Comprehensive Guide: Training a Reinforcement Learning Bot to Play Hollow Knight on Mac using Ryujinx and OpenAI Gym

Disclaimer: This guide is for educational purposes only. Ensure you own a legal copy of Hollow Knight and have the right to use it in this manner.

Table of Contents

1. Prerequisites
2. Setting Up the Environment
3. Installing and Configuring Ryujinx
4. Preparing Hollow Knight
5. Creating the Reinforcement Learning Bot with OpenAI Gym
6. Integrating the Bot with Ryujinx
7. Training the Bot
8. Optimizing Performance

9. Evaluating Results

1. Prerequisites

- A Mac computer with macOS Catalina (10.15) or later
- Xcode and Command Line Tools installed
- Homebrew package manager
- Python 3.8 or later
- A legal copy of Hollow Knight (Nintendo Switch version)
- Familiarity with terminal commands and Python programming

1.1 Software Used

- Terminal: macOS built-in Terminal or iTerm2
- Text Editor: Visual Studio Code (VSCode)
- Python IDE: PyCharm (optional)
- Emulator: Ryujinx
- Version Control: Git (optional)

2. Setting Up the Environment

2.1 Install Required Tools

Open Terminal and run the following commands:

```
# Install Homebrew (if not already installed)
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/
```

```
# Update Homebrew and install required packages
brew update
brew install [email protected] cmake boost

# Install pip and virtualenv
curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py
python3 get-pip.py
pip3 install virtualenv

# Install Visual Studio Code
brew install --cask visual-studio-code

# Install PyCharm (optional)
brew install --cask pycharm
```

2.2 Create a Virtual Environment

In Terminal, run:

```
mkdir hollowknight_rl_bot
cd hollowknight_rl_bot
virtualenv -p python3 venv
source venv/bin/activate
```

2.3 Install Python Libraries

With the virtual environment activated, run:

```
pip install numpy pandas matplotlib gym stable-baselines3 to
```

```
pip install opencv-python pyautogui mss
```

3. Installing and Configuring Ryujinx

3.1 Download Ryujinx

Visit the official Ryujinx website and download the latest macOS version.

3.2 Install Ryujinx

In Terminal, run:

```
cd ~/Downloads
unzip Ryujinx-*.zip
mv Ryujinx.app /Applications/
```

3.3 Configure Ryujinx

1. Open Ryujinx from the Applications folder
2. Go to Options > Settings
3. In the System tab, set your preferred resolution and enable docked mode
4. In the Input tab, configure your controls (we'll use these later for the bot)

4. Preparing Hollow Knight

4.1 Install Hollow Knight on Ryujinx

1. In Ryujinx, go to File > Open Ryujinx Folder
2. Navigate to the "games" folder
3. Copy your Hollow Knight NSP file into this folder
4. In Ryujinx, click on "Add new game directory" and select the games folder
5. Hollow Knight should now appear in your game list

4.2 Configure Game Settings

1. Launch Hollow Knight in Ryujinx
2. In the game's options menu, set the resolution to match your Ryujinx settings
3. Disable any visual effects that might interfere with the bot's perception

5. Creating the Reinforcement Learning Bot with OpenAI Gym

5.1 Understanding OpenAI Gym

OpenAI Gym provides a standardized interface for reinforcement learning environments. It defines a common API for interacting with an environment, making it easier to develop and compare different RL algorithms.

5.2 Set Up Project Structure

In Terminal, create the following directory structure:

```
mkdir -p src/{environment,agent,utils}
touch src/__init__.py src/environment/__init__.py src/agent/_
touch src/environment/hollow_knight_env.py src/agent/ppo_ager
touch main.py
```

5.3 Design the State Space

Open VSCode and create a new file `src/environment/hollow_knight_env.py` . Define what the bot can "see":

```
import gym
from gym import spaces
import numpy as np

class HollowKnightEnv(gym.Env):
    def __init__(self):
        super(HollowKnightEnv, self).__init__()

        # Define observation space
        self.observation_space = spaces.Box(low=0, high=255,
```

```
        # Other initialization code...
```

