Software Requirements Specification

for

Snake AI Gym

Prepared by:

Nilusche Liyanaarachchi

FH Aachen University of Applied Sciences, Germany
Software Engineering

Contents

Chapter 1

Introduction

1.1 Disclaimer

This document is prepared or accomplished by Nilusche Liyanaarachchi (Co-software engineer of Tik Tasks) in his own personal capacity.

The implementation of this Project is based on Deepmind's Playing Atari with Deep Reinforcement Learning Paper.

1.2 Purpose

The purpose of this document is to present a detailed description of the requiurements and features of the Gym-Environment and is intended for the developers and users of the Gym.

1.3 Product Scope

SnakeGym is supposed to simulate the game of Snake in an OpenAI-like environment and should be able to be used as an environment to train several reinforcement learning agents.

1.4 Definitions, Acronyms, and Abbreviations

Term/ Acronym / Abbreviation	Expansion / Description
GUI	Graphical User Interface
DQN	Deep Q Learning
RL	Reinforcement Learning
CNN	Convolutional Neural Network

1.5 References

1. IEEE Software Engineering Standards Committee, “IEEE Std 830-1998, IEEE Recommended Practice for Software Requirements Specifications”, October 20, 1998.

2. Deepmind, "Playing Atari with Deep Reinforcement Learning", December 19, 2013

Chapter 2

Overall Description

2.1 Project Perspective

The project will consist of two parts: one script for the definition of a DQN-Agent, one package that defines a Snake environment class that will inherit from openai's base "gym.Env"-class.
Necessary functions that need to be overwritten conclude:

- reset() - Reset the environment to the initialized state
- step(action) - Uses specified action to take a step and return observation, reward, ending status of game, info
- render(mode) - Takes a specified render mode to render the environment
- close() - Stops render mode

2.2 Operating Environment

- Operating System: Windows
- Python installation
- Pip-dependencies: gym, numpy, tensorflow, keras-rl2, pillow:

2.3 Environment Action Space

The action spaces defines movement-directions of the snake agent

- 0 = UP
- 1 = DOWN
- 2 = LEFT
- 3 = RIGHT

Chapter 3

Requirements

3.1 Functional Requirements

1. Users should be able to render the environment humanly with a GUI
2. Users should be able to manually different types of environment with different metadata
3. A DQN Agent should be able to learn through image observations

3.2 External Interface Requirements

- Video driver that can run pygame

Chapter 4

SnakeGym As Is

4.1 Setup

- Run **pip install -e snake** to register the environment.
- Use agent.py as reference to register a different type of environment.
- Run **python DQN.py** to restart training the DQN Agent.
- Run **python agent.py** to showcase the training

4.2 Reward Function

- 1 for every bit of food eaten
- -1 for every step taken
- 0 if the limit of steps have been reached

4.3 DQN Agent

DQN.py contains a Deep Q-Learning Solution to an agent that learns through a CNN
Keras-rl2 is required to start training. Check this Repository for more info.
DQN.py saves the trained model at certain checkpoints (every 100000 Steps).

Hyperparemeters:

- Epsilon Greedy Policy for action selection for the exploration vs exploitation tradeoff
- CNN for Image replay
- Total number of Epochs: 1.5 Million
- Target model update interval: every 10.000 epochs
- Image Shape: 84 x 84
- Replay Buffer length: 4
- Checkpoint interval: every 100.000 epochs

- Learning Rate: 0.99

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

Figure 4.1: Deepmind's DQN Agent algorithm

Layer (type)	Output Shape	Param #
permute_3 (Permute)	(None, 84, 84, 4)	0
conv2d_9 (Conv2D)	(None, 20, 20, 32)	8224
conv2d_10 (Conv2D)	(None, 9, 9, 64)	32832
conv2d_11 (Conv2D)	(None, 7, 7, 64)	36928
flatten_3 (Flatten)	(None, 3136)	0
dense_5 (Dense)	(None, 512)	1606144
dense_6 (Dense)	(None, 4)	2052

Total params: 1,686,180
Trainable params: 1,686,180
Non-trainable params: 0

Figure 4.2: CNN Topology

4.4 Scores

Scores by number of eaten food using DQN

- Max: 20
- Average:15
- Min: 0

4.5 Conclusion

This project helped me (Nilusche Liyanaarachchi) grasp the concept of DQN by using state of the art algorithms of RL.
Additionally reading Deepmind's paper about Q-Learning only using raw pixels as input helped me understand how to turn papers into code.
Possibilities of Improvement:

- Other RL-Algorithms (i.e asynchronous policy gradient methods like AC3 to decrease training time)
- Different reward function