple network with an input layer, output layer, and four hidden Dense layers. Each hidden layer utilized the ReLU activation function and varied in size based upon the size of the input, which depended on the board configuration. For example, on a board with four paddles and four balls, the state of the system at any given time step would contain 24 pieces of information: four paddle coordinates (x_p, y_p), four ball coordinates (x_b, y_b), and four ball speed vectors ($\dot{x}_b$, $\dot{y}_b$).

The final output layer was sized to provide Q-values for each possible action pertaining to each paddle on a given team. More specifically, since each model was trained to provide actions for one team, the output size was twice the number of paddles on that team. In the example above, the output would be of size four, each position representing the Q-value for moving up or down for a given paddle. Figure 2 shows the network architecture adopted for the training.

VI. Reward Structure

In order to train our paddles in a way that will facilitate an actual improvement in game-play, we had to establish a decent reward structure to incentive *good* moves versus *bad* moves. Generally speaking, in MARL, rewards can be applied jointly, as one global reward, or independently, rewarding each agent accordingly. Since our *FoosPong* game was set up such that each team's paddles have the same abilities, in terms of actions available, and the same goal, we decided on a global reward structure for each team.

Our reward structure was set up as follows:

1) +20 points when team scores
2) -20 points when opponent scores
3) +20 additional points when a team's paddle hitting a ball directly results in a goal for that team.
4) +2 points for each ball in team's half of the court moving away from their goal line
5) -2 points for each ball in team's half of the court moving toward their goal line

Rewarding goal scores and penalizing opponent scores works to incentivize scoring while also incentivizing defense against the opponent scoring. The additional +20 for paddle hits was added to ensure that individual paddle movements when hitting a ball are rewarded each time they directly result in a goal, since the time period between paddle hits and scored goals may be multiple time steps. The purpose of this was to incentivize emergent behavior in terms of hitting the ball in creative ways, like at sharp angles to steer and accelerate the ball instead of simply bouncing it randomly.

Lastly, the small +/-2 rewards for ball movement in a team's half court allowed the team to learn rewards for simply hitting a ball and changing the direction back toward the other team. Without this addition, the vast majority of time steps provided zero reward, resulting in minimal training value at that time step. We saw this as a drawback to our training pipeline since we sampled randomly from the experience replay for each training session, and as such, we wanted to ensure maximum training value in each collection of samples. Another way around this could have been selective memory addition to the experience replay, which is an avenue of exploration for future work.

VII. Experimentation and Game-play

Our intention with this project was to create simple yet definitively chaotic game environment that would leverage both cooperative and competitive aspects between RL agents. Thus, we conducted the majority of our testing in a 2v2 setup with four balls. From here, two main lines of effort were taken: training against a heuristic (we'll call this "heuristic-play"), and training against other RL agents (i.e. self-play).

A. Heuristic Play

In this approach, the opponent team consisted of paddles driven by a hard-coded heuristic, whose y coordinates were changed to match (as well as possible, with the speed of the paddle) the y coordinate of the nearest ball, i.e. a team of ball-followers. The rationale (at least, in our initial training) was to use this to circumvent the large amount of training that is typically required for appreciable results with self-play. [14] Additionally, this has become a fairly common approach when training RL pong models. [13] We hypothesized that this approach would quickly push our models to a basic competency in the game, but would have significant limitations in training the model to the point where it could beat the heuristic (or even a person, for that matter).

B. Self Play

With this approach, as was showcased in OpenAI's hide-and-seek model [4], we intended to leverage the lessons learned from rapid-prototyping with the heuristic play, while simultaneously training longer with fewer runs. In pinning each RL team against one another, we hypothesized that we would observe deeper, 'more intelligent' behaviors than we would from an explicitly heuristic-trained model, as each team would be able to explore more creative strategies without being overly punished by the high-scoring heuristic.

Additionally, we explored initialization of the two self-play RL teams with individually-pretrained heuristic-play teams, which yielded some interesting outcomes.

C. Miscellaneous Other Tests

In perhaps what was a 'too-many-knobs-to-turn' scenario, we experimented with too many game and DQN hyperparameters to list out here. Some of the more poignant results came from changing number of paddles and paddle/ball speed. Interesting too were the effects of changing the number of hidden neurons in the DQN, as well as using the Huber loss [15] instead of Mean-Squared Error (MSE).

