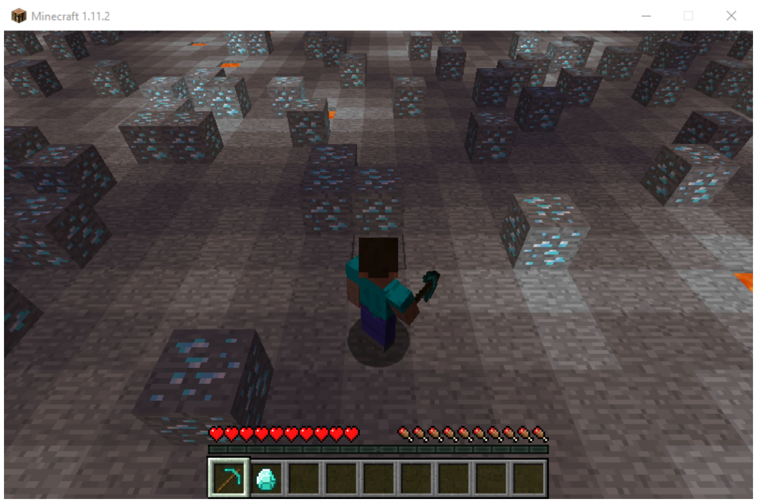
CS 175: Project in AI (in Minecraft): Winter 2021

Assignment 2: Diamond Collector

Sameer Singh (with Kolby Nottingham)
`https://canvas.eee.uci.edu/courses/34142`

In this assignment we will utilize the Proximal Policy Optimization algorithm to learn how to find and collect diamonds. Rather than implementing the algorithm outselves we will use a preimplemented version from the RLlib library. Right now, the agent spawns in an empty field. You will be tasked with spawning diamonds for the agent to collect, adding obstacles for the agent, and modifying the action space from discrete to continuous.

Note: Running `assignment2.py` to completion takes from 40 minutes to an hour, so start this assignment several days early. If you have access to GPU acceleration, feel free to modify the code to use CUDA, but you will not get amazing speedups. The network is so small that the real bottleneck is Malmo.



1 Task Description

Steve has a diamond pickaxe and wants to continue to craft his inventory of diamond armor and tools. The thing is, Steve can only see two squares in any direction. He needs to learn how to navigate to diamond ore while avoiding any hazards he may face along the way.

We will be modeling this scenario using Malmo. We provide you with the reinforcement learning agent, Steve, in `assignment2.py`. We provide an environment class that interfaces with RLlib via the gym API. The gym API consists of the `action_space` and `observation_space` members and the `reset` and `step` methods. The purpose of this assignment is to familiarize yourself with states, actions, rewards, and the Malmo documentation.

Your changes to the Malmo mission should end up looking similar to the screenshot above.

1.1 Overview of the Code

The python file `assignment2.py` contains a number of functions for interfacing with RLlib and building a Malmo mission. Hyperparameters for Malmo and the environment setup can be found in the `__init__` method. Hyperparameters for RLlib can be found at the bottom of the file. The code also regulary logs your agent's returns in a text file and a graphs them in a png file called `returns.txt` and `returns.png` respectively. You will not be required to modify most of these functions. Any functions you do modify should be included in your report with an explanation of your modification.

1.2 Setup and Running the Code

Before running the code for this asssignment, you will need to pip install `numpy`, `matplotlib`, `ray[rllib]` and `torch`. The first three should easily install with `pip install numpy matplotlib ray[rllib]`. Depending on your setup, pytorch will require a different installation process. Follow the installation instructions at https://pytorch.org/get-started/locally/. Once that's done, all you need to do to run this assignment is to copy the `assignment2.py` file above to the `Python_Examples` folder, and after launching Minecraft, run `python assignment2.py`. If everything runs successfully, the agent should be moving around. Please do this as soon as possible, and post on Campuswire if unable to do so.

1.3 Spawning The Diamonds

```
<DrawCuboid x1='{}' x2='{}' y1='2' y2='2' z1='{}' z2='{}'
type='air'/>".format(-SIZE, SIZE, -SIZE, SIZE)
<DrawCuboid x1='{}' x2='{}' y1='1' y2='1' z1='{}' z2='{}'
type='stone'/>".format(-SIZE, SIZE, -SIZE, SIZE)

YOUR CODE GOES HERE

<DrawBlock x='0' y='2' z='0' type='air' />
<DrawBlock x='0' y='1' z='0' type='stone' />
```

Your first task is to spawn diamonds for Steve to mine. The `get_mission_xml` function returns an XML string that specifies the mission. Inside `Mission->ServerSection->ServerHandlers->DrawingDecorator` there are two `DrawCuboid` lines that reset the map and two `DrawCube` lines that ensure the agent is standing in an available spot on top of a stone block. These lines are shown in the code block above. After the map is reset and before the agent's starting position is prepared, you will add tags that spawn diamond ore in the y=2 plane. These blocks should be placed randomly so that each time `get_mission_xml` is called the blocks will be in different places. We provide a hyperparameter called `reward_density` that represents the suggested probability of placing diamond ore at any given location.

