FLAPPY BIRD IMPLEMENTATION USING AI

Ayushi Singh[1], Naman Tyagi[2], Sahil Khunger[3], Vipul Chaudhary[4]
[1, 2, 3, 4]Student
Department of Information Technology,
[1,2,3,4] Dr. Akhilesh Das Gupta Institute of Technology and Management, New Delhi

Abstract- This paper presents a minimal training strategy based on genetic algorithm and reinforcement learning where an agent is capable of playing the Flappy Bird game itself using NEAT algorithm. Here artificial agents were trained to take the most favorable action at each game instant. And the machine learns tobetter itself at a particular task through repetitive iterations. NEAT Algorithm uses a ANN (artificial neural network) along with the fitness function to maximize the score of the current generation and replace the old population with this newly generated population for better performance .The idea of actual biological evolution is implemented here based on Darwinian Natural Selection which consists of three properties (Heredity, Variation & Selection) here some members of population have achance to pass on their genetic information for better results. It isalso referred as 'survival of the fittest'. Hence, using these strategies to achieve low complexity and better performance.

Index Terms- Artificial intelligence, flappy bird, genetic algorithm, neuroevolution, reinforcement learning.

1. INTRODUCTION

The Neuroevolution technique is the artificial evolution of neural network using genetic algorithm. It is a technique to evolve artificial neural networks in unsupervised learning problems [1]. NEAT is best and optimized way compared to Descending Gradient algorithm like Backpropagation.It does not depend onthe output value, which was used to generate error to optimizethe network. Neuroevolution is used as part of the reinforcement learning, NEAT evolves dense neural network node by node, and it will change the values of connections and will randomly add other nodes to find topology for neural network that works best. The reinforcement learning is the training of models to make sequence of decisions. The agent learns to achieve the goal in a complex environment and the computer employs trial and error method to come up with a solution to the problem to get the machine do what the programmer wants. The AI gets either reward or penalties based on the action it performs; the agent istrained in such a way that it takes the best action to maximize thetotal reward. [1]

The goal of this flappy Bird game is to simply keep the bird alive as long as possible by passing it through the pipes without colliding with them and achieving the maximum game points. In this game we apply a Neuroevolution algorithm, the choice of NEAT is related to the fact that it starts from simpler configuration agents and complicated it over the generations, generally increasing the topology, so that the solution found is the simplest.

The rest of the paper is structured as follows. Section II consists of related work based on Neuroevolution in other games.

In the Methodology section we provide the details of strategy, tools and techniques used describing the training and performance of the Reinforcement Learning based agent .The Result section consistssummary of the whole work.

2. RELATED WORK AND BACKGROUND

In this section we will discuss about the key concepts of reinforcement learning, genetic algorithms, neuroevolution giving background details of research in this area and their applications.

Reinforcement Learning (RL) [2] is an area where agents take action in an environment in order to maximizethe reward. It learns from interaction with the environment similar to human beings [2].In reinforcement learning, an artificial intelligence faces a game-like situation. An illustration of agents interacting with environment can be found below in Figure 1[2]..Another example of reinforcement learning is autonomous cars. The situation becomes ideal if the AI performs better with no instructions on driving the car hence fully automatic.[3]

One more example can be "Learning to move"[4], which aims at producing complex motions, such as quick turn and walk-to-stand transitions to make an agent learn how to walk and run [4] .The main challenge in reinforcement learning lies in preparing the simulation environment, which is dependent on the task performed.

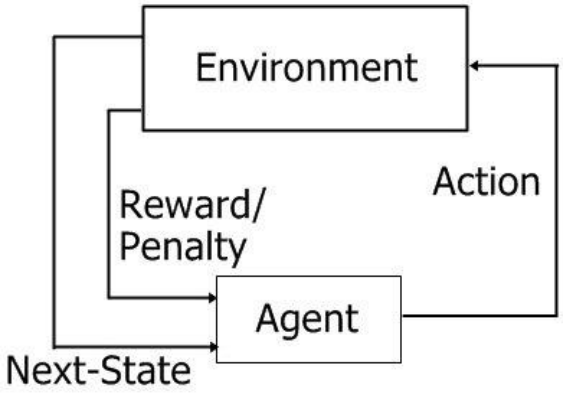


Figure 1. The Agent and its environment

Genetic Algorithms (GA) is based on the idea of actual biological evolution. From a given population with certain characteristics, the fittest have the highest probability in being transferred to the next generation. In order to evolve to a new generation GA applies Selection, Crossover and Mutation[5].Neuroevolution uses this technique of a GA to evolve the weights and architecture of a Neural Network.[6] Neuroevolution has been used in many other fields like designing games such as chess, checkers etc.[7] .In this paper Flappy Bird we focus on evolving the new population of birds that is agent using GA and Multiagent Reinforcement Learning with variations using TanhH Function to reduce time [8] .NEAT algorithmis used to progressively improve the performance of Flappy Bird.