VIII. Results and Discussion

A. Performance Analysis

Featured below in Figure 3 are results from one sample training run from heuristic-play with the standard model (2 paddles per team, 4 balls). In this particular run, one of the RL paddles became 'lazy' and glued itself to the wall, not attempting to take any actions past roughly episode 200.

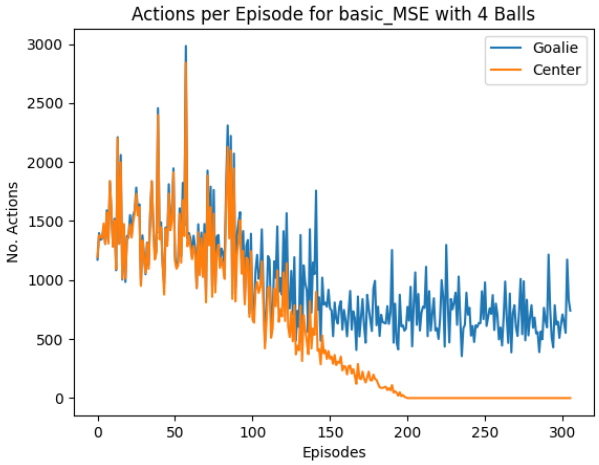

Fig. 3: Actions per episode of standard heuristic-play model with MSE loss

Additionally, below in Figure 4 are results from one sample training run from an identical heuristic-play with the Huber loss in place of MSE. It is clear that both RL paddles maintain consistent motion throughout.

Exploring the plots of cumulative actions (Figures 5 and 6) and ball hits (Figures 7 and 8) over these two runs, there is a

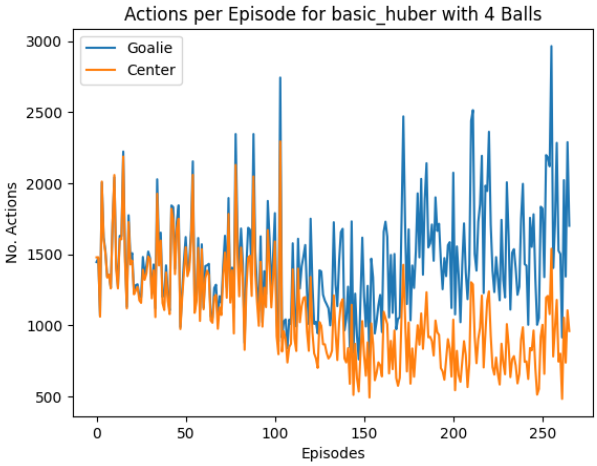

Fig. 4: Actions per episode of standard heuristic-play model with Huber loss

marked decrease in actions per episode taken past episode 100 for both training variations. These plots show us essentially the same information; note that the 'goalie' and 'center' positions described proximity to each team's goal, and the LEFT team was heuristic (while the RIGHT team were RL agents).

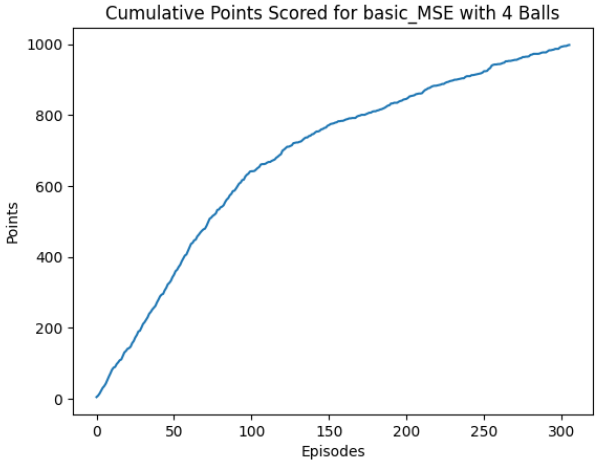

Fig. 5: Cumulative points scored by standard heuristic-play model with MSE loss

These two runs were very representative of most other heuristic play across many variations of the standard testing parameters. Thus, our hypothesis of the saturation of the RL team's learning against the much better heuristic proved to be consistent, though saturation at 100 episodes is earlier than our original estimates of 700-900 episodes.

