Exploring Open AI Gym for Tic Tac Toe

By Vikraant Pai

Link to Colab: https://github.com/vikpy/AISem3/blob/master/HW/HomeWork_Tic_Tac_Toe.ipynb

Some environments in OpenAI gym

gym-anytrading: Environments for trading markets
OpenAI Gym environments for reinforcement learning-based trading algorithms

gym-carla:Gym Wrapper for CARLA Driving Simulator
Realistic 3D simulator for autonomous driving research

gym-inventory: Inventory Control Environments
single agent domain featuring discrete state and action spaces that an AI agent might encounter in inventory control problem

osim-rl: Musculoskeletal Models in OpenSim
A human musculoskeletal model and a physics-based simulation environment where you can synthesize physically and physiologically accurate motion

Tic Tac Toe using gym-tictactoe

Tic Tac Toe

Importing the tic tac toe env

Before starting with implementation of the

Game you have to import gym tic tac toe

```
[ ]  !pip install gym-tictactoe
     import gym
     import gym_tictactoe
```

Create an Instance

Once you have imported, you have to create an instance of the tic tac toe game.

```
env = gym.make('tictactoe-v0')
env.reset()
```

```
[[[0, 0, 0], [0, 0, 0], [0, 0, 0]],
 [[0, 0, 0], [0, 0, 0], [0, 0, 0]],
 [[0, 0, 0], [0, 0, 0], [0, 0, 0]]]
```

Tic Tac Toe

Start the game in the defined environment

Here both the players that are playing the game are humans, hence we design the game accordingly

```
import gym
import gym_tictactoe

def play_game(actions, step_fn=input):
  env = gym.make('tictactoe-v0')
  env.reset()

  # Play actions in action profile
  for action in actions:
    observation, reward, done, info = env.step(action)
    print(info)
    env.render()
    if step_fn:
      step_fn()
  return env
```

Define the moves for player 1 and 2 and start the game

Define what moves player 1 and player 2 would like to take and accordingly create the game

```
# [player] | [block] | [column] | row
actions = ['1001', '2111', '1221', '2222', '1121', '2011', '1021']
_ = play_game(actions, None)
```

Tic Tac Toe

Output

```
{'round': 1, 'next_player': 2}
- - -    - - -    - - -
x - -    - - -    - - -
- - -    - - -    - - -
{'round': 2, 'next_player': 1}
- - -    - - -    - - -
x - -    - o -    - - -
- - -    - - -    - - -
{'round': 3, 'next_player': 2}
- - -    - - -    - - -
x - -    - o -    - - x
- - -    - - -    - - -
{'round': 4, 'next_player': 1}
- - -    - - -    - - -
x - -    - o -    - - x
- - -    - - -    - - o
```

```
{'round': 5, 'next_player': 2}
- - -    - - -    - - -
x - -    - o x    - - x
- - -    - - -    - - o
{'round': 6, 'next_player': 1}
- - -    - - -    - - -
x o -    - o x    - - x
- - -    - - -    - - o
{'round': 7, 'next_player': 'NONE'}
- - -    - - -    - - -
x o X    - o X    - - X
- - -    - - -    - - o
```

The Game Ends here

Gym-sokoban

Reference: https://github.com/mpSchrader/gym-sokoban

Sokoban is Japanese for warehouse keeper and a traditional video game. The game is a transportation puzzle, where the player has to push all boxes in the room on the storage locations/ targets. The possibility of making irreversible mistakes makes these puzzles so challenging especially for Reinforcement Learning algorithms, which mostly lack the ability to think ahead.

