Duck Hunt Challenge

February 5, 2022

1 Introduction

This document is an onboarding guide for the ”Duck Hunt Computer Vision Challenge” for the UVic ECE 471 Computer Vision course. Students are challenged to use computer algorithms to get as high a score as possible on the modified classic NES game, Duck Hunt. A Python 3.7 module created using Pygame[1] and OpenAI Gym[2] has been provided to the students. The scripts and files described in this document are available at:

`https://github.com/dash-uvic/ece471_536-S2022/tree/main/duck-hunt`.

Reminder: The use of Reinforcement Learning algorithms is not allowed. Path finding algorithms, such as A-star search, are allowed.

2 Installation

The Duck Hunt Computer Vision challenge is provided to you as a wheel (`.whl`) file. A Google Colab variation has been provided if you choose to develop on Google Colab which includes the some of the instructions/commands below.

Follow the following installation process:

1. It's **highly** recommended you use GitHub/BitBucket for development, and create a private repository for your project;
2. Follow the guide ”Setup Virtual Environment” posted on BrightSpace to create a `virtenv` for **Python 3.7** or run the provided script create_venv.sh (Linux and OSX). **Make sure you are in your virtual environment for the following instruction**;
3. Download the project files from the link listed in the Introduction to your project directory;
4. Install the required packages listed in `requirements.txt` by running: `pip install -r requirements.txt`
5. Install the Duck Hunt package by running: `pip install --no-cache-dir ece471_duckhunt`

[1]https://www.pygame.org/news
[2]https://gym.openai.com/

2.1 Possible Issues

Unable to Install Package

You need to make sure you have created your virtual environment using Python 3.7. The wheel package was compiled for Python 3.7 and will not work if you use a different version.

Duck Hunt visualization is bad in Google Colab

Google Colab does not do live visualizations well. An example on how to create live visualizations from the Duck Hunt environment has been provided, but it is not ideal. I recommend using Google Colab for training purposes and your local machine to any live visualization tasks.

Doesn't work in Windows 10

This package is untested in Windows 10. It is recommended you use the WSL or Ubuntu distribution available in Windows 10. However, you may report Windows 10 bugs.

Report bugs and installation errors, with error messages/code, to #duckhunt-bug-reports on Slack

2.2 Example Files

You are provided with two example main files which are commented:

- `duck_hunt_main.py`
- `duck_hunt_main.ipynb`

A command-line interface is provided. The defaults are set to the demo parameters. To see all options, run:

```
python duck_hunt_main.py -h
```

A variation of `duck_hunt_main.py` will be used during the demo.

3 DuckHuntEnv

The Duck Hunt Challenge is encapsulate in a python module `ece471_duckhunt`. The game environment is run through the `DuckHunEnv` class. The goal of the game is to achieve a high score. The score is calculated by the "hit" accuracy; where for ducks that appear in a level, how many times did you correctly "hit" one. "Hits" are achieved by moving the "gun" (indicated by a cross-hair icon) so that it overlaps with a duck.

The`DuckHuntEnv` construct has the following parameters:

- level: Game Level to run (default=0, all levels);
- move_amount: How fast the gun should move (move_type=relative) (default=1);
- quiet: Iconify the game screen and do not show debugging output;

- randomize: Run game levels in random order;
- shape: Game screen size (width, height) (default="1024 768");
- duration: Override game length in seconds (default=60);
- seed: Set random seed for reproducability (default=None);

3.1 Solution Setup

You will create a python called: `solution.py`. You may setup the rest of your solution however you'd like, but this file is required for the demo. This file will contain a function `GetLocation` that we will use to run your code. This function will return a list of dictionaries. The dictionary will have two fields, (1) *coordinate* which contains the location you want the "gun" to move and (2) *move_type* which the type of move to use. The type of move can be: absolute or relative.

With *move_type=absolute* that indicates you are passing a tuple of list of (x,y) coordinates, while *move_type=relative* indicates you are passing an integer indicating discrete move: North = 0, North East = 1, East = 2, South East = 3, South = 4, South West = 5, West = 6, North West = 7, NOOP = 8. A *NOOP* is defined as a "no operation", and no new action is performed.

If your absolute coordinate it outside the game screen, a NOOP will be performed. The gun automatically "shoots" at every step.

The game does not "wait" for you algorithm to finish. **To ensure this, your algorithm is run in a Thread.**[3] You may modify or create your own file to construct the `DuckHuntEnv` environment for your own training and testing; include removing the multiprocessing code. However, during the demo multiprocessing will be used to run your code.

Once your solution has produced a set of locations to move the "gun" you will call the `step(coordinate, move_type)` function. This will perform your desired action on the game environment.

The `step()` function returns the following values:

- current_frame: np.dnarray (W,H,3), np.uint8, RGB: current frame after *coordinate* is applied;
- level_done: True if the current level is finished, False otherwise;
- game_done: True if all the levels are finished, False otherwise;
- info: dict containing current game information;

While the level is current running, the info dict contains:

- hits: number of ducks hit during this step;
- total_ducks: number of ducks so far in the level;
- time: time elapsed (in seconds);

While the level is finished, the info dict contains:

[3]Pygame is not thread-safe which is why this could not be bundled into DuckHuntEnv.

- hits: total number of ducks hit during the level;
- total_ducks: total number of ducks in the level;
- scores: all the level scores (accuracy) so far;

3.2 Levels

There are a total of 819 possible levels, with increasing levels of difficulty. During your demo, I will select a subset of levels in the interest of time. The levels are divided into 10 ”sets” which have the same base ”properties”. Level sets 1-3 have deterministic scenes, and non-deterministic (randomly generated) for levels 4-10. Your tasks is the analyze the data provided to you in order to determine what characteristics your solution must have. Your task in this project is to analyze the levels to find a solution (the location of a duck). **Note:** `seed` will set the `numpy` random seed. If you are using other packages, you will need to set their seeds separately.

The number levels per level set are:

- Level Set 1: levels 1-13
- Level Set 2: levels 13-28
- Level Set 3: levels 28-127
- Level Set 4: levels 127-226
- Level Set 5: levels 226-325
- Level Set 6: levels 325-424
- Level Set 7: levels 424-523
- Level Set 8: levels 523-622
- Level Set 9: levels 622-721
- Level Set 10: levels 721-820