

AI challenge

Artificial Intelligence

WS 2020

Verena Haunschmid
Jan Schlüter
Rainer Kelz
Florian Schmid



Figure 0.1: A randomly generated Pommerman arena with unicorn design

General Information

This is the instruction sheet to get you started with the AI challenge. Participating is not obligatory, but can earn points worth as much as one exercise and you can compare your programming and AI skills with those of your colleagues.

Your task is to implement an intelligent agent in a small game called Pommerman. Official competitions in the Pommerman environment were already held at the NeurIPS conferences in 2018 and 2019. For the competition in this course we use a slightly simplified version of the Pommerman environment. A description of the game can be found on the official website ¹.

¹<https://www.pommerman.com>

1 Pommerman Setup

First, you will need to setup your programming environment. We use the conda environment for the challenge setup. If the instructions we provide here are not enough for you, you can always consult the conda documentation at <https://docs.conda.io/projects/conda/en/latest/index.html> or use the provided instruction videos on moodle.

Whenever you see text such as the following:

```
$ command param0 param1
```

We assume you have access to a sane shell environment (i.e. bash) and a nice terminal emulator, in which you can type these commands. The symbol \$ represents your command prompt, and it can look different for your shell.

We will now assume that you have Miniconda already **installed** and start to setup the conda environment and install the Pommerman package:

- Create a new environment:

```
$ conda create --name py37_pommerman_student python=3.7.4
```
- Activate your virtual environment:

```
$ conda activate py37_pommerman_student
```
- Note that you can configure your created virtual environment to be used as default interpreter for a project opened in PyCharm. For instructions have a look at the document Setup PyCharm in moodle.
- For most shells, your command prompt should now have changed, to indicate that the virtual conda environment named py37_pommerman_student is now **active**.
- Download the file pommerman_students.zip (contains the framework) from the moodle course page.
- Unzip the file pommerman_students.zip into an empty directory named pommerman_students.
- You should find the following directory structure (showing only relevant files):

```
\---pommerman_students
|   requirements.txt      <- requirements of pommerman (do not touch)
|   setup.py              <- install the pommerman package (do not touch)
|
+---docs                  <- pommerman documentation (e.g. the game rules)
|   \---assets
+---examples
|       simple_ffa_run.py  <- runs a pommerman game
|
+---pommerman
|   +---agents            <- provides pommerman sample agents (e.g. SimpleAgent)
|   |
|   +---cli
|   +---envs
```

```

|   +---helpers
|   +---resources
|   \---resources_unicorn_style
\---student_agents      <- most relevant package for you
    +---groupXX          <- your own package (see next steps)
        |   |   setup.py  <- install your agent (do not touch!)
        |   |
        |   \---groupXX
        |       |   groupXX_agent.py  <- your agent module
        |       |   __init__.py
        |       |
        |       \---resources  <- location of resources you might need
        |
    +---heuristic_agent  <- an example: MCTS agent
        |   \---heuristic_agent
        |       \---resources
    +---learning_agent   <- an example: learning agent
        |   \---learning_agent
        |       \---resources
    \---very_simple_agent <- very basic agent to get you started
        \---very_simple_agent

```

- In your shell, navigate to the base directory (= pommerman_students) of pommerman.
- Issue the following command:
`$ pip install .`
This installs the pommerman package and its dependencies into your active conda environment.
- If you plan to have a look at the learning agent or want to develop a learning agent yourself, then you should install Pytorch additionally by running the command:
`$ pip install torch==1.6.0+cpu torchvision==0.7.0+cpu`
`-f https://download.pytorch.org/whl/torch_stable.html`

This is the Pytorch version we use to run the challenge. To train an agent you will probably need a Pytorch version with CUDA enabled, but please make sure that your agent is compatible with the version listed above!

- **Stick to the packages listed in the file pommerman_students/requirements.txt + the pytorch version from above. This setup will be used to run your agent on the challenge server.**
- You are now ready to run your first game of Pommerman by running the following commands starting from the pommerman base directory:
`$ cd examples`

```
$ python simple_ffa_run.py
```

You should now see two SimpleAgents competing against each other. If you want to stop the game, press Ctrl+C in the shell.

