CS 7642: Reinforcement Learning and Decision Making **Project #3**

Overcooked

1 Overview

For this project you will solve the multi-agent Overcooked environment (modeled after the popular video game). In this environment, you have control over 2 chefs in a restaurant kitchen who have to collaborate to cook onion soups. To cook a soup, the agents need to put 3 onions into a cooking pot, initiate cooking, wait for the soup to cook, put the soup into a dish, and serve the dish at a serving area. As you have acquired some experience in building reinforcement learning agents and analyzing and presenting results, we expect much of the project to be open-ended and self-directed. Your primary goal is to maximize the number of soups delivered within an episode on a variety of layouts ranging from fairly easy to extremely difficult. We also expect you to discover and analyze auxiliary goals or metrics that help you to achieve the primary goal.

Our expectation is that you have learned what is significant to include in this type of report from previous projects and the material that we have covered so far. It is thus up to you to define:

- The direction of your project including which aspect(s) you aim to focus upon.
- How you specify and measure such aspects.
- How to train your agents.
- How to structure your report and what graphs to include (in addition to the mandatory graphs discussed later).

Your focus should be on demonstrating your understanding of the algorithm(s)/solution(s), clarifying the rationale behind your experiments, and analyzing their results. You can use all the tools you have learned in the course so far, such as function approximation and reward shaping. The environment provides a reward shaping data structure that you are free to use. You may also design your own reward shaping in place of, or in addition to, this default setup. However, all algorithms and solutions used to solve the environment should be your own. Note that unlike the environment in Project 2 which has a continuous action space, we're dealing with a discrete action space here, so you may need to make modifications if you desire to reuse or augment your approach from P2.

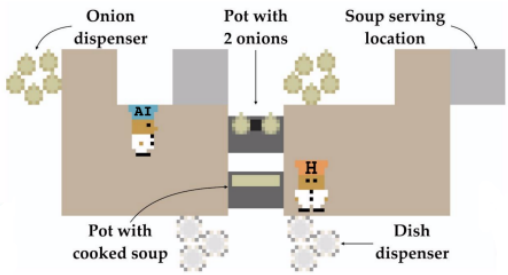

Figure 1: Visualization of the Overcooked environment. Carroll et al. 2019

2 Problem

2.1 Environment and Task

In this project, you will train a team of 2 agents to cook onion soups in a kitchen. The objective is always to deliver as many soups as possible within an episode. Each soup takes 20 timesteps to cook and delivering a soup successfully yields a +20 reward. Cooking a soup with less than 3 onions, dropping a soup on the

ground, or serving the soup on the counter (instead of the designated serving area) yields no reward and hinders progress as agents lose precious time (and starve customers). Episodes are truncated by the environment to a 400 timestep horizon with no termination conditions. You are not permitted to increase the 400 step horizon. You are asked to solve 3 distinct layouts of varying difficulty - `[cramped_room, coordination_ring, counter_circuit_o_1order]` as shown in the first, third, and fifth layouts of Figure 2 (layouts 2 and 4 in the diagram are not required) [1]. **Your task is to achieve a mean soup delivery count of ≥ 7 per episode in all three layouts using a single approach**. This means a single algorithm and a single reward-shaping function (if you utilize reward shaping). This also means a single set of hyperparameters (note that the number of episodes needed for training a particular layout is not considered a hyperparameter, and is something you can vary between layouts). The idea is to build an agent that does not have to be tweaked in order to solve any layout that is thrown at it. Having a constant set of parameters in a setting where environments are similar is important because it makes reproducibility much easier for people seeking to learn from your research. Note however that this does **not** mean you train only one agent (that would be extremely challenging!). You will need to train a separate agent for each layout. Note also that while some layouts can be solved by a single agent algorithm and don't require any collaboration, other layouts benefit significantly from collaboration and some may only be solvable via collaboration. **This means that a successful approach to solving all 3 layouts will likely require an explicit multi-agent approach.** We also expect you to develop your approaches and analyze the results by explicitly looking at multi-agent metrics (see Section 3).