5.4 Define the Action Space

In the same file, define the action space:

```
class HollowKnightEnv(gym.Env):
    def __init__(self):
        # ... previous code ...

        # Define action space
        self.action_space = spaces.Discrete(8)  # 8 possible

        # Map actions to game controls
        self.action_map = {
            0: 'NOOP',
            1: 'MOVE_LEFT',
            2: 'MOVE_RIGHT',
            3: 'JUMP',
            4: 'ATTACK',
            5: 'DASH',
            6: 'CAST_SPELL',
            7: 'HEAL'
        }
```

5.5 Implement the Reward Function

Create a new file `src/utils/reward.py` and implement the reward function:

```
def calculate_reward(prev_state, current_state, action):
    reward = 0

    # Reward for moving right (progressing)
    if current_state['player_x'] > prev_state['player_x']:
        reward += 1

    # Reward for defeating enemies
    if current_state['enemies_defeated'] > prev_state['enemie
        reward += 10

    # Penalty for taking damage
    if current_state['player_health'] < prev_state['player_he
        reward -= 5

    # Large reward for reaching a new area or defeating a bos
    if current_state['area'] != prev_state['area']:
        reward += 100

    return reward
```

5.6 Complete the RL Environment

Back in `src/environment/hollow_knight_env.py`, implement the remaining methods:

```
from src.utils.reward import calculate_reward
from src.utils.screen_capture import capture_screen
from src.utils.image_processing import process_image
from src.utils.action_execution import execute_action
```

```
class HollowKnightEnv(gym.Env):
    # ... previous code ...

    def step(self, action):
        execute_action(self.action_map[action])
        next_state = self.get_state()
        reward = calculate_reward(self.current_state, next_st
        done = self.check_if_done()
        info = {}
        self.current_state = next_state
        return next_state, reward, done, info

    def reset(self):
        # Implement reset logic (e.g., restart game or load s
        self.current_state = self.get_initial_state()
        return self.current_state

    def render(self, mode='human'):
        # Game is already being rendered by Ryujinx
        pass

    def get_state(self):
        screen = capture_screen()
        return process_image(screen)

    def get_initial_state(self):
        # Implement logic to get the initial state
        pass

    def check_if_done(self):
        # Implement logic to check if the episode is finished
        pass
```

6. Integrating the Bot with Ryujinx

6.1 Screen Capture

In `src/utils/screen_capture.py`, implement screen capture:

```
from mss import mss
import numpy as np

def capture_screen(monitor={"top": 40, "left": 0, "width": 12
    with mss() as sct:
        screenshot = sct.grab(monitor)
    return np.array(screenshot)
```

6.2 Image Processing

In `src/utils/image_processing.py`, implement image processing:

```
import cv2

def process_image(image):
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    resized = cv2.resize(gray, (84, 84))
    return resized
```

6.3 Action Execution

In `src/utils/action_execution.py`, implement action execution:

```
import pyautogui

def execute_action(action):
    if action == 'MOVE_LEFT':
        pyautogui.keyDown('left')
        pyautogui.keyUp('left')
    elif action == 'MOVE_RIGHT':
        pyautogui.keyDown('right')
        pyautogui.keyUp('right')
    elif action == 'JUMP':
        pyautogui.keyDown('up')
        pyautogui.keyUp('up')
    # ... implement other actions
```

7. Training the Bot

7.1 Set Up the Training Loop

Create a new file `src/agent/ppo_agent.py`:

```
from stable_baselines3 import PPO
from stable_baselines3.common.callbacks import BaseCallback
```

```
class TensorboardCallback(BaseCallback):
    def __init__(self, verbose=0):
        super(TensorboardCallback, self).__init__(verbose)

    def _on_step(self) -> bool:
        self.logger.record('reward', self.training_env.get_at
        return True

def train_agent(env, total_timesteps=1000000):
    model = PPO("CnnPolicy", env, verbose=1, tensorboard_log=
    model.learn(total_timesteps=total_timesteps, callback=Ten
    return model
```

7.2 Implement Checkpointing

In the same file, add a function for checkpointing:

```
def train_with_checkpoints(env, total_timesteps=1000000, che
    model = PPO("CnnPolicy", env, verbose=1, tensorboard_log=
    for i in range(0, total_timesteps, checkpoint_interval):
        model.learn(total_timesteps=checkpoint_interval, rese
        model.save(f"hollowknight_model_{i}")
    return model
```

7.3 Main Training Script

In `main.py`, set up the main training script: