Matthew Virgin

Dr. Roy Turner

COS 470/570

9 December 2022

OpenAI Gym & Flappy Bird, that's NEAT!

ABSTRACT:

Historically, game playing and AI often crossover, which each field being useful in some way to the other. Novel implementations of games using AI are prevalent today, and can be a useful way of testing a machine learning algorithm. OpenAI gym is an open source tool that allows users to select from a wide variety of pre-defined environments and attempt to solve the problems they present through any number of methods. Many of OpenAI gym's built in environments include classic games like Space Invaders, Breakout, or Pac Man. This paper will focus on a newer game, Flappy Bird, which is not built in to OpenAI gym but is one of the very first third party environments referenced in the documentation. The goal of this paper is to design a system that uses OpenAI gym to create an agent that can play the game of Flappy Bird as well as a human could, if not much better. This will be accomplished through the use of NEAT, or NeuroEvolution of Augmenting Topologies, a genetic algorithm that can be used to create evolving artificial neural networks. However, results show that while it is possible to implement an exceptional agent for Flappy Bird using NEAT, the environment that OpenAI gym references in its documentation is outdated, making it difficult to work with. More work must be done in the future to determine a more flexible alternative than the environment that OpenAI gym provides for Flappy Bird.

INTRODUCTION:

Games provide novel ideas and interesting state spaces to apply A.I. techniques to. There are no end of bots for various games being created all the time. Chess, Asteroids, almost any game imaginable has been "solved" through artificial intelligence techniques in some way. Current technology is making interacting with the environment these games represent easier and easier, with OpenAI gym being one of the most prolific tools for applying A.I to games and game-like environments [1]. Specifically, OpenAI gym is designed with reinforcement learning techniques in mind. In other words, OpenAI gym is built for training agents to take action in their environment based on their observations and the reward they might get for doing so. OpenAI gym is a python package that comes with a huge amount of built in environments that make getting started with some reinforcement learning algorithms simple. The implementation of these reinforcement learning algorithms are made even easier through the addition of the Ray RLlib package, or Ray's Reinforcement learning library [2]. This library comes with built in support for any environment that OpenAI gym pre defines, making setting up an algorithm as simple as choosing one from the variety of options that RLlib provides. However, the goal of this paper is to use genetic algorithms, not reinforcement learning algorithms, to create an agent that can play Flappy Bird on the same level as or better than the average human can. The environment OpenAI gym provides for Flappy Bird is not built-in, but instead developed by a third party, meaning that RLlib does not work with it out of the box. Given that the goal is to use genetic algorithms anyway, integrating NEAT with the OpenAI gym Flappy Bird environment is going to be the focus of this paper. NEAT, or NeuroEvolution of Augmenting Topologies is "an evolutionary algorithm that creates artificial neural networks" [3]. The algorithm uses node genes specified by a single neuron and connection genes which each specify a single connection between said neurons alongside a user-provided fitness function to achieve its result [3]. Ideally, OpenAI gym and the NEAT-Python package will be able to interact in such a way that makes creating a Flappy Bird agent that is better than a human somewhat simple.

RELATED WORK:

Other papers of the past have shown that using genetic algorithms to achieve a highly intelligent agent for Flappy Bird is quite possible. [4] describes the use of genetic algorithms and neural networks to achieve a favorable result in the game of Flappy Bird. [4] divides their implementation into two difficulty levels, which this paper's design is not going to do, meaning that implementing NEAT with the normal form of playing Flappy Bird will likely be quicker and simpler than [4]'s approach. While [4] does not use NEAT, it uses a machine learning algorithm based off of Neuro-Evolution, which is essentially the same concept but without the "augmenting topologies" part. [4] does not use OpenAI gym or NEAT-Python, instead opting for HTML5 programming and the Phaser Framework, which this paper will not use. [4] found that their agents would slowly become fit enough to score 600 points or more, and that any generation with a score of 150 or more were essentially immortal, with optimum fitness. This proves that the concept of using NEAT to implement a Flappy Bird agent is sound and even quite effective.

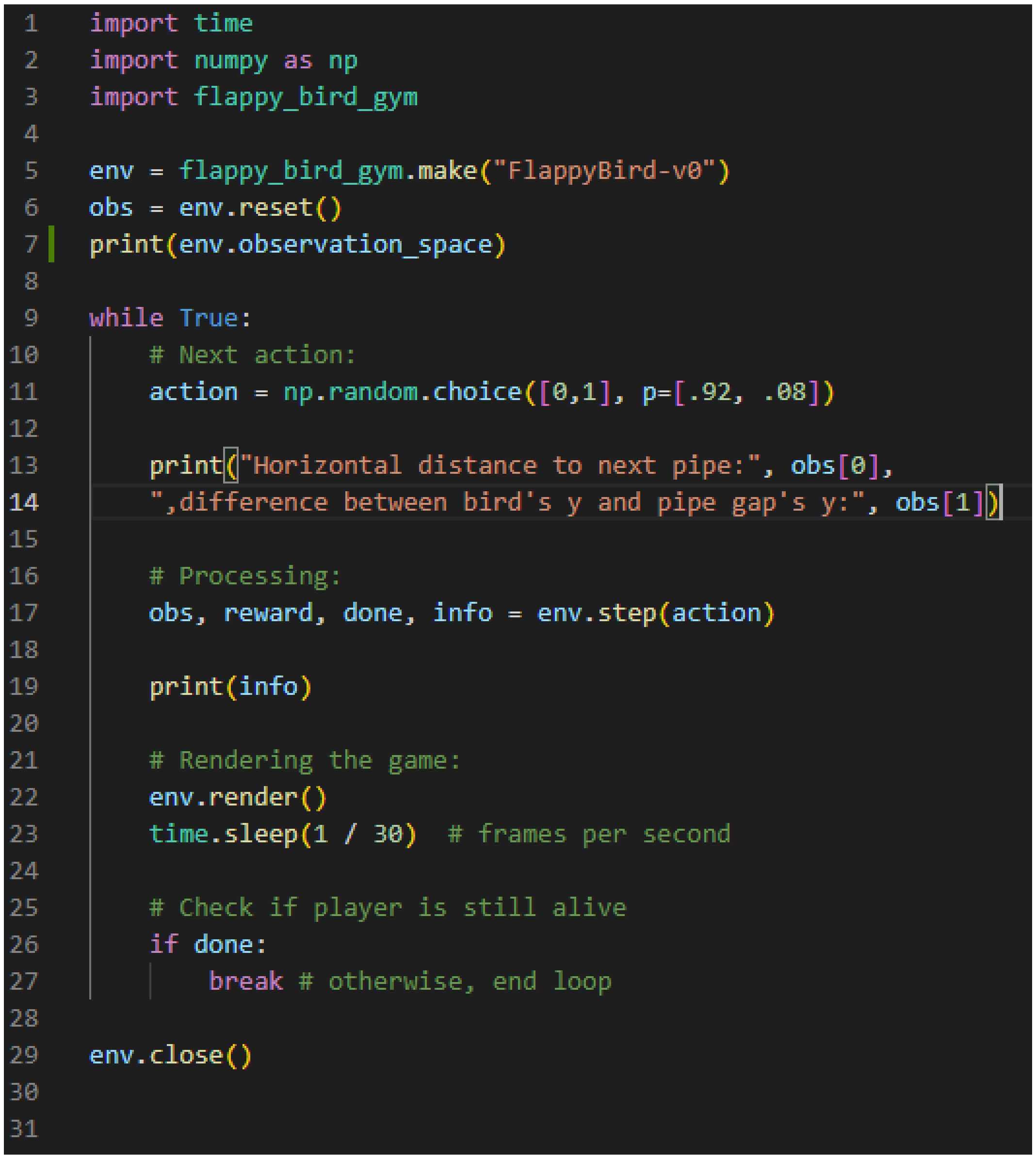
The OpenAI gym documentation [1] provides information on how the package can be used as well as where to find the third party Flappy Bird environment that this paper will be using. The documentation describes everything from the critical objects, methods, and data types of the package to the built-in environments and examples of how to use them. The most important part of this documentation is the section that describes the observations that OpenAI gym environments produce. The flappy_bird_gym environment's observations are defined by the