Sokoban

Background

Sokoban is based on Deep Reinforcement Learning Architecture

It uses Imagination Augmented Agents(I2A)

I2As learn to interpret predictions from a trained environment model to construct implicit plans in arbitrary ways, by using the predictions as additional context in deep policy networks

Benefits

Sokoban is based on Deep Reinforcement Learning Architecture

It uses Imagination Augmented Agents(I2A)

I2As learn to interpret predictions from a trained environment model to construct implicit plans in arbitrary ways, by using the predictions as additional context in deep policy networks

Sokoban

Background

Sokoban is based on Deep Reinforcement Learning Architecture

It uses Imagination Augmented Agents(I2A)

I2As learn to interpret predictions from a trained environment model to construct implicit plans in arbitrary ways, by using the predictions as additional context in deep policy networks

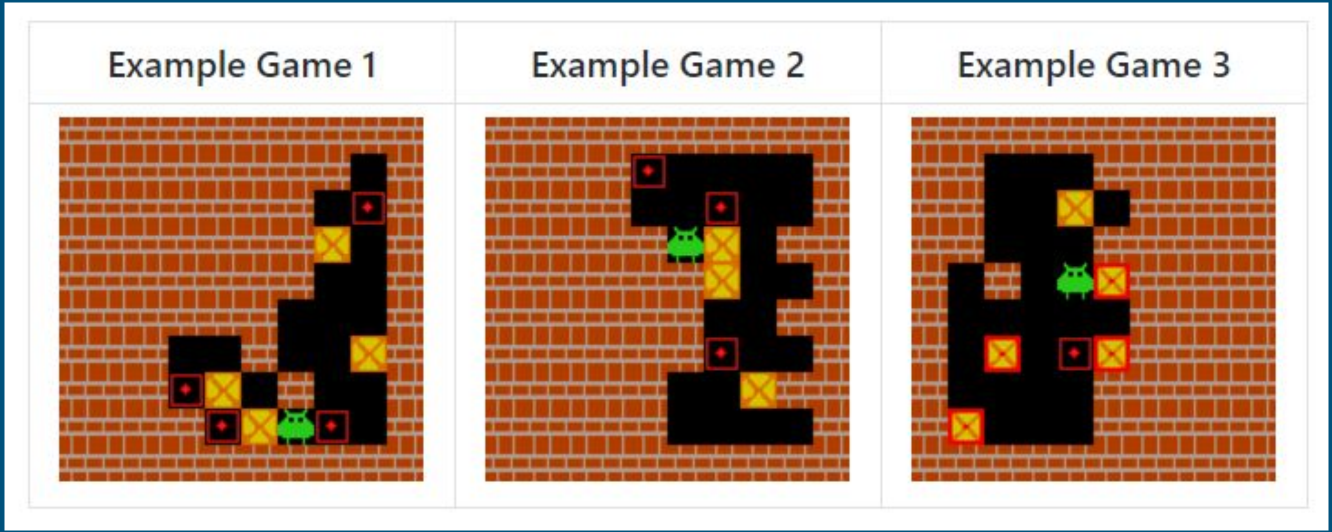
Benefits

I2As show improved data efficiency, performance, and robustness to model misspecification compared to several strong baselines.

Reference:
https://papers.nips.cc/paper/7152-imagination-augmented-agents-for-deep-reinforcement-learning

Sokoban

Example Games



Reference: https://github.com/mpSchrader/gym-sokoban

Sokoban

Room Elements

Type	State	Graphic	TinyWorld
Wall	Static		
Floor	Empty		
Box Target	Empty		
Box	Off Target		
Box	On Target		
Player	Off Target		
Player	On Target		

Reference: https://github.com/mpSchrader/gym-sokoban

Actions

Action	ID
No Operation	0
Push Up	1
Push Down	2
Push Left	3
Push Right	4
Move Up	5
Move Down	6
Move Left	7
Move Right	8

Sokoban

Rewards

Reason	Reward
Perform Step	-0.1
Push Box on Target	1.0
Push Box off Target	-1.0
Push all boxes on targets	10.0

Score Calculation

$$RoomScore = BoxSwaps \times \sum_{i \in Boxes} BoxDisplacement_i$$

Reference: https://github.com/mpSchrader/gym-sokoban

Sokoban

Rendering Modes

Mode	Description
rgb_array	Well looking 2d rgb image
human	Displays the current state on screen
tiny_rgb_array	Each pixel describing one element in the room
tiny_human	Displays the tiny rgb_array on screen

Size Variations

Different box configurations and different grid size are available which can be configured as per the requirements

Eg: 7x7, 3 boxes;etc

Reference: https://github.com/mpSchrader/gym-sokoban