Project #3

Rocking Rock Paper Scissors

Deadline: 13.01.2025, 12:00h

! **Note:** The code you get as a starting point for the project (see below) might still be changed slightly to improve the game. Also, if you encounter strange behavior, there might be a bug. Send me (Daniel) a mail if there is something fishy.

What is the project about?

In this project you will train a reinforcement learning agent in a simple game. In the Tuesday lecture after the project deadline (14.01.2025) we will have a competition between different groups. Participation is mandatory.
The game is a modification of the famous child's game *rock paper scissors* played on a two-dimensional playing field.

Rules

- Each player gets one playing piece somewhere on the grid. The playing piece can be either a rock, a paper or scissors.
- The players take turns. In each round, the player whose turn it is can either make a move (one cell up, down, left or right) or convert its piece.
- A conversion converts the piece to the next type in the repeating cycle (R $\rightarrow$ P $\rightarrow$ S). Conversion is only possible if the distance to the opposing piece is at least two moves.
- If the opposing pieces meet, they engage in a rock-paper-scissors fight, where rock beats scissors, scissors beats paper and paper beats rock. The loosing piece is removed from the board and the player lost the match.
- If two pieces are of the same type, they cannot move to the same field. Moves are bound by the borders of the playing field.

- There are holes in the playing field. Moving onto these holes also kills the piece and causes the respective player to lose.
- In the bottom left corner there is a goal field. Walking on that wins the round, but killing the opponent gains more points.

Code framework

Under this link

https://itp.uni-frankfurt.de/~nevermann/teaching/rps.zip

you find a starting point for working with the game. The repository includes

- The game itself (`rps_game/game.py`).
- An OpenAI gymnasium environment for the game (`rps_game/env.py`).
- Opponent classes (`rps_game/opponent.py`). They manage the moves of the opponent player during training.

Game API

The game is implemented in the `Game` class. When constructing a new game, one must pass the `size` of the square playing field and optionally may pass the number of holes `n_holes`.
The `players` (instances of the `Player` class) store a position (`pos: NDArray[int]`), a piece type (`piece: Piece`) and a unique ID (`id: int`).

Attributes

`Game.board`
The board as a NumPy array, where the game entities are encoded by

empty	rock / paper / scissors	hole	goal
`0`	`1 / 2 / 3 + player.id * 3`	`-2`	`-100`

`Game.size`
The size of the playing field.

`Game.next_player`
The ID of the player whose turn it is.

`Game.spawn_goal_field`
If the game contains a goal field or not.

Methods

`Game.reset()-> None`
Resets the playing field randomly.

`Game.is_game_over()-> bool`
If the game is over.

`Game.set_static_hole_config(positions: NDArray)-> None`
Set a static hole positions that persist resets.

`Game.act(self, action: Action | None)-> None`
Performs an action for the `Game.next_player`. If `None` is passed, no action is taken.

`Game.__str__()-> str`
Returns a string representation of the game board.

Gymnasium environment

OpenAI gymnasium provides an API standard for reinforcement learning. The package is well documented. Most importantly, an environment defines an `observation_space`, an `action_space` and methods for `reset` and `step`, where `step` expects a number representing an action in the `action_space`, performs that action in the environment and returns, alongside other things, a new observation and a reward for the action. Every time the step-function is called, player 0 performs the passed action and player 1 performs acts in response in accordance with the opponent class.

The `render_mode` controls if and how the observations are rendered, when `RPSEnv.render` is called. For `render_mode == "human"` a graphic visualization is presented (not recommended during training).

If you feel like you have to modify the environment, you can, but make sure that you don't break the API.

Hints on training your agent

You can, if you seek the challenge, implement a deep reinforcement learning algorithm yourself. Alternatively, Stable Baselines3 provides implementations for different algorithms, fully compatible with gymnasium environ-

ments.

Note, that in the game, not all possible actions are necessarily allowed, given the current game state, (e.g. a piece may not leave the board boundaries or conversion with another piece close by). The environment already provides an action mask (`RPSEnv.compute_action_mask`) that indicates valid moves.

If you encounter problems during training, simplify! Use a game without holes and a trivial opponent. Then, once your agent is learning, gradually increase complexity.

When thinking about what opponent to use during training, trivial opponents are probably a good start. Later you might want to employ self-training, where the agent is trained against a previous version of itself. You can find examples how to implement that online.

Minimum requirements

Your project should fulfill some minimum requirements:

- Train a reinforcement learning agent to play the 2D rock paper scissors game on a 4×4 grid with two holes in the static hole configuration predefined in the environment and a goal field using an algorithm of your choice.
- Document your training efforts, compare different approaches you might take (e.g. a plot of the reward over training steps, the mean episode length or the success rate over episodes).
- Submit the code and a documentation in a suitable format. Furthermore, include a script to enable participation in the tournament – details are enclosed in the `README` file found in the code repository.

Optional

- Increase game complexity: Add a second playing piece per player, implement barriers in the game or consider randomly placed holes. For this, you will most likely have to modify the game and the environment. In this case, provide functionality such that the game with your modifications can be matched up against other teams in a 'PRO'-tournament. You can also coordinate with other teams that do the same modifications and fight against them on equal terms.