Before Steve will learn to mine diamonds, we need to give him a reward signal for him to optimize. Add a reward tag to the AgentHandlers in the mission xml that will give Steve +1 reward each time he picks up a diamond. When you run the program, the logging files should show a non-zero return. Note that logging is set to begin after the 10th episode. Modify `log_frequency` to get results back sooner, but please use `log_frequency=10` for your submission.

Once this is all working, run the python file. The agent should begin to learn to navigate to the diamond ore and collect it. You can watch the `returns.png` file as the agent trains. Run `assignment2.py` for at least 10,000 steps and save the generated plot and text files. Be sure to change the name of the files or save them in another directory so that you don't accidentally overwrite them.

Refer to https://microsoft.github.io/malmo/0.30.0/Schemas/Mission.html for documentation on editting the Malmo mission xml.

1.4 The Floor Is Lava!

Now lets make things more interesting for Steve. Your next task is to add lava in the floor. Randomly place lava in the y=1 plane. We suggest placing the lava block with the probability specified by `penalty_density`. Your Malmo environment should now look similar to the screenshot above.

Finally, we will be penalizing Steve for stepping on your lava blocks by giving him negative reward. Add a reward tag to the AgentHandlers in the mission xml to penalize the agent for falling into lava. A reward of -1 should work. Run `assignment2.py` to at least 10,000 steps and save the generated plot and text files.

1.5 Continuous Actions

Right now Steve chooses between four discrete actions at each step. Those are move forward, turn left, turn right, and break block. Rather than doing this we could give Steve continuous control over the environment. That is we can allow Steve to move between blocks smoothly and choose *how much* he moves forward, backward, left, and right. To switch to using continuous malmo, follow the below instructions.

1. Switch to continuous malmo commands

   Modify the mission xml to replace `<DiscreteMovementCommands/>` with `<ContinuousMovementCommands/>`.

2. Indicate to RLlib that you want continuous actions

   Modify the `action_space` member defined in `__init__` to define a continuous action space using the `gym.spaces.Box` class. Steve's continuous action space will be composed of seperate dimensions for the move, turn, and attack continuous commands. The valus for each of these three dimensions should range from -1 to 1 continuously. You can find details about `gym.spaces.Box` here: https://github.com/openai/gym/blob/master/gym/spaces/box.py.

3. Send continuous commands from your step function for each of the action dimensions

   Now that you have changed `action_space`, the type of input to the step function will change. When using a discrete action space, the action argument was an index. Now the argument to the step function will be an array with shape and values that coorespond to your newly defined action space. Rather than using `action` as an index to `action_dict`, use the three dimensions of the array to send move, turn, and attack commands. The move and turn commands take continuous values from -1 to 1 while the attack command must be 0 or 1. Use a threshold so that negative values in the attack dimension of the array map to "attack 0" and positive values map to "attack 1".

4. Modify the XML to maintain the same quit conditions for malmo

   Note that you should now be sending three commands per step rather than one. Modify the maximum number of malmo commands per episode so that the maximum number of steps remains the same. Do this by changing the `AgentQuitFromReachingCommandQuota` line in the XML.

5. Make breaking blocks easier

   Your agent should be successfully moving around your environment no longer fixed to the grid. However, you will notice that the agent rarely performs the attack action long enough to actually break a block. This is because after recieving "attack 1", the agent will continue to attack until recieving "attack 0". Modify your step function so that the agent successfully breaks blocks every episode. Hint: try sleeping longer after sending "attack 1" and freezing the agent in place so that it has time to break the blocks.

Run you completed continuous agent to at least 10,000 steps and save the result files.

