PP5 Report

Hyperparameter Tuning

We have trained the agent in 3 phases:

- Against a Random Opponent
- Against a Smart Opponent
- And recursively against each other (this will need to be continuous)

We started off with values suggested for PPO configurations for the hyperparameters:

```
"max_timesteps_per_episode": 300,
"n_updates_per_iteration": 15,
"lr": 3e-5,
"clip": 0.2,
```

We have defined custom hyperparameters here, specifically:

`break_after_x_continuous_win_percent` - The agent will stop learning process after it beats the opponent in the **last** `max_num_of_episodes_to_calculate_win_percent * how_many_consecutive_wins_to_break` games.

Each `rollout` will require `episodes_per_batch` episodes to be collected.

`step_reward_multiplier` - If we want to lessen the impact of step rewards in the dense environment.

Against Random, Against Smart Opponent with Center Weight = 1, 1.5 and 2

```
"episodes_per_batch": 20,
"n_updates_per_iteration": 20,
```

We increased the `n_updates_per_iteration=20` and episodes per batch to improve convergence time.

Against Recursive Learning

Due to forgetting, we needed to change the following parameters:

```
"break_after_x_continuous_win_percent": 75,
"n_updates_per_iteration": 25,
"episodes_per_batch": 50,
"clip": 0.2,
"step_reward_multiplier": 0.9,
```

- We also had to make the `step_reward_multiplier=0.9` as we did not want the games length to be a factor that makes the agent stop exploring. Plus we added more `episodes_per_batch` to improve learning. Finally, we used `break_after_x_continuous_win_percent=75` to prevent overfitting.

Saved Model

We save the model by saving the actor and critic networks:

```
torch.save(
    self.actor.state_dict(),
    f"{path}actor.pth",
)

torch.save(
    self.critic.state_dict(),
    f"{path}critic.pth",
)
```

and is loaded using:

```
self.actor.load_state_dict(torch.load(f"{model_path}/actor.pth"))
self.critic.load_state_dict(torch.load(f"{model_path}/critic.pth"))s
```

Sparse and Dense

- We did not have to make any changes between the sparse and dense environments as PPO was able to train both the environments effectively for `full_random`, `smart_random_1` and `smart_random_1_5`.
- After training we saved both the sparse and dense models in 2 separate files. Our agent files chooses the model to use while playing based on the environment.