"Box" data type explained in the OpenAI gym documentation [5]. This datatype describes numpy arrays of elements that take values continuously in a range. The range of these elements for the flappy_bird_gym environment is positive to negative infinity, with an array shape of (2,) and elements of Float32 type [5]. This documentation is used to inform the design of achieving a Flappy Bird agent that uses NEAT and the OpenAI gym Flappy Bird environment to learn to play the game better than a human could.

Genetic algorithms and reinforcement learning are common ways of achieving a bot that can rival a human in game-playing activities. [6] describes reinforcement learning as an agent becoming proficient in unfamiliar environments given only percepts and infrequent rewards. Specifically, the authors describe Q-learning as especially effective in game-playing, as it requires no model in the learning or action-selection phases. This type of learning isn't as effective in complex environments, but Flappy Bird is not very complex, meaning Q-learning would likely work just as well on it as it did on the Atari games the authors mentioned. This fact is doubly true if the RLlib package worked easily with the flappy_bird_gym environment, as the agent could be trained via Q-learning with just a few lines [2]. The other implementations that are compared to Q-learning in [7] would also be quite easy to implement with RLlib if the two were compatible. Additionally, Russell and Norvig define genetic or evolutionary algorithms as those that create more and more fit generations through natural selection and mutation [6]. This kind of algorithm would be useful for a Flappy Bird bot because it could "train" the AI over the course of multiple generations. This fact makes NEAT the intuitive choice, as NEAT demonstrates "how it is possible for evolution to both optimize and complexify solutions simultaneously, making it possible to evolve increasingly complex solutions over time, thereby strengthening the analogy with biological evolution" [8, p.1]. In other words, NEAT is not only faster than other neuroevolution algorithms, but also closer to the biological version of evolution that inspired the creation of genetic algorithms in the first place. NEAT, therefore, should deliver the best solution for creating a Flappy Bird agent. In fact, this agent could be said to be analogous to a real-life bird, given NEAT's close parallels with how evolution actually works.

DESIGN / IMPLEMENTATION:

The original plan for this project was to implement a genetic Flappy Bird agent using solely the OpenAI gym environment. Given the lack of experience of the author, it quickly became apparent that an existing solution would be needed to expedite the implementation process, which led to the incorporation of NEAT. Time constraints and the inflexibility of the flappy_bird_gym environment then lead to this project becoming a design project. However, a "control" agent that can be used to compare the performance of a NEAT-influenced agent, once implemented, was developed via a randomly moving agent. A simple reinforcement learning algorithm that could be used to compare with NEAT and the control was also attempted through RLlib, but quickly abandoned after multiple dependency conflicts were discovered between RLlib and the flappy_bird_gym packages. These conflicts were attempted to be resolved by the author via a fork of the flappy-bird-gym github and the creation of a new package with updated requirements, flappy_bird_gymS, but this only led to more issues.

