Design Analysis of GridEnv environment

Design Choices

This document covers the design and intuition of the GridEnv environment developed using the Gymnasium library of OpenAI.
To start off, the problem is defined as follows: A robot is placed randomly on a 2D MxN grid, along with K obstacles and a target location. The goal is to train an agent capable of reaching the target without hitting the obstacles or stepping out of bounds. For the solution to this problem we followed a bottom-up approach, solving first easier problems and increasing the difficulty of the problem gradually.
For the environment, the action space is chosen to be discrete with 4 possible actions:

- Up
- Down
- Left
- Right

These four actions are what make the robot move inside the grid.
For the observation space and the reward system of the environment the following three cases are discussed:

- Empty grid with the agent forced to stay inside the grid by clipping its position on the board
- Grid with obstacles with the agent forced to stay inside the grid by clipping its position on the board
- Grid with obstacles and a penalty to the agent for stepping out of bounds

These 3 cases helped with the understanding of the problem and the proposed solution. Starting from the first case, the easiest, we give the agent a positive reward only when reaching the target. The reward given is +1. This fairly easy scenario is solved by both the PPO and DQN agents that were trained as it will be discussed later. The observation space contains the positions of the agent and the target.

Now moving to the second case, the idea is to penalize the agent whenever it hits an obstacle. Also we terminate the episode if an obstacle is hit. The reward for hitting an obstacle is -1 and the reward for hitting the target is still +1. But now a problem arises, because the positive reward of actually hitting the target is very sparse and the agent has little to no time to find the goal at the early stages of the training. So the training takes much time and is unstable. To counteract this problem we need to reward the agent more frequently and create a denser reward. For this denser reward we are going to use the positions of the agent and the goal and the distance between them. We reward the agent whenever it makes a step that leads it closer to the target and penalize it whenever it makes a step that takes it further away from the goal. This reward needs to be much much smaller than the one when reaching the goal because the agent must learn that the position of the target is the ultimate position that it must move. But this idea is what makes the agent find the goal much sooner and leads to a more stable training. Also this is what allows the agent to solve the environment with many obstacles. One thing to note is that the reward for moving closer to the target must be smaller (in absolute value sense) than the one of moving away from the target. This is because the agent will learn to take a few steps towards the goal and then keep going back and forth staying in one position for however long the episode is, as it was observed by failed training attempts. The reward for moving closer to the target is +0.01 and the reward for moving away is -0.05. All in all the rewards for this case are:

- +1 for reaching the target
- -1 for hitting an obstacle (the episode is terminated)
- +0.01 for moving closer to the goal
- -0.05 for moving away from the goal

This environment is solvable fairly easily by all the agents and at the end they do not even step out of bounds. The observation space now contains the positions of the target, the agent and the obstacles.

To conclude the design of the environment now we need to make the “stepping out of bounds” case learnable i.e. let the agent learn that it must not step out of the grid. The only change from the previous environment is that we add a negative reward for stepping out of the grid and terminate the episode. Since stepping out of the grid must lead to termination of the episode, the reward is set to -1, as harsh as hitting an obstacle. So the final environment has the following reward system

- +1 for reaching the target
- -1 for hitting an obstacle (the episode is terminated)
- +0.01 for moving closer to the goal
- -0.05 for moving away from the goal
- -1 for stepping out of bounds

The observation space remains the same. For the bonus question of the expanded environment, we need to add additional logic to the environment so the agent learns to visit the bonus positions. The idea is to encourage the agent to visit the bonus positions but discourage the agent from reaching the target before collecting all the bonuses. To implement this we first add the positions of the bonuses to observation space along with the number of bonuses collected. The logic of the normal grid environment remains the same but now we add some extra rewards. First of all, we give the agent a reward of +0.5 for reaching a bonus position. If the agent reaches the target after collecting all the bonuses then it is rewarded with a +1 reward. If the target is reached but not all the bonuses are collected then the episode is terminated and the agent receives a -1 reward. So now the reward function looks like this:

- +1 for reaching the target
- -1 for hitting an obstacle (the episode is terminated)
- +0.01 for moving closer to the goal
- -0.05 for moving away from the goal
- -1 for stepping out of bounds
- +0.5 for reaching a bonus position
- +1 if all the bonuses are collected and the target is reached
- -1 if the target is reached but not all the bonuses are collected (the episode is terminated)

The observation space now contains the positions of the target, the agent, the obstacles and the bonuses but also the remaining bonuses to be collected.