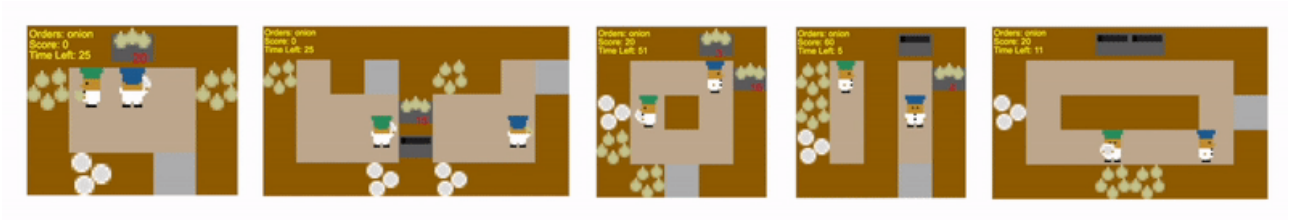

Figure 2: 5 example layouts. Of these you are tasked to solve 1, 3, and 5. From left to right they are named `[1 cramped_room, 2 asymmetric_advantages, 3 coordination_ring, 4 forced_coordination, 5 counter_circuit_0_1order]`. Carroll et al. 2019

2.2 State Space

This is a fully observable MDP and both agents have access to the full observation. Therefore, the state and observation spaces are equivalent. By default, the observations are provided as a 96-element vector, customized for each agent. The encoding for player $i \in \{0, 1\}$ contains a player-centric featurized view for the ith player, and is as follows:

[player_i_features, other_player_features, player_i_dist_to_other_player, player_i_position]

The first component, player_i_features, has length 46 and is described below. Note that if you add all the feature lengths in the specification below, you will get 36 instead of the expected 46. This is because the five features related to the "j^{th} closest pot" (having a combined length of 10) occur twice, once for each pot, and are concatenated together. Note also that none of our layouts contains tomatoes, so the fields corresponding to tomatoes will always be 0. In addition, counter information in this version of the environment is disabled, so the empty_counter variable should always be zero. Finally, layouts containing only one cooking pot will have the second pot's features zeroed out as well.

- p_i_orientation: one-hot-encoding of direction currently facing (length 4)
- p_i_obj: one-hot-encoding of object currently being held ([onion, soup, dish, tomato]) (all 0s if no object held) (length 4)
- p_i_closest_onion|tomato|dish|soup: (dx, dy) where dx = x dist to item, dy = y dist to item. (0, 0) if item is currently held (length 8)
- p_i_closest_soup_n_onions|tomatoes: int value for number of this ingredient in closest soup (length 2)

[1] The overcooked environment has dozens of layouts but for this project we will only be focusing on these 3.

- p_i_closest_serving_area|empty_counter: (dx, dy) where dx = x dist to item, dy = y dist to item. (length 4)
- p_i_closest_pot_j_exists: {0, 1} depending on whether j^{th} closest pot is found. If 0, then all other pot features are 0. Note: can be 0 even if there are more than j pots on layout, if the pot is not reachable by player i (length 1)
- p_i_closest_pot_j_is_empty|is_full|is_cooking|is_ready: {0, 1} depending on boolean value for j^{th} closest pot (length 4)
- p_i_closest_pot_j_num_onions|num_tomatoes: int value for number of this ingredient in j^{th} closest pot (length 2)
- p_i_closest_pot_j_cook_time: int value for seconds remaining on soup. 0 if no soup is cooking (length 1)
- p_i_closest_pot_j: (dx, dy) to j^{th} closest pot from player i location (length 2)
- p_i_wall_j: {0, 1} boolean value of whether player i has a wall immediately in direction j (length 4)

The remaining components of the observation vector are as follows:

other_player_features (length 46): ordered concatenation of player_j_features for $j \neq i$

player_i_dist_to_other_player (length 2): [player_j.pos - player_i.pos for $j \neq i$]

player_i_position (length 2)

2.3 Action Space

The action space is discrete with six possible actions: up, down, left, right, stay, and "interact," which is a contextual action determined by the tile the player is facing (e.g., placing an onion on the counter when facing a counter or initiating cooking when facing a pot). Each layout has one or more onion dispensers and dish dispensers, which provide an unlimited supply of onions and dishes, respectively.

2.4 Installation Notes and Student Notebook

The environment is officially supported on Python 3.8 and is installed via `pip`. To help you with getting started, we are providing you with a Jupyter notebook and a requirements.txt file. Note that this requirements.txt file does not include torch, which you should install separately following the instructions from the PyTorch website. Note also that you can get Python 3.7 to work but you may have to play with the dependencies. You may create a copy of this notebook in order to run the starting code. This notebook demonstrates installing, building, interacting with, and visualizing the environment. Note that you must build the environment exactly how it's built in the notebook (i.e., the section in the notebook entitled "Environment Setup"). Everything else, including the visualization routines, are optional. You are not required to use this notebook in your project, but we encourage you to use it as a companion to this document to better understand the environment. We require the use of PyTorch if using deep learning methods. You absolutely do not need a GPU to solve any of the layouts in less than 10 hours (in fact, GPUs typically slow RL algorithms down unless you are processing very large batches).