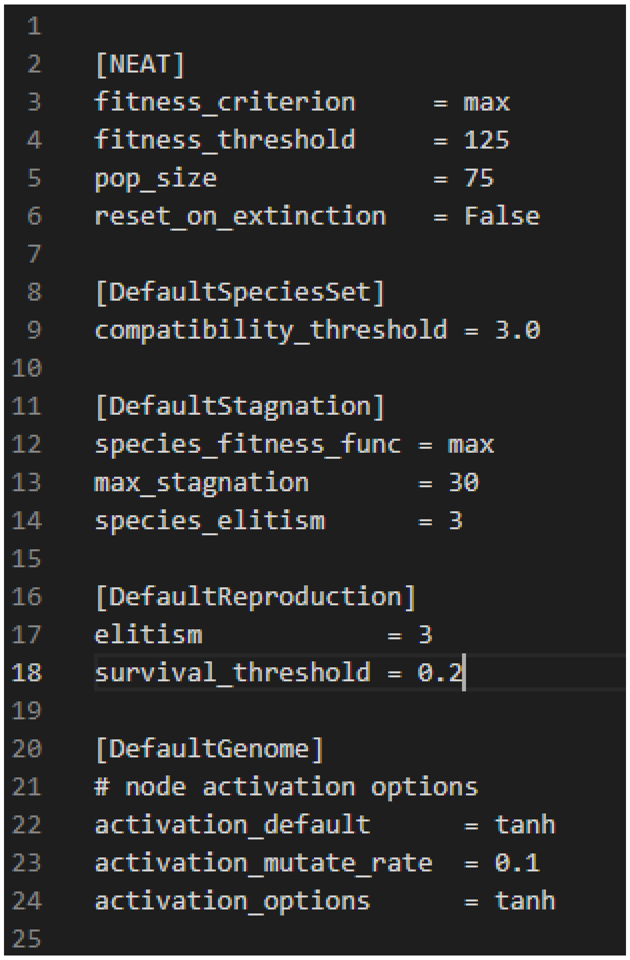


```
1   import time
2   import numpy as np
3   import flappy_bird_gym
4
5   env = flappy_bird_gym.make("FlappyBird-v0")
6   obs = env.reset()
7   print(env.observation_space)
8
9   while True:
10      # Next action:
11      action = np.random.choice([0,1], p=[.92, .08])
12
13      print("Horizontal distance to next pipe:", obs[0],
14      ",difference between bird's y and pipe gap's y:", obs[1])
15
16      # Processing:
17      obs, reward, done, info = env.step(action)
18
19      print(info)
20
21      # Rendering the game:
22      env.render()
23      time.sleep(1 / 30)  # frames per second
24
25      # Check if player is still alive
26      if done:
27          break # otherwise, end loop
28
29  env.close()
30
31
```

Figure 1: random agent

The figure above represents the implementation that was completed, a randomly moving Flappy Bird agent. An OpenAI gym environment is created on line 5, whose initial observations are instantiated on line 6 by calling the reset method, which sets the environment back to defaults, beginning a new game. An action is then chosen on line 11, choosing between 0 or 1. 1 means “flap” and 0 means “do not flap.” To prevent the bird from immediately flying off the screen, the choice is heavily weighted towards “do not flap,” emulating how a human would play the game, as the “flap” button only needs to be tapped occasionally to achieve success. The action is then fed to the environment’s step method, which runs one timestamp of the environment’s dynamics [1]. This action also sets obs, reward, done, and info for each pass of the loop. Info is then printed to display the score the agent has currently achieved, and the environment is rendered at 30 frames per second until done is true and the loop is broken.

This random agent implementation also represents the basic flow of training agents with reinforcement learning models on an environment through OpenAI gym, as each action is defined based on previous observations before new ones come in. To train with NEAT, the most important aspect is the configuration file, which will define things like the fitness function used and number of samples per generation. An example of this file can be seen below:

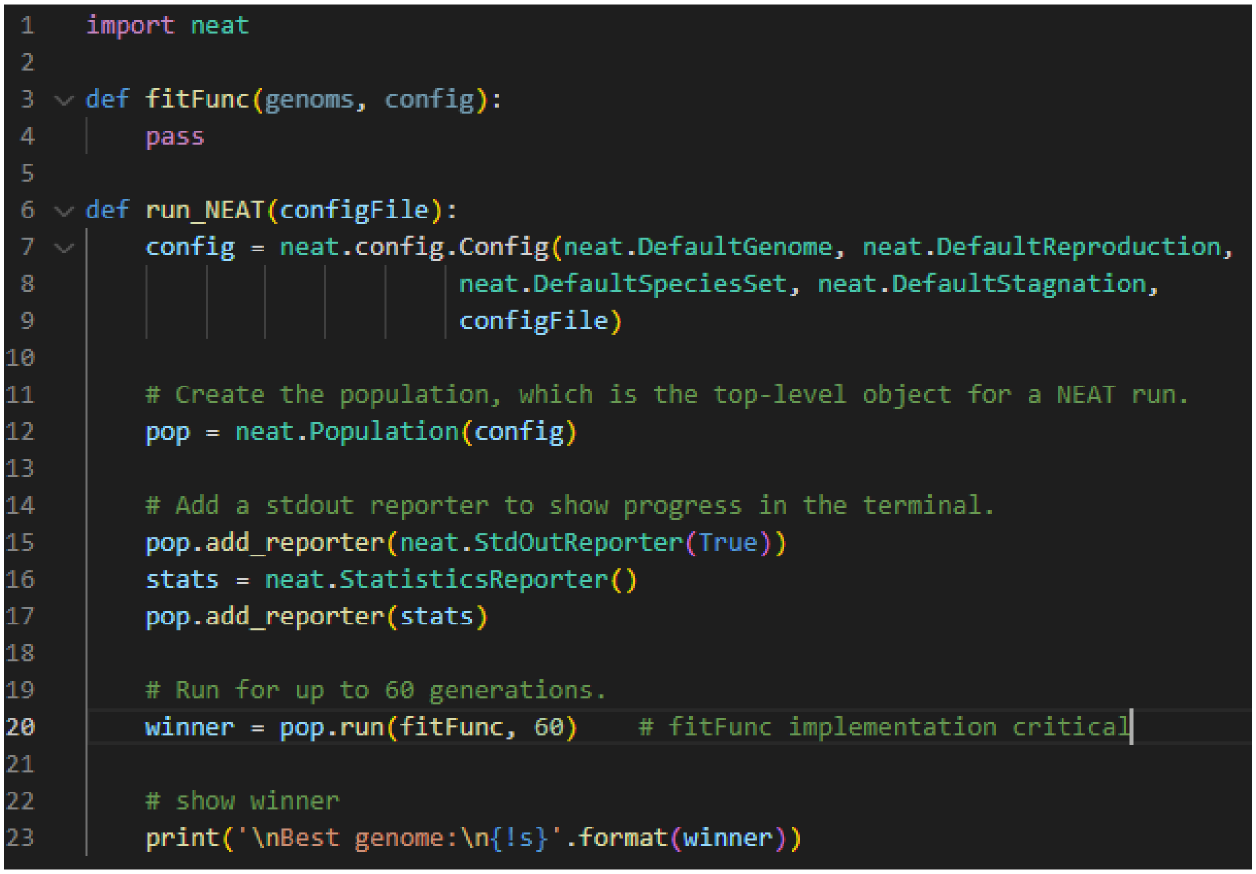


```

[NEAT]
fitness_criterion     = max
fitness_threshold     = 125
pop_size              = 75
reset_on_extinction   = False

[DefaultSpeciesSet]
compatibility_threshold = 3.0

[DefaultStagnation]
species_fitness_func = max
max_stagnation       = 30
species_elitism      = 3

[DefaultReproduction]
elitism            = 3
survival_threshold = 0.2

[DefaultGenome]
# node activation options
activation_default      = tanh
activation_mutate_rate  = 0.1
activation_options      = tanh

```

Figure 2: Important Sections of NEAT config file (adapted from [8])