Additionally, a set of heuristic-play models were run as a variation on the presented standard. Instead of training on a decaying percentage of heuristic actions (see above), several were trained on constant percentages (ϵ) to see if the heuristic-aided outcome could outscore teams that were

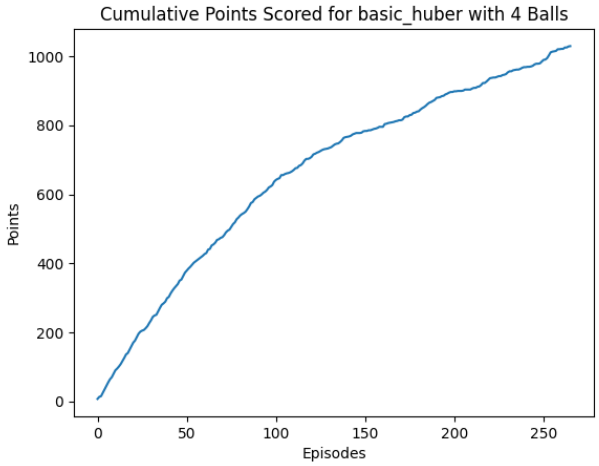

Fig. 6: Cumulative points scored by standard heuristic-play model with huber loss

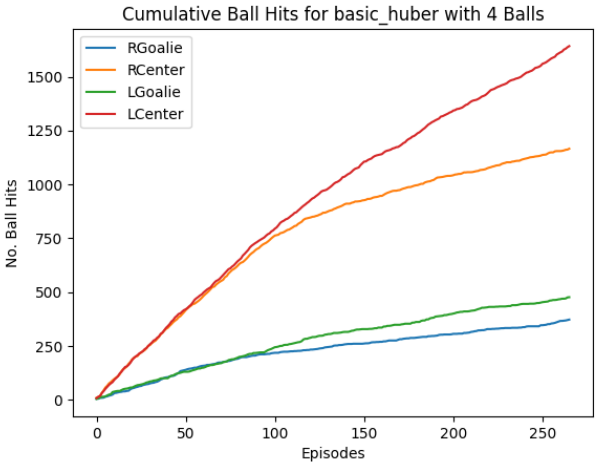

Fig. 8: Cumulative ball hits by standard heuristic-play model with huber loss

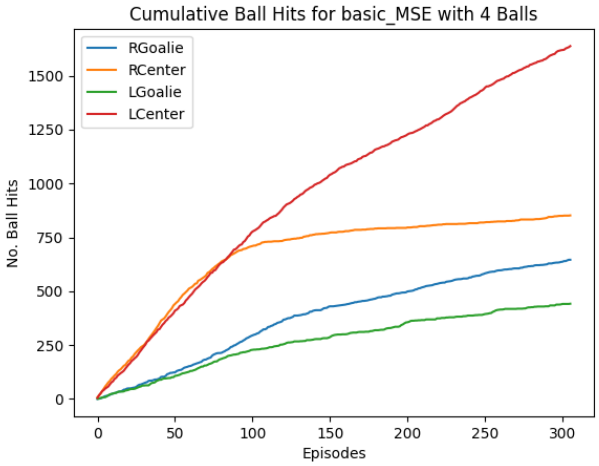

Fig. 7: Cumulative ball hits by standard heuristic-play model with MSE loss

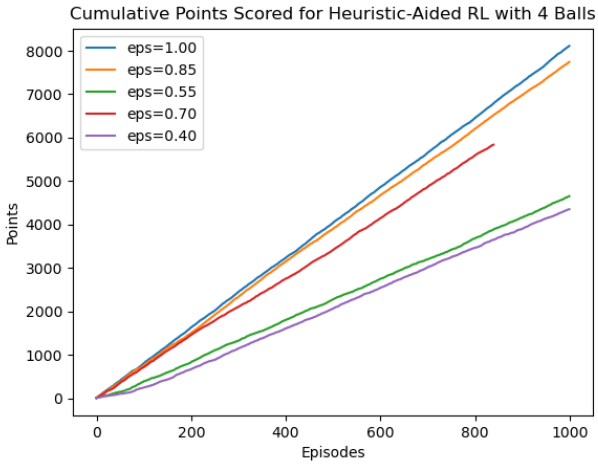

Fig. 9: Cumulative ball hits by heuristic-aided RL model with huber loss

purely-RL based. The plots of cumulative actions for each is presented below in Figure 9. In summary, performance appeared to be best for moderate epsilon values at 200 episodes, but unfortunately their performance declined to the same as standard heuristic-play by 1000 episodes.

To further test our saturation hypothesis, we tested the effect of a longer training duration in performance of RL teams, this time against a team of 'noisy heuristic' opponents, which took the ideal heuristic action 70% of the time and random actions the other 30%. We trained two models, one of which trained RL agents on the RIGHT, and one for the LEFT. The cumulative and average points scored over the course of training are included in Figures 10 and 11.

Lastly, we present the results from utilizing these 70/30 heuristic-play models to initialize a self-play training. The cumulative and average points scored by each of the RL teams are included in figures 12 and 13. In summary, it is clear that these plots reveal the underlying nature of self-play wherein both teams are pre-trained to the same level of proficiency at playing the game. Through self-play we note interesting emergent behavior and skills exchange that we will talk about in the next part.

B. Emergent Behavior

Showcased here are several notable strategies that originated in the RL team's trial-and-error with the FoosPong environment. The figures depicting these emergent behaviors are presented in Appendix A.

1) Jitter: The action space of our *FoosPong* game comprises of only two actions, *up* or *down*. As mentioned in section III, although we made a choice of not incorporating a third action of *staying still*, our agents learned to compensate for it using a learned skill, that we call *jitter*. The agents learn

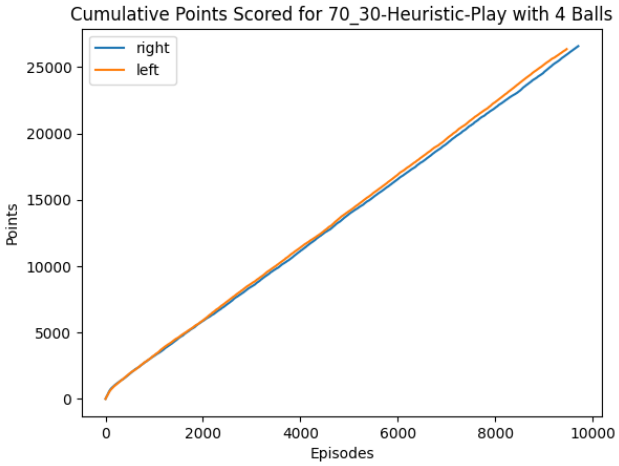

Fig. 10: Cumulative points scored by 70/30 heuristic-play model

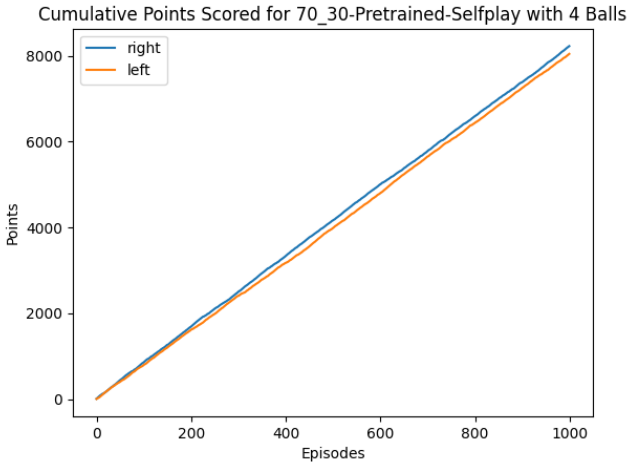

Fig. 12: Cumulative points scored by each team in pretrained self-play model

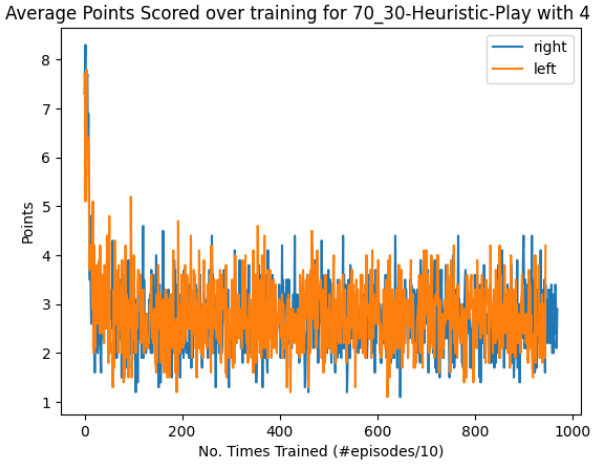

Fig. 11: Average points per training scored by 70/30 heuristic-play model

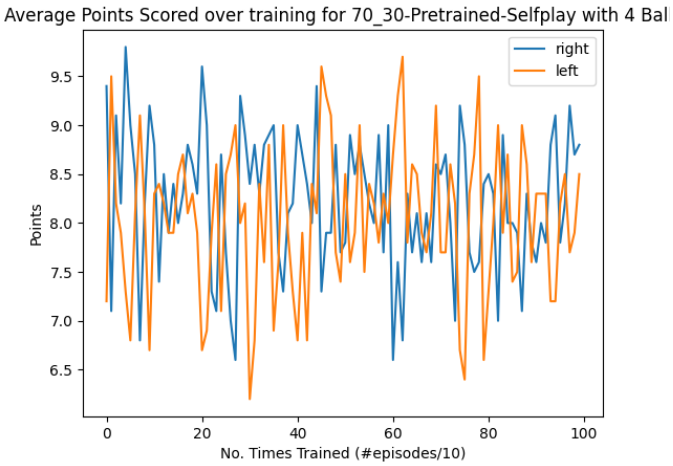

Fig. 13: Average points per training scored by each team in pretrained self-play model

to alternate between *up* and *down* rapidly in order to *stay still*. It can be seen to be an emergent control behavior learned by the agents to stabilize their own positions in the board. (Figure 14)

2) Ready-Rest: Although the agents are not levied with costs for moving, this emergent behavior was interesting to observe. We call this the *ready-rest* behavior, wherein the agents stick to the top and/or bottom walls of the board when no balls are being shot towards them. As soon as there is a ball in the vicinity, they spring out of their *rest* positions into the *ready* mode – moving to block the balls. An interesting behavior is that the two paddles seem to **cooperate** and thereby stick to opposite walls so as to be *ready* to efficiently block the balls while springing back from *rest*. (Figure 15)

3) Load and Shoot: This behavior was seen to be a perfect example of **cooperation** between agents in the same team. The paddles on the same team were seen to bounce a ball off of each other several times (rallying among each other), thereby speeding up the ball. Then, the front paddle would move away, thereby making way for the ball to shoot at the opponents at high speeds (Figure 16). This is by far the most interesting emergence that we observed in our agents that they learned during single-team pre-training and implemented during self-play (i.e. multi-team training) against teams that possibly did not learn it during single-team pre-training. Furthermore, during self-play we observed the team which had not learned this skill during pre-training, learned it during self-play from the other team. This was a clear demonstration of skills exchange through **co-evolution**.

4) Slide and Slice: During normal play the paddle agents seem to be blocking the ball normally by arriving at the expected ball position before hand and allowing the ball to

bounce off the paddle surface. However, through our training we observed yet another interesting emergence wherein the paddles seemed to time their movement to block the ball such that they were able to *slide and slice* the ball as they blocked it, thereby speeding the ball as it shot towards the opponent team. (Figure 17)

5) Corner Chop: This emergent behavior was first observed when our agents were being trained against the heuristic model. As the episodes progressed in training and our agents played fully based out of their trained model (and the heuristic component fully decayed out) we observed that the expert opponent team was able to chop the balls in a way that hit the paddle corners instead of the edges, thereby shooting towards our trained agents are fast speeds, rapidly bouncing back and forth between the top and bottom walls of the game environment, and in turn confusing our agents. However, eventually we noticed our agents could effectively start tracking even the rapid change of directions of the incoming balls and interestingly even learned to perform the *corner chops* themselves (Figure 18). This was a scenario wherein our agents learned to mimic even such interesting behavior from their expert opponents. This behavior would be an example of a skill learned purely from single-team training.

With these emergent behaviors having been observed in our game-plays, we have been able to demonstrate cooperation and co-evolution among RL agents and thus, a naive emergent multi-agent autocurricula in *FoosPong*.

IX. Conclusion and Future Work

The DQN model was trained for the *FoosPong* game in a centralized fashion and was tested decentrally in a multi-agent scenario. The model performed well against a randomized agent and fared decently against a well defined heuristic model. Finally, the trained agents were allowed to self-play for training and testing.

In future work we'd like to advance Self Play and try to establish more consistent emergent behaviors. Although the current model performs decently against the heuristic model, it does not consistently match or outperform it. Hence, our goal is to beat the heuristic model by increasing the network density and making modifications to the hyperparameters. We also plan to convert the model to a Spiking Neural Network (SNN) to create a biologically-realistic model of neurons to carry out better computation. A third action of 'Stay' can be added along with the 'Up' and 'Down' actions to improve the stability of the paddles during gameplay. This might complicate the network and might call the need for more hidden layers.

References

[1] N. Haber, D. Mrowca, S. Wang, L. Fei-Fei, and D. L. Yamins, "Learning to play with intrinsically-motivated, self-aware agents," *Advances in Neural Information Processing Systems*, p. 8388–8399, 2018.

[2] K. Kim, M. Sano, J. D. Freitas, N. Haber, and D. Yamins, "Active world model learning with progress curiosity," *International conference on machine learning*, pp. 5306–5315, 2020.

[3] K. Ndousse, D. Eck, S. Levine, and N. Jaques, "Multi-agent social reinforcement learning improves generalization," *CoRR*, vol. abs/2010.00581, 2020. [Online]. Available: https://arxiv.org/abs/2010.00581

[4] B. Baker, I. Kanitscheider, T. Markov, Y. Wu, G. Powell, B. McGrew, and I. Mordatch, "Emergent tool use from multi-agent autocurricula," *arXiv preprint, arXiv:1909.07528*, p. 8388–8399, 2018.

[5] H. X. Pham, H. M. La, D. Feil-Seifer, and A. Nefian, "Cooperative and distributed reinforcement learning of drones for field coverage," 2018.

[6] D. Bloembergen, K. Tuyls, D. Hennes, and M. Kaisers, "Evolutionary dynamics of multi-agent learning: A survey," *Journal of Artificial Intelligence Research*, vol. 53, pp. 659–697, 08 2015.

[7] J. K. Gupta, M. Egorov, and M. Kochenderfer, "Cooperative multi-agent control using deep reinforcement learning," in *Autonomous Agents and Multiagent Systems*, G. Sukthankar and J. A. Rodriguez-Aguilar, Eds. Cham: Springer International Publishing, 2017, pp. 66–83.

[8] D. Bloembergen, "Multi-agent learning dynamics," Ph.D. dissertation, Maastricht University, 2015.

[9] K. D. Julian and M. J. Kochenderfer, "Distributed wildfire surveillance with autonomous aircraft using deep reinforcement learning," *Journal of Guidance, Control, and Dynamics*, vol. 42.8, pp. 1768–1778, 2019.

[10] D. Baldazo, J. Parras, and S. Zazo, "Decentralized multi-agent deep reinforcement learning in swarms of drones for flood monitoring," in *2019 27th European Signal Processing Conference (EUSIPCO)*, 2019, pp. 1–5.

[11] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, "Trust region policy optimization," *arXiv cs.LG*, 2017.

[12] M. Guerzhoy and D. Begun, "Ai engines for pong," 2016. [Online]. Available: http://www.cs.toronto.edu/~guerzhoy/niftypong/

[13] E. A. O. Diallo, A. Sugiyama, and T. Sugawara, "Learning to coordinate with deep reinforcement learning in doubles pong game," in *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, 2017, pp. 14–19.

[14] "Ai learns how to play pong - reinforcement learning," 2020. [Online]. Available: https://www.youtube.com/watch?v=CBb0Q3GCHh0

[15] P. J. Huber and E. M. Ronchetti, "Robust statistics," *Wiley, N. Y.*, 1981.

Appendix

The source code and the GIFs of the emergent behavior are presented in the GitHub repository[1].

The following figures (14-16) show the emergent behavior as mentioned in section VIII-B.

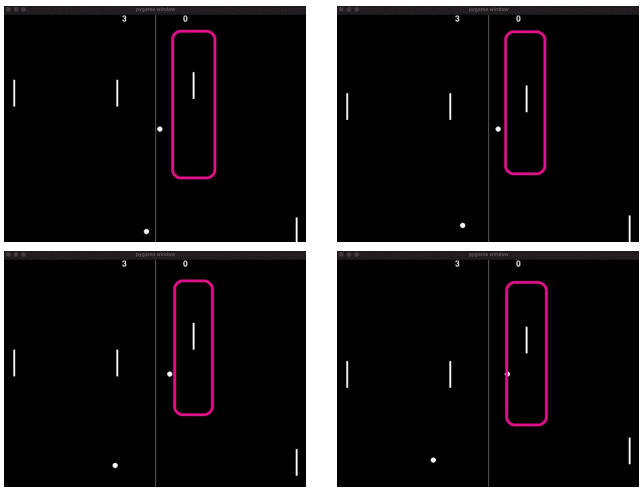

Fig. 14: Jitter

[1]https://github.com/gnitish18/Multi-Robot_Reinforcement_Learning

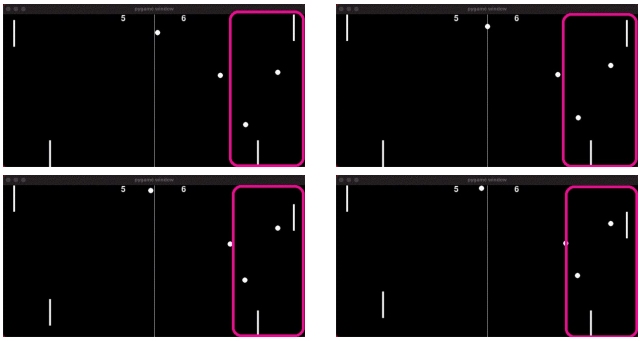

Fig. 15: Ready-Rest

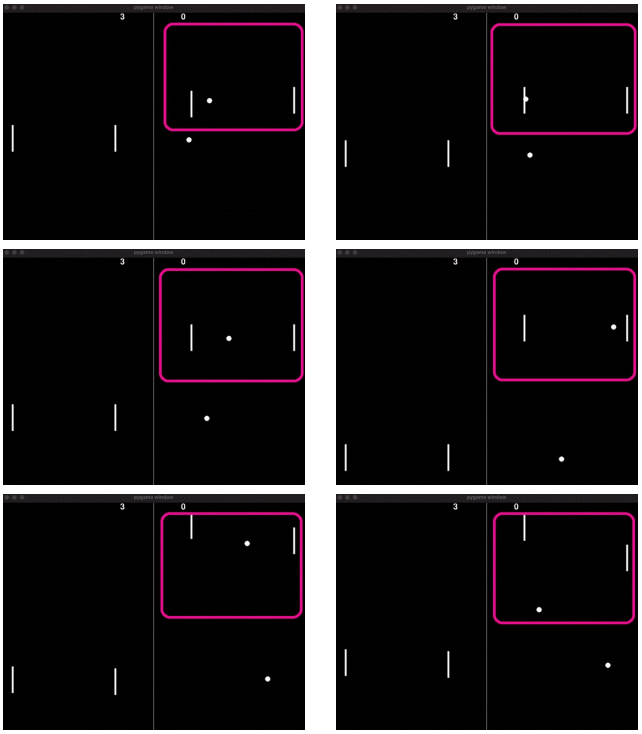

Fig. 16: Load and Shoot

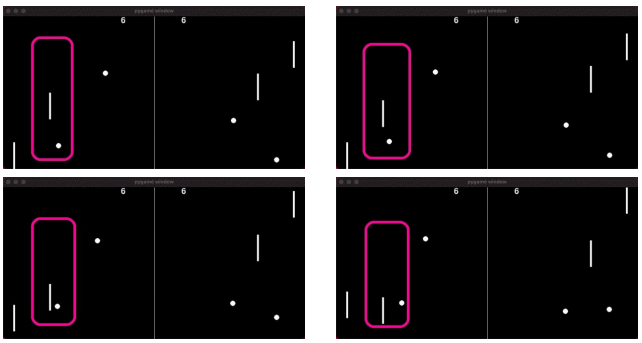

Fig. 17: Slide and Slice

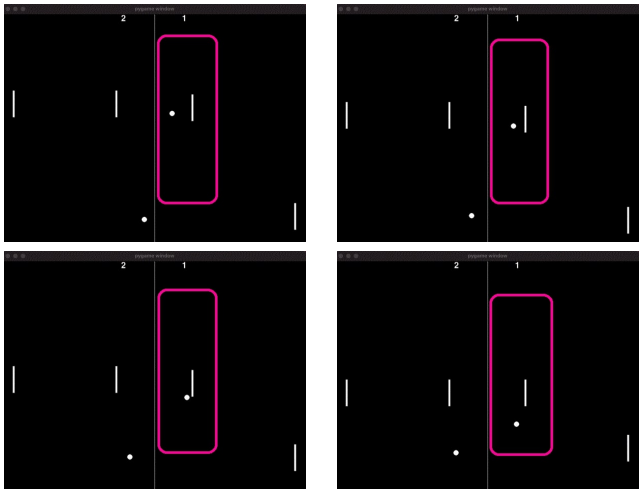

Fig. 18: Corner Chop