For more details on installation and operation, refer to the GitHub repository - `https://github.com/HumanCompatibleAI/overcooked_ai`

2.5 IMPORTANT: Reward Shaping Addendum

If you plan on using reward shaping, take a look at how the default shaped rewards are swapped by the agent index in the provided notebook (in the "Train your agent" section). Upon episode reset, agents are assigned randomly to one of the 2 starting positions. This assignment is only reflected in the official observation that is returned to you by the environment's `step` method. **Any state variable you obtain from the Overcooked environment that is not in this observation variable (including anything in the `info` dictionary or the base environment)** needs to be similarly swapped. Failure to do this means you will be assigning credit to the wrong agent roughly half the time, crippling your algorithm.

2.6 Strategy Recommendations

You are free to pursue any multi-agent RL strategies in your soup-cooking quest. For instance, you may pursue a novel reward-shaping technique, however, make sure that the method chosen is relevant to multi-agent RL problems. We **strongly recommend** that you start with the `cramped_room` layout. It is the easiest of the three layouts and will give you a chance to iron out any bugs and familiarize yourself with the environment. Below are further examples of strategies worth pursuing:

- using reward shaping techniques for improving multi-agent considerations such as collaboration and credit assignment;
- asynchronous methods Mnih et al. 2016;
- centralizing training and decentralizing execution (Lowe et al. 2017; J. N. Foerster et al. 2017);
- value factorisation Rashid, Samvelyan, De Witt, et al. 2020;
- employing curriculum learning (some single-agent ideas in this dissertation may be interesting and easy to extend to the multi-agent case *e.g.*, Narvekar 2017).
- adding communication protocols (J. Foerster et al. 2016);
- improving multi-agent credit assignment (J. N. Foerster et al. 2017; Zhou et al. 2020);
- improving multi-agent exploration (Iqbal and Sha 2019; Wang et al. 2019)
- finding better inductive biases (*i.e.*, choosing the function space for policy/value function approximation) to handle the exponential complexity of multi-agent learning, *e.g.*, graph neural networks (Battaglia et al. 2018; Naderializadeh et al. 2020).

3 Procedure

This problem is more sophisticated than anything you have seen so far in this course. **Make sure you reserve enough time to consider what an appropriate approach might involve and, of course, enough time to build and train it.**

- Clearly define the direction of your project and which aspect(s) you aim to focus on as you try to solve all of the layouts. For example, do you want to improve collaboration among your agents?
  - This includes why you think your algorithm/procedure will accomplish this and whether or not your results demonstrate success.
- Implement a solution that produces such improvements.
  - Use any algorithms/strategy as inspiration for your solution.
  - **The focus of this project is to try new algorithms/solutions, rather than to simply improve hyper-parameters of the algorithms already implemented. Further, avoid searching for random seeds that happen to work the best as this is inconsequential analysis. Remember that the algorithm/reward-shaping/hyperparameters must be fixed across all 3 layouts.**
  - Justify the choice of that solution and explain why you expect it to produce these improvements.
  - **Even if your solution does not solve all of the layouts, you still have the ability to write a solid paper.** We have specifically made the penalty for not solving layouts relatively small to encourage student exploration.
  - Upload/maintain your code in your private repo at `https://github.gatech.edu/gt-omscs-rldm`.
- Describe your experiments and create graphs that demonstrate the success/failure of your solution.
  - You must provide one graph demonstrating the number of soups made across all three layouts **during training**. You can combine all three layouts' plots onto one graph if you wish. Displaying a simple moving average for each layout's training run is suggested to help with clarity.

- You must provide one graph demonstrating performance of your trained agent on each layout **over at least 100 consecutive episodes**. Again, you can combine all three layouts' plots into one graph. If all three of these graphs are flat lines (a possible consequence of using a deterministic algorithm on a deterministic environment), then a bar graph is ok.
- Additionally, you must provide at least two graphs using metrics you decided on that are significant for your hypothesis/goal.
- Analyze your results and explain the reasons for the success/failure of your solution.
- Since graphs are largely decided by you, they should have **clear** axis, labels, and captions. You will lose points for graphs that do not have any description or label of the information being displayed.
- Example metrics you might consider are number of dish pickups, dropped dishes, incorrect deliveries, or picked up onions. These example metrics and more are built-in to the environment and are accessible via the `info` variable at the end of an episode. In your report you should clearly motivate why you are interested in a particular metric. See the provided notebook.