1.6 Extra Credit: Observation Space

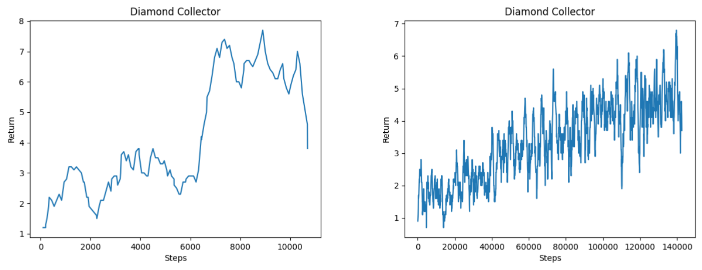
An environment's state summarizes all of the information the agent would need to know to for it to perform optimially. In our case the state is the location and orientation of the agent along with the location of all of the diamonds and lava on the map. However our agent only recieves a partial observation of this state. For example, the discrete agent can only see diamonds and lava up to 2 squares away.

The continuous agent suffers even more from partial obervability. Rather than being centered on each block facing at 90 degree angles, the continuous agent can stand in a countinuous range of values within a block and face a continuous range of values in a circle. Meanwhile the observation continues to assume the agent is centered and at right angles. That means that the agent's knowledge of it's own location has an error of $\pm\sqrt{2}$ blocks and an orientation error of ±45 degrees.

Modify the observation to reduce this error and help the agent perform better. It will probably require new malmo observations and modifications to the RLlib obseravtion space. Test your solution by running experiments with and without your modifies observation space for at least 50,000 each.

2 What Do I Submit?

For the diamonds and diamonds&lava return graphs, the graph should be obviously improving as the number of steps increases, but you do not need to achieve a specific return value. You should see improvement within 10,000 steps. You may not see improvement within 10,000 steps for the continuous agent, but the graph should show non-zero reward. Also if you are running out of time you can submit incomplete return graphs, but you will be docked points. A complete return graph should go up to at least 10,000 steps. Example graphs for diamonds&lava (left) and a continuous agent (right) are provided below.



Note: On Canvas, after pasting your code, highlight all the lines, click on the drop down that says "Paragraph", then select "Preformatted". The font of those lines should change. When you submit those lines will be formatted as code.

1. **Code: get_mission_xml function (30 points):** The `get_mission_xml` function should be modified in 4 ways. First, diamond ore should be spawned and a positive reward added for collecting it (10 points). Second, lava should be spawned with a negative reward for falling into it (10 points). Third, malmo should be using continuous commands (5 points). Fourth, the command quota should be increases so that episode lengths stay approximately the same (5 points).
2. **Code: action_space member (5 points):** Modify the `action_space` member and show us what you changed. It should be of type `gym.spaces.Box`
3. **Code: step function (10 points):** Modify the `step` function and show us what you changed. This should show your new sendcommand lines for the continuous action space.
4. **Graph: Diamonds Only Graph (15 points):** Provide the graph/png output of your original experiment mining for diamonds without lava. Show at least 10,000 steps. There should be *clear improvement*.
5. **Graph: Diamonds and Lava Graph (15 points):** Provide the graph/png output of from mining for diamonds with lava. Show at least 10,000 steps. There should be *clear improvement*.
6. **Graph: Continuous Actions Graph (15 points):** Provide the graph/png output of from mining for diamonds with lava with a continuous action space. Show at least 10,000 steps. There should be *non-zero rewards*.
7. **Statement of Collaboration (10 points):** It is **mandatory** to include a *Statement of Collaboration* with respect to the guidelines below. Include the names of everyone involved in the discussions (especially in-person ones), and what was discussed. You should also include the links to all online resources you used for the assignment in this section.
8. **Comments:** Any comments about your submission that you want to bring to our attention as we are grading it. This is completely optional, I expect most of you to leave this empty.
9. **Extra Credit: Continuous Action Observation Space (up to 20 points):** Implement your new continuous action observation space. Include any code changes you made. Run your agent with and without these changes for at least 50,000 steps. You may want to run the agent overnight. Explain why you think you got the results you got.

All students are required to follow the academic honesty guidelines posted on the course website. For programming assignments, in particular, we encourage the students to organize (perhaps using Campuswire) to discuss the task description, assignment requirements, bugs in our Malmo code, and the relevant technical content *before* they start working on it. However, you should *not* discuss the specific solutions, and, as a guiding principle, you are not allowed to take anything written or drawn away from these discussions (i.e. no screenshots/photographs, written notes, etc.). The same holds for online resources: you are allowed to read the description of algorithms, but your code should be your own. Especially *after* you have started working on the assignment, try to restrict the discussion to Campuswire as much as possible, so that there is no doubt as to the extent of your collaboration.

Acknowledgements

This assignment was conceived and created by Kolby Nottingham.