RL Algorithms and Performance

For the RL algorithms we pick an on policy algorithm in PPO and an off policy algorithm in DQN. Both of these algorithms are taken from the stable-baselines3 library. For both, an MLP Policy was used but with a few tweaks to make the training more stable.

For the PPO we opted for the following hyperparameter setup:

Parameter	**Value**
learning rate	1e-4
n_steps	1024
batch size	64
n_epochs	10
discount factor (gamma)	0.99

gae_lambda (generalized advantage estimation)	0.95
clip range (epsilon value in Lclip loss function)	0.2
entropy coefficient (helps with exploration)	0.01
maximum gradient norm	0.5

For the network architecture, the weight sharing that is the default of sb3 was disabled and instead two different MLP networks for the actor and critic were used. The networks both had two Linear Layers of 256 neurons each with LeakyReLU as the activation function.
The AdamW optimizer was used with decoupled weight decay set to 0.01.
For the DQN we opted for the following hyperparameter setup:

Parameter	**Value**
learning rate	1e-4
buffer size	100,000
batch size	64
tau (for soft updates)	1e-3
gamma (discount factor)	0.99
train_freq (update model very trai_freq steps)	4
gradient steps	1
target_update_interval	8
exploration_fraction	0.4
exploration_initial_eps	1.0
exploration_final_eps	0.05

For the Qnetwork like the PPO agent an MLP network was used with two Linear Layers of 256 neurons each again with LeakyReLU as the activation function and the optimizer was set to AdamW with decoupled weight decay set to 0.01.

For both agents the maximum steps for an environment was set to 50 and the total training timesteps to 10M. Both of the agents were able to solve the environment after 10M timesteps. Below the results can be seen for 100 episodes.

Agent	**Success Rate**	**Average Steps/Episode**	**Average Reward/Episode**
PPO	91.6%	5.9	0.87
DQN	94.3%	6.7	0.94

As we can see the DQN is able to beat the PPO in the success rate and the average reward per episode. Although there is a stochasticity factor embedded into the environment (due to the randomization of the positions) the DQN is able to maintain a 93-95% success rate across different runs. The PPO is itself a stochastic agent in the sense of picking actions, so the variance of the success rate is higher than that of the DQN. The above results correspond to an average run. So after 10M timesteps the DQN is the superior agent due to the more consistent results. This can be attributed to the underlying differences of the two agents. The DQN tries to maximize the value function Q(s,a) so at each step it takes the best possible action in that sense. The PPO creates a probability density function of the actions and samples actions from that distribution. So low probability actions are still possible to be picked which may result in sub optimal performance and actions that lead to termination of an episode. Further training is required to boost the performance of the agent but the current performance is still acceptable.

For the expanded environment, the number of obstacles was set to 3 for simplicity and the number of bonus positions was set to 2. So in total 5 important positions, 3 obstacles and 2 bonuses. The agents were trained for 10M steps and the setup of the agents remained the same as before. The results can be seen bellow:

Agent	**Success Rate**	**Average Steps/Episode**	**Average Reward/Episode**
PPO	65%	18.2	0.5037
DQN	31%	35.4	0.252

The results suggest that now the PPO agent performs significantly better on this expanded environment although there is room for improvement on both agents. This could be because the

PPO agent, despite its stochasticity, provides a much more sophisticated policy estimation and is able to find the correct policy in fewer steps.

Challenges

The main challenge was creating the environment and picking the observation space and the rewards. After a few failed attempts to solve the problem, the idea of gradually increasing the difficulty of the problem and solving the easiest problems first, is what helped with the brainstorming and the design choices discussed earlier. After the environment was created the training of the agents and the experimentation with the hyperparameters was a basic task.