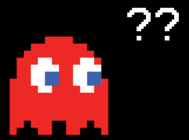
Teaching AI Pac-Man

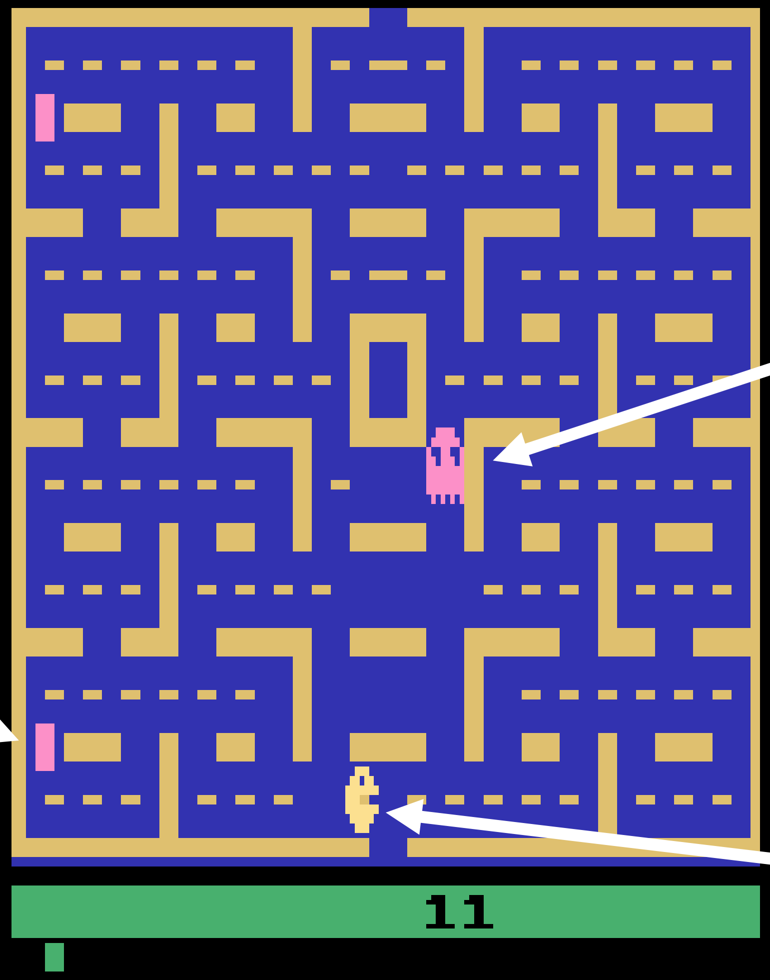
A reinforcement model

Jared Haley
Jimmy Anglim
Mitchel Rigsby
Puja Goyal Pasari



Where to start?

Pac-Man from Atari 2600



Model needs to learn how to avoid ghosts

BUT the model also needs to learn to eat ghosts after collecting the power-up

Model needs to learn how to control Pac-Man in the Maze

Requirements

1. Vision
2. Directional control
3. Rewards and conditional rewards

And many (many) trials…

The Tech

- Deep Q Learning (DQN) as our network of choice
  - Uses a Q-function rather than a Q-table to account for a large state space
- CNN is used for the network to observe the game
  - Takes several frames of the game as input and outputs state values
- The agent learns in a standardized version of the game called a Gym

Parameters

- Training the model took 350,000 steps over 6 hours
- Tested the model on 50 trials and averaged the rewards

Average reward per game: 20.26

- Where one reward is one pellet

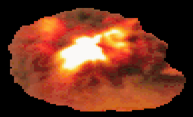
In [8]:

```python
# Import necessary libraries
import gymnasium as gym
from stable_baselines3 import DQN
from stable_baselines3.common.evaluation import evaluate_policy
import numpy as np
from stable_baselines3 import DQN
from stable_baselines3.common.env_util import make_atari_env
from stable_baselines3.common.vec_env import VecFrameStack
import matplotlib.pyplot as plt

GAME_NAME = "ALE/Pacman-v5"

# Create the game environment
env_id = f"{GAME_NAME}"
env = make_atari_env(env_id, n_envs=1, seed=0)
env = VecFrameStack(env, n_stack=4)

# Load the trained model
model = DQN.load(f"{GAME_NAME}_dqn_model.zip", env=env)

# Evaluate the trained agent
mean_reward, std_reward = evaluate_policy(model, env, n_eval_episodes=10)
print(f"Mean reward: {mean_reward} +/- {std_reward}")

# Function to evaluate the trained agent for a single game episode
def evaluate_game(model, env):
    obs = env.reset()
    done = False
    total_reward = 0
    while not done:
        action, _ = model.predict(obs, deterministic=True)
        obs, reward, done, info = env.step(action)
        total_reward += reward
    return total_reward

# Evaluate the model over multiple episodes
num_episodes = 50
total_rewards = []

for _ in range(num_episodes):
    episode_reward = evaluate_game(model, env)
    total_rewards.append(episode_reward)

# Print and plot the evaluation results
print(f"Average Reward per Game: {np.mean(total_rewards)}")

import numpy as np

episode_list = np.array([i + 1 for i in range(num_episodes)], dtype='int32')
score = np.array([reward.item() for reward in total_rewards], dtype='int32')

summary = np.column_stack((episode_list,score))

print('evealuation done')
```

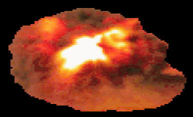
How can this be used for real-world problems?

PacXMan

TheXech



NUCLEAR FUSION[1]



1. Nuclear Fusion is actually very safe

IAEA nuclear fusion

PAPER

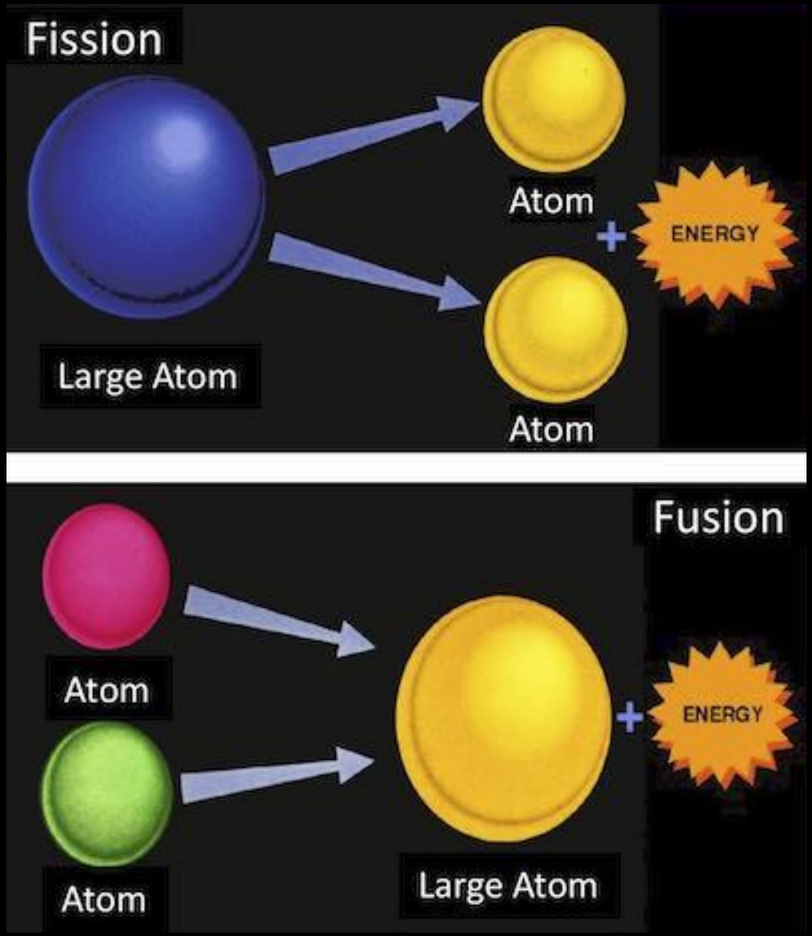
Feedforward beta control in the KSTAR tokamak by deep reinforcement learning

To cite this article: Jaemin Seo *et al* 2021 *Nucl. Fusion* **61** 106010

Research article on using deep reinforcement learning to help control fusion reactions

Nuclear Fusion

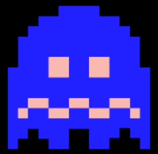
- Nuclear fusion (compared to fission) offers a potential solution to the energy crisis
- Fusion reactions power the Sun and produce little byproducts
- So far, the tech has lagged the understanding



Key Concepts

- New way to control normalized beta in tokamak plasmas using deep reinforcement
- LSTM-based simulation system used as a virtual tokamak environment
  - Model trained off five years of data
  - Objective: control the tokamak to achieve the desired state of plasma

Same underlying framework as Pac-Man!



Ethical Concerns

- Deep RL models are computationally intensive
  - Also require substantial monetary investment
- Reinforcement learning is safe for nuclear fusion, but that is not the case for other fields
  - Tesla’s self-driving cars