4 Deliverables

4.1 Code

- We have created a private Georgia Tech GitHub repository for your code. Upload/maintain your code in your private repo at https://github.gatech.edu/gt-omscs-rldm:
  - The quality of the code is not graded. You don't have to spend countless hours adding comments, etc. However, it will be examined by the TAs.
  - Make sure to include a thorough `README.md` file with your submission to instruct the TAs on how to run your source code.
  - If you work in a notebook, like Jupyter, include an export of your code in a `.py` file along with your notebook
  - Using an RL, Deep RL or MARL library instead of providing your own work will earn you a 0 grade on the project, and you will be reported for violating the Honor Code.
  - You will be penalized by 25 points if you do not have any code in your Github repo or if you do not merge your code into the main branch.
  - You will be penalized by 15 points if you do not include the git hash for your last commit in your paper.

4.2 Report

- Write a paper describing your agents and the experiments you ran.
  - Include the hash for your last commit to the GitHub repository in the header on the first page of your paper.
  - Make sure your graphs are legible and you cite sources properly. As with previous projects, you should use the RLC conference paper format: Overleaf Teamplate.
  - 8 pages maximum—anything you write after page 8 will not be read and will not be a part of your grade.
  - Explain your algorithm(s).
  - Explain your training implementation and experiments.
  - An ablation study would be a interesting way to find out the different components of the algorithm that contribute to your metric. (See J. N. Foerster et al. 2017.)
  - Graphs highlighting your implementations successes and/or failures.
  - Explanation of algorithms used: what worked best? what didn't work? what could have worked better?
  - Justify your choices.

    - Unlike Project 1, there are multiple ways of solving this problem and you have a lot of discretion over the general approach you take as well as experimental design decisions. Explain to the reader why, from amongst the multiple alternatives, you chose the ones you did.
    - **Your focus should be on justifying the algorithm/techniques you implemented.**
  - Explanation of pitfalls and problems you encountered.
  - What would you try if you had more time?
  - Save this paper in PDF format.
  - Submit to Gradescope!

4.3 Video

- Record a short video (maximum 5 minutes) explaining your approach to solving the Overcooked layouts, presenting your results, and highlighting the key lessons learned. Your video should follow a clear narrative:
  - Introduce the problem and describe the RL algorithm you selected, and why you selected it.
  - Explain your training process, design choices, and any challenges you faced.
  - Present your findings using visual aids (e.g., slides, plots, or a demo recording of your agent in action). You may reuse figures from your report, but do not scroll through the report itself.
  - Conclude with the most important lessons learned, including what you might do differently in the future.

  To protect your work, add a small watermark (e.g., your GT username) to any images or videos in your presentation. Do not include source code in the video.
- You may use Kaltura (provided by Georgia Tech), PowerPoint's video recording tool, or any other screen recording software to create your video. You are not required to appear on camera.
- Submit a link to your video hosted in a Georgia Tech enterprise service (Microsoft OneDrive, Box, Dropbox, or Media Space) along with the git hash of your final code in your report. The link must be set so that anyone within Georgia Tech can **view**, so that the instructional team can access and evaluate your submission.
- You will be penalized by 15 points if the video link is missing or the sharing settings do not permit the instructional team to view it.

4.4 Resources

4.4.1 Lectures

- Lesson 11A: Game Theory
- Lesson 11B: Game Theory Reloaded
- Lesson 11C: Game Theory Revolutions

4.4.2 Readings

- J. N. Foerster et al. 2017
- Lowe et al. 2017
- Rashid, Samvelyan, Witt, et al. 2018

4.4.3 Talks

- Factored Value Functions for Cooperative Multi-Agent Reinforcement Learning
- Counterfactual Multi-Agent Policy Gradients
- Learning to Communicate with Deep Multi-Agent Reinforcement Learning
- Automatic Curricula in Deep Multi-Agent Reinforcement Learning

4.5 Submission

The due date is indicated on the Canvas page for this assignment. Make sure you have set your timezone in Canvas to ensure the deadline is accurate.
Due Date: **Indicated as "Due" on Canvas**
Late Due Date [**20 point penalty per day**]: **Indicated as "Until" on Canvas**
The submission consists of:

- Your written report in PDF format (Make sure to include the git hash of your last commit)
- Your source code in your personal repository on Georgia Tech's private GitHub

To complete the assignment, submit your written report to Project 3 under your Assignments on Gradescope: https://www.gradescope.com/

You may submit the assignment as many times as you wish up to the due date, but, we will only consider your last submission for grading purposes. Late submissions will receive a cumulative 20 point penalty per day. That is, any projects submitted after midnight AOE on the due date get a 20 point penalty. Any projects submitted after midnight AOE the following day get a 40 point penalty and so on. No project will receive a score less than a zero no matter what the penalty. Any projects more than 4 days late and any unsubmitted projects will receive a 0.