The complete file can be found in this project's github along with the code for the random agent: https://github.com/mvirgin/570-project. A full explanation of every config option can also be found in [3]. The above snapshot demonstrates some of the most important aspects of the config file. The variables under [NEAT] are the parameters taken by the algorithm. Every other name in [brackets] represents a built in class, and the variables underneath are its attributes. A fitness_criterion of max means the most fit of the population will be picked. Fitness_threshold = 125 means that once the fitness reaches 125, the bird is considered immortal, the same idea used in [4]. Pop_size = 75 simply means each generation will consist of 75 individuals [3]. Reset_on_extinction set to false means that a new population will not be created if all species go extinct because of stagnation. A compatibility_threshold of 3 means that species with a genomic distance less than 3 are considered the same species. A max_stagnation of 30 means that species that have not improved in 30 generations will be removed for being stagnant, and a species_elitism of 3 means that 3 of those species will not be removed to prevent total extinction. Elitism = 3 on the [DefaultReproduction] class means that 3 individuals will be preserved as-is between generations. A survival threshold of 0.2 means that 0.2 of a species are allowed to reproduce each generation. [DefaultGenome] has the most attributes in this configuration file example, the most important of which is not listed above. The most important attributes on the DefaultGenome class are: num_hidden, num_inputs, and num_outputs, which define the number of hidden layers, the number of input neurons, and the number of output neurons, respectively. These values can be played around with to change results and determine what works best, as [3] does.

Actually getting NEAT up and running is quite simple, it's the fitness function that's the issue, as stated in [9].



```python
import neat

def fitFunc(genoms, config):
    pass

def run_NEAT(configFile):
    config = neat.config.Config(neat.DefaultGenome, neat.DefaultReproduction,
                                neat.DefaultSpeciesSet, neat.DefaultStagnation,
                                configFile)

    # Create the population, which is the top-level object for a NEAT run.
    pop = neat.Population(config)

    # Add a stdout reporter to show progress in the terminal.
    pop.add_reporter(neat.StdOutReporter(True))
    stats = neat.StatisticsReporter()
    pop.add_reporter(stats)

    # Run for up to 60 generations.
    winner = pop.run(fitFunc, 60)    # fitFunc implementation critical

    # show winner
    print('\nBest genome:\n{!s}'.format(winner))
```

Figure 3: run function (adapted from [9])

The figure above demonstrates the run function skeleton that most NEAT related problems will use. The important part to focus on is the fitFunc function, which will calculate the fitness for each individual of a species. However, this is where problems arise with OpenAI gym. As [9] mentions, it is much more efficient to have as many birds/agents as needed running at the same time than it is to run each one after the other, one at a time. However, the way the flappy_bird_gym is set up just doesn't support that method without some major modifications. The fitFunc would have to be done the slow way, one at a time, as the while loop operates in the random agent program.

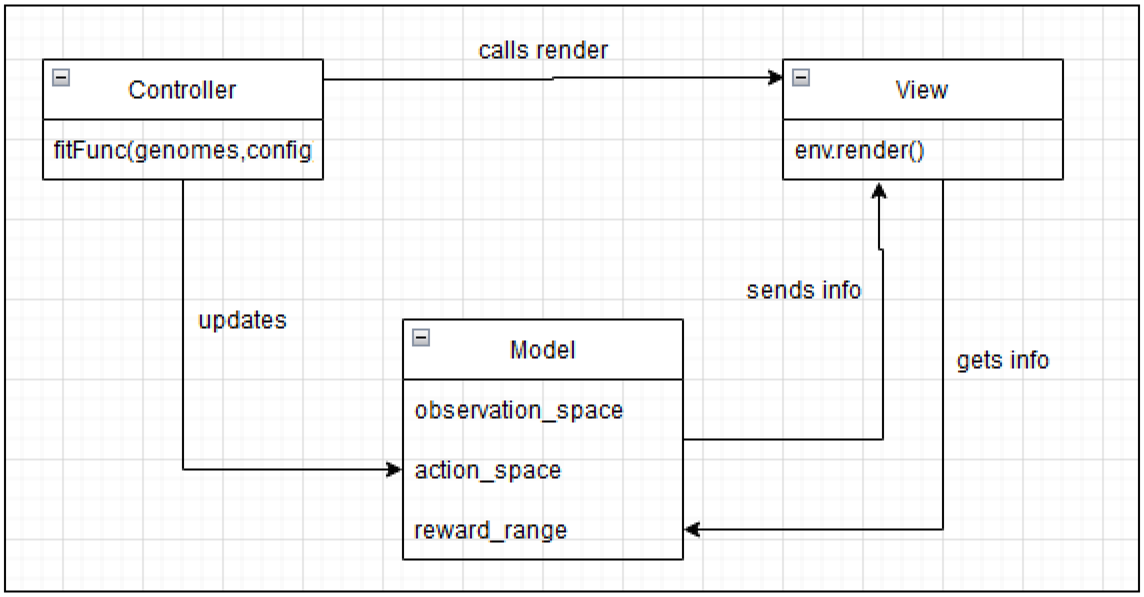
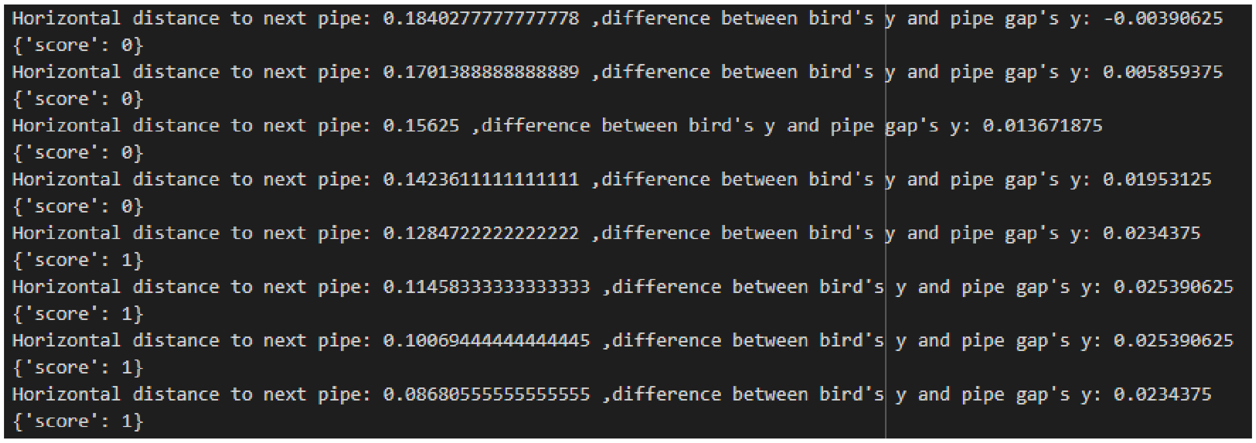


Figure 4: Model-View-Controller Architecture (template from [10])

The above image represents the system architecture for this paper's proposed design of a Flappy Bird bot that uses NEAT and openAI gym. While simple, this architecture encapsulates the basics of the design. The fitness function will call render as its running to display the bird moving on the screen and update the model, which communicates with the environment to keep giving the render method new information.

RESULTS:

Because little implementation was done for this project, there are few results to show off. The exNEATconfig.txt and NEATsetup.py files featured in the project github don't actually do anything, and serve only as examples. The randTest.py file works fine, however, and demonstrates a Flappy Bird agent that flaps randomly, with a large preference for not flapping. Out of about 100 runs, the random agent achieved a score of 1 twice. The exOutput.txt file contains the output of one of these runs.



```
Horizontal distance to next pipe: 0.1840277777777778 ,difference between bird's y and pipe gap's y: -0.00390625
{'score': 0}
Horizontal distance to next pipe: 0.1701388888888889 ,difference between bird's y and pipe gap's y: 0.005859375
{'score': 0}
Horizontal distance to next pipe: 0.15625 ,difference between bird's y and pipe gap's y: 0.013671875
{'score': 0}
Horizontal distance to next pipe: 0.1423611111111111 ,difference between bird's y and pipe gap's y: 0.01953125
{'score': 0}
Horizontal distance to next pipe: 0.1284722222222222 ,difference between bird's y and pipe gap's y: 0.0234375
{'score': 1}
Horizontal distance to next pipe: 0.11458333333333333 ,difference between bird's y and pipe gap's y: 0.025390625
{'score': 1}
Horizontal distance to next pipe: 0.10069444444444445 ,difference between bird's y and pipe gap's y: 0.025390625
{'score': 1}
Horizontal distance to next pipe: 0.08680555555555555 ,difference between bird's y and pipe gap's y: 0.0234375
{'score': 1}
```

Figure 5: Random Agent Achieves a Score of 1

The above image demonstrates the exact moment the agent passed between the first set of pipes in the environment and earned a score of 1. Note that there are 30 lines of output printed per second, displaying the observations of the agent. Such observations include the agent's horizontal (x) distance to the next pipe, as well as the difference between the agent's y pos and the gap in the pipe's y pos. The goal was to compare NEAT's performance with the random agent as well as one trained with reinforcement learning, but that will have to be saved for the future.

DISCUSSION:

Overall, it may be that OpenAI gym was a bad choice for the environment if using NEAT was the goal. The provided Flappy Bird solution is quite inflexible, as it's only meant for "single-agent reinforcement learning algorithms" [1]. This means that getting the flappy_bird_gym to work with NEAT wasn't guaranteed from the start, even though it should be possible with some modifications. Additionally, there was trouble getting RLlib to work because of dependency issues (because flappy_bird_game uses outdated versions of the gym and numpy packages), as discussed above, meaning that a reinforcement learning model couldn't be prepared for the project to have ready to compare with the random agent or NEAT agent. It's likely that the state space was too large for Flappy Bird for a simple implementation to be effective anyways, as there are about 146,640 possible states given that the screen is 288 x 512 and the bird is 34 x 24. [7] discusses this same issue, although they get around it by using deep learning techniques, which was not the focus of this project. It is definitely possible to implement this type of project, as it's been done before in [4] and [9], but the flappy_bird_gym environment recommended by the OpenAI gym documentation is simply too inflexible in its current state. In the future, forking the github for flappy-bird-gym and modifying it to support NEAT-Python's needs instead of attempting to force a genetic algorithm shaped object into a reinforcement learning algorithm shaped hole would make the design and implementation for this project much simpler. Additionally, more research could be done into NEAT and other genetic algorithms in order to obtain a deeper understanding of them that makes implementation easier.

REFERENCES:

[1] Gedam, Pradyun. “OpenAI Gym Documentation.” Third Party Environments-Gym Documentation, Sphinx, 2022, www.gymlibrary.dev/environments/third_party_environments/.

[2] Ray Team Staff. “Environments.” Ray RLlib Environments, Ray Team, 2022, https://docs.ray.io/en/latest/rllib/rllib-env.html

[3] CodeReclaimers Staff. “NEAT Overview.” NEAT-Python Docs, CodeReclaimers, 2019, https://neat-python.readthedocs.io/en/latest/neat_overview.html.

[4] Mishra, Yash, et al. “Performance Analysis of Flappy Bird Playing Agent Using Neural Network and Genetic Algorithm.” Communications in Computer and Information Science, 13 Nov. 2019, pp. 253–265., doi:10.1007/978-981-15-1384-8_21.

[5] Talendar (2021) flappy-bird-gym [Source code]. https://github.com/Talendar/flappy-bird-gym

[6] Russell, Stuart J., and Peter Norvig. Artificial Intelligence: A Modern Approach. Pearson Education Limited, 2022

[7] LVu, Tai, and Leon Tran. “FlapAI bird: training an agent to play flappy bird using reinforcement learning techniques.” arXiv preprint arXiv:2003.09579 (2020)

[8] “Python Flappy Bird AI Tutorial (with NEAT) - NEAT Configuration and Explanation.” Youtube, uploaded by Tech With Tim, Aug 18, 2019, https://www.youtube.com/watch?v=MPFWsRjDmnU&list=PLkYi-U4Q0f_ulvDVTFVSAXdwmEDIjM_Z4&index=13

[9] “Python Flappy Bird AI Tutorial (with NEAT) - Implementing NEAT/Creating Fitness Function”

[10] Ghanavati, Sepideh. “Lecture 7 & 10 - Architectural Design” COS 420/520: Intro Software Engineering / Software Engineering 1. 2022, University of Maine, Orono. Class Lecture.