2 Agent Setup

This Section provides you with information on how to get started implementing your own agent. Although the game is designed very intuitively, it might be very helpful to get in touch with the rules of Pommerman that can be found in the file `pommerman_students/pommerman/README.md`, which you can also look up online: <https://github.com/MultiAgentLearning/playground/tree/master/pommerman>. Some small adaptations to these rules are made that can found in Section 4 Competition Rules.

- Your starting point to implement an agent is the directory `pommerman_students/student_agents/groupXX`. This directory is structured as follows:

```
+---groupXX      <- your agent directory
|   |   setup.py  <- used to install your agent (do not touch!)
|   |
|   \---groupXX   <- the python package that contains your agent
|       |   groupXX_agent.py <- the module that contains your agent class
|       |   __init__.py
|       |
|       \---resources <- will stay empty in most cases, can be used to store a model
```

- Firstly, in order to stick to the naming conventions the symbols 'XX' have to be replaced by your group number. To do this automatically, **navigate** to the directory `pommerman_students/student_agents` and run the following command:

```
$ python create_agent_folder.py --gn XX
```

where XX must be replaced by **your own group number (e.g. 05)**. This will create a copy of the folder `groupXX` according to your group number. Please note that whenever we talk about **your** agent, we refer to the directory containing your group number (e.g. `group05` or `group19`), but we will use the name `groupXX` as a placeholder.

- You can then install your agent into your active conda environment by running:

```
$ cd groupXX
$ pip install -e .
```

Make sure that you use your own group number instead of 'XX'!

- Verify that your agent was installed correctly by running:

```
$ cd ..
$ python check_submission.py --gn XX
```

Again use your own group number instead of 'XX'

- If the last step did not give an error message, you have successfully setup your agent. You can now import it as package and instantiate it in your python programs by running:

```
from groupXX import groupXX_agent
my_agent = groupXX_agent.GroupXXAgent()
```

Again replace 'XX' by your group number! You can test this by plugging your agent into a game executed by `simple_ffa_run.py`.
- *Hint:* If you want to have multiple agents in parallel, you can create another copy of the `groupXX` directory by running:

```
$ python create_agent_folder.py --gn XX --t a
```

which will create a directory `groupXXa`. Again navigate to the newly created `groupXXa` directory und run:

```
pip install -e .
```

to install your test agent into your conda environment.

3 Improving your agent

In the last Section you set up your agent and installed it as a package. In the tournament, before every game your agent (located in module `groupXX_agent.py`, class `GroupXXAgent`) is instantiated by calling the constructor, which means you do some necessary initializations inside `__init__(self, *args, **kwargs)`. The core of your agent is the `act(self, obs, action_space)` method that is called for every step of the environment. `obs` is the observation space, which represents a state of the environment. The second input is `action_space` that contains all the actions that are possible in Pommerman. Both components are described here: <https://github.com/MultiAgentLearning/playground/tree/master/pommerman>. and in the module `groupXX_agent.py`. (Don't forget that `XX` stands for your group number).

- Right now your agent behaves randomly, which means it will not survive long if you start a game by plugging it into `simple_ffa_run.py`. Your goal is to improve its policy (the way the agent chooses its moves inside the `act` method).
- You can either decide to implement an agent that is based on *heuristics* or a *learning agent* that uses a reinforcement learning algorithm to train a neural network. The easier and less time consuming variant will be to program a heuristic based agent. If you choose to implement a reinforcement learning agent, be prepared that you will have to invest some time. Nevertheless, if you are interested in this topic, it will be a very good opportunity to get in touch with it.
- If you want to implement a **heuristic agent**, have a look at the directory `student_agents/heuristic_agent`, which provides you with an example. The agent included in the Pommerman framework itself (`pommerman/agents/simple_agent.py`) might also be very helpful.
- If you choose to implement a **learning agent**, have a look at the directory

student_agents/learning_agent, which provides you with an example how a learning agent can be structured. This sample learning agent uses the reinforcement learning algorithm *DQN* (Deep Q-Networks). Note that using this algorithm as it is will not create a well performing agent, but you will at least notice that the agent learns not killing itself all the time.