Note: Late is late. It does not matter if you are 1 second, 1 minute, or 1 hour late. If Gradescope marks your assignment as late, you will be penalized. **Additionally, if you resubmit your project and your last submission is late, you will incur the penalty corresponding to the time of your last submission.**

4.6 Words of Encouragement

We understand this is a daunting project with many possible design directions to consider. As Graduate Students in Computer Science, projects that allow you to challenge and expand your skills in a practical and low-stakes manner are crucial. These projects are ideal for testing the knowledge you have garnered throughout the course and applying yourself to a difficult problem commonly faced when applying reinforcement learning in industry. After completing the course, a project like this can be valuable to highlight during interviews, to demonstrated your newfound knowledge to current employers, or to add a (new) section on your resume. Historically, many students have reported back the positive interactions encountered when discussing their projects, sometimes leading to job offers or promotions. However, please remember not to publicly post your report or code. The project is a good **talking** point and you would be within the bounds of the GT Honor Code if you were to share it privately with a potential employer (if you so desire), however making any part of this project publicly available would be a violation of the GT Honor Code.

We encourage you to **start early** and dive head-first into the project to try as many options as possible. We strongly believe the more successes and failures you experience, the greater your growth and learning will be.

The teaching staff is dedicated to helping as much as possible. We are excited to see how you will approach the problem and have many resources available to help. Over the next several Office Hours, we will be discussing various approaches in detail, as well as dive deeper into approaches on Ed Discussions. We are here to help you and want to see you succeed! With all that said:

Good luck and happy coding!

References

[Bat+18] Peter W Battaglia et al. "Relational inductive biases, deep learning, and graph networks". In: *arXiv preprint arXiv:1806.01261* (2018).

[Car+19] Micah Carroll et al. "On the utility of learning about humans for human-ai coordination". In: *Advances in neural information processing systems* 32 (2019).

[Foe+16] Jakob Foerster et al. "Learning to Communicate with Deep Multi-Agent Reinforcement Learning". In: *Advances in Neural Information Processing Systems* 29 (2016), pp. 2137–2145.

[Foe+17] Jakob N. Foerster et al. "Counterfactual Multi-Agent Policy Gradients". In: *CoRR* abs/1705.08926 (2017). arXiv: 1705.08926. URL: http://arxiv.org/abs/1705.08926.

[IS19] Shariq Iqbal and Fei Sha. "Coordinated Exploration via Intrinsic Rewards for Multi-Agent Reinforcement Learning". In: *arXiv preprint arXiv:1905.12127* (2019).

[Low+17] Ryan Lowe et al. "Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments". In: *CoRR* abs/1706.02275 (2017). arXiv: 1706.02275. URL: http://arxiv.org/abs/1706.02275.

[Mni+16] Volodymyr Mnih et al. "Asynchronous methods for deep reinforcement learning". In: *International conference on machine learning*. PMLR. 2016, pp. 1928–1937.

[Nad+20] Navid Naderializadeh et al. "Graph Convolutional Value Decomposition in Multi-Agent Reinforcement Learning". In: *arXiv preprint arXiv:2010.04740* (2020).

[Nar17] Sanmit Narvekar. "Curriculum Learning in Reinforcement Learning." In: *IJCAI*. 2017, pp. 5195–5196.

[Ras+18] Tabish Rashid, Mikayel Samvelyan, Christian Schröder de Witt, et al. "QMIX: Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning". In: *CoRR* abs/1803.11485 (2018). arXiv: 1803.11485. URL: http://arxiv.org/abs/1803.11485.

[Ras+20] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder De Witt, et al. "Monotonic value function factorisation for deep multi-agent reinforcement learning". In: *The Journal of Machine Learning Research* 21.1 (2020), pp. 7234–7284.

[Wan+19] Tonghan Wang et al. "Influence-Based Multi-Agent Exploration". In: *International Conference on Learning Representations*. 2019.

[Zho+20] Meng Zhou et al. "Learning implicit credit assignment for multi-agent actor-critic". In: *arXiv e-prints* (2020), arXiv–2007.