- In **both** cases have a look at the directory student_agents/very_simple_agent. If you are interested in some advanced ideas of creating an agent, have a look at Section 6.

4 Competition Rules

The rules of the game are described on the official Pommerman website ² and the small adaptations (simplifications) we made are listed in this Section. Our competition is based on the "Free For All (FFA)" version of Pommerman. This means that the board is fully observable all the time (no fog), there are no teammates and no messages can be sent. Figure 4.1 sums up the most important information and introduces all the different objects that are encountered in Pommerman unicorn style.













Quick introduction to Pommerman unicorn style			
Bombs = Seeds Flames = Rainbows Wood = Cloud			
	The unicorn agents - you control them by defining their policy in the act()-method of your agent		Passage - a tile the agent can walk on
			Cloud - solid tile, but can be removed by Seeds
			Rigid - solid tile that can not be removed by Seeds
			ExtraSeed - increases agent's ammo
	Seeds layed by unicorns - they explode after 10 ticks and convert to Rainbows		IncrRange - increases the range of the Seeds
	Rainbows - your agent has to avoid them, Clouds and Items are removed by Rainbows		CanKick - agent can move Seeds
<ul style="list-style-type: none"> • 1 vs. 1 games • Winner: last unicorn standing (no winner in all other cases) • agents move concurrently (bounce back if step leads to illegal field) • max 500 ms per move • max 500 MB memory usage • interaction with the environment using act()-method of agents 			

Figure 4.1: A quick introduction and overview of Pommerman unicorn style

²<https://www.pommerman.com/about>

The rules in more detail:

- The official Pommerman competition takes place with 4 players per game. We decided to reduce the complexity by only letting **2 players** compete in a game. Note that these two players play their moves **concurrently**, not one after the other.
- The official competition restricts move time to 100ms. We decided to increase it to **500 ms**, which gives you the chance to use longer lasting algorithms. Furthermore, we restrict the maximum size of main memory your program is allowed to use to **500 MB**. Exceeding the time limit is equal to sending a Stop move, **exceeding memory loses the game**.
- To ensure equal circumstances for all submissions we will **only use the CPU (Intel Core i5-6400)** to run the agents. 500 ms should still be enough to run a forward pass if an agent uses a neural network.
- You are not allowed to use other packages than those that are installed in your conda environment that you created in Section 1. Also do not change the file `setup.py` of both, the pommerman package and your agent's package.
- There will be multiple chances for you to submit an agent. Depending on the number of submissions we receive, we will then either host a round tournament, or we will split the field into groups. The best agents of each group then propagate to the next stages to play within the group of winners and so on. **A point is only received** if one agent wins, while the opponent is destroyed (in case of a draw both agents receive 0 points). As **secondary evaluation** method we use the total number of items collected by the agents during the games.

5 Submission

- Naming conventions: do not manually change names of the files that are inside the directory `groupXX` (XX is your group number). However, creating new modules with arbitrary names is fine.
- If your agent needs to access resources, like a neural network model, place them in the folder `groupXX/resources` and access them exactly as shown within the `learning_agent` package!
- **Before handing in your submission**, check if your agent works as expected by navigating to the directory `student_agents` and running:

```
$ python check_submission.py --gn XX
```

where XX is your group number

- You can either use **relative imports** or **imports from your installed package**. For example, if you create a module `util.py` inside the package `groupXX`, then you can import it inside the module `groupXX_agent.py` using either `"from . import util"` or

"from groupXX import util". You can **not** use "import util".

If the check_submission.py script from above gives no error, then everything is fine!

- After running check_submission.py successfully, **zip the whole directory** groupXX (not only the package!) and upload the zip file to moodle. Your submission should look like this (XX is your group number):

```
\---groupXX.zip
  \---groupXX
    |   setup.py
    |   __init__.py
    |
    \---groupXX
      |   groupXX_agent.py
      |   __init__.py
      |   <optional modules here>
      |
      \---resources
          | <optional resources here>
```

- Attention: Moodle uploads are limited to 20 MB!
- Published results, including replay videos can be found here: <http://rainbows.cpi.jku.at/>.

6 Literature to get you started

This Section provides you with some advanced ideas and additional resources that you can use for your agent.

- The **official Pommerman paper** [1]: *Pommerman: A Multi-Agent Playground*

If you want to implement a heuristic agent and use some advanced concepts like **Monte Carlo Tree Search (MCTS)**, then you can have a look at this paper:

- A suggested heuristic + combination with Monte Carlo Tree Search (MCTS) [2]:
A hybrid search agent in Pommerman

If you want to implement a learning agent a good starting point is **DQN** (relatively simple and "easy" to understand compared to other algorithms, but might not be the best choice for Pommerman). A more promising choice for Pommerman is the RL algorithm **PPO** (Proximal Policy Optimization), but this one is more complex than DQN.

- *Reinforcement Learning - an intuitive overview* [3]
- DQN theory [4]: *Playing Atari with Deep Reinforcement Learning*
- DQN Implementation [5]: *Reinforcement Learning (DQN) Tutorial*
- PPO theory [6]: *Proximal Policy Optimization Algorithms*

- PPO theory more practically explained [7]: *Proximal Policy Optimization*
- PPO implementation [8]: *Github Repository*

It is probably also helpful to go through prior work in Pommerman:

- *Skynet: A Top Deep RL Agent in the Inaugural Pommerman Team Competition* [9]
- Skynet website [10]:
The Pommerman team competition or: How we learned to stop worrying and love the battle
- *Accelerating Training in Pommerman with Imitation and Reinforcement Learning* [11]
- In-depth analysis of an Action Filter [12]: *On Hard Exploration for Reinforcement Learning: a Case Study in Pommerman*

If you want to search for further ideas the following tags might be helpful:

Curriculum Learning, Monte Carlo Tree Search, Imitation Learning, **Action Filter**, Reward Shaping, Backplay

References

- [1] C. Resnick, W. Eldridge, D. Ha, D. Britz, J. Foerster, J. Togelius, K. Cho, and J. Bruna, “Pommerman: A multi-agent playground,” 2018.
- [2] H. Zhou, Y. Gong, L. Mugrai, A. Khalifa, A. Nealen, and J. Togelius, “A hybrid search agent in pommerman,” in *Proceedings of the 13th International Conference on the Foundations of Digital Games*, FDG ’18, (New York, NY, USA), Association for Computing Machinery, 2018.
- [3] R. Moni, “Reinforcement learning algorithms — an intuitive overview,” 2019. Last accessed 04 November 2020.
- [4] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, “Playing atari with deep reinforcement learning,” *CoRR*, vol. abs/1312.5602, 2013.
- [5] A. Paszke, “Reinforcement learning (dqn) tutorial,” 2017. Last accessed 04 November 2020.
- [6] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” *CoRR*, vol. abs/1707.06347, 2017.
- [7] OpenAI, “Proximal policy optimization,” 2018. Last accessed 06 October 2020.
- [8] I. Kostrikov, “Pytorch implementations of reinforcement learning algorithms.” <https://github.com/ikostrikov/pytorch-a2c-ppo-acktr-gail>, 2018.
- [9] C. Gao, P. Hernandez-Leal, B. Kartal, and M. E. Taylor, “Skynet: A top deep RL agent in the inaugural pommerman team competition,” *CoRR*, vol. abs/1905.01360, 2019.
- [10] C. Gao, P. Hernandez-Leal, B. Kartal, and M. E. Taylor, “The pommerman team competition or: How we learned to stop worrying and love the battle,” 2019. Last accessed 01 September 2020.
- [11] H. Meisheri, O. Shelke, R. Verma, and H. Khadilkar, “Accelerating Training in Pommerman with Imitation and Reinforcement Learning,” no. NeurIPS, pp. 1–10, 2019.
- [12] C. Gao, B. Kartal, P. Hernandez-Leal, and M. E. Taylor, “On Hard Exploration for Reinforcement Learning: a Case Study in Pommerman,” pp. 24–30, 2019.