3. METHODOLOGY

Three components are essential to this work: fitness function calculation, presented in Section III-A; how to expose the scenario to the agent, presented in Section III- B; and phenotype settings, presented in Section III-C.

A. Fitness:

To compute the fitness of the agent that is the bird Scenario Fitness Components (SFC) is used:

- Distance Traveled (DT): is calculated each time an interaction of the agent with the environment that is the background of the game;
- Score: which is calculated each time flappy bird passed through those pipes;
- Y Factor (ΔY): It is calculated by the difference between the y coordinate of the agent and the midpoint of top and bottom pipe, this value is obtained when an agent fails in any scenario. The formula is defined as:

$$\Delta Y = y_{agent} - y_{passage} \quad [9]$$

The main usage of Y Factor is that it enables the penalize based on the performance of the agent in the fitness function which is calculated on the basis of fitness score that is how far the bird goes without hitting the obstacles. In Fig. 2 three Factors of Y are highlighted, ΔY_1, ΔY_2 and ΔY_3, each of them corresponding to a different agent of the same population which will create different scenarios defines as collision .When this occurs the interaction with the environment ceases and the agent's performance is measured. The Y Factor is used for fitness calculation. The goal is to ensure that agents closer to the middle of pipe that is the free space are considered better than others further away.

As shown in Fig. 2, if all agents failed at same time, theywill have the same DT and score, but the Y Factors would be different, such that $\Delta Y_1 < \Delta Y_2 < \Delta Y_3$, showing that agent 1 is closer to the pipe. Also, the value of ΔY is absolute since it tries to penalize the performance of the agent based on the distance from the pipe, even if it was above or below the agent.

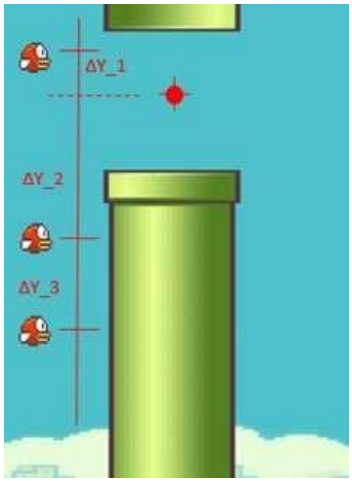


Figure 2. Y factor shows difference between three agents with the same DT.[9]

SFCs are combined in a single equation, known as scenariofitness function (SFF):

$$SFF = \alpha \times DT_S + \beta \times Scores_S - \gamma \times \Delta Y_S \quad [9]$$

This equation shows that the fitness of an agent which is flappy bird is a linear combination of SFCs and 3 constant weights α, β, and γ. The goal of the agent is to cover the maximum distance without colliding, hence achieving the highest score.

The *S* subscript corresponds to the standardized version of thecomponent:

- $DT_S = DT/DT_{max}$, where DT_{max} is the DT when covering themaximum number of pipes.
- $Scores_S = Scores/Scores_{max}$, where $Scores_{max}$ is themaximum score obtained by agent.
- $\Delta Y_S = \Delta Y/W_h$, where W_h is the height of the game window,which will be highest value of the y coordinate.

Obtaining SFFs from the game and combining them into anagent fitness function (AFF) calculates the fitness of agent :

$$[9]\ AFF = \frac{\sum_{i=1}^{MS}(k_i \text{ X } SFF_i)}{\sum_{i=1}^{MS} k_i}$$

Thus, it shows that the fitness of an agent in generation is given by a weighted average which is based on the fitness of each scenario. Here *MS taken* is the sum of number of scenarios, which have different aspects in the program [9].

B. Scenarios

The main objective regarding the way the scenariois exposed to the agent during training is to reduce thenumber of pipes while still preserving the ability of the algorithm to converge in a short time. Three scenarios are represented with variation of the gap between the pipes as shown in the Fig. 4. Hence, the total number of scenarios, i.e. *MS* is 3.

The gaps were taken as 0 pixels, and approximately80 pixels and 160 pixels. Since 0 and 160 pixels are the height of the screen and 80 pixels becomes the midpoint .An agent that performs well in these scenarios with different gaps will be able to handle any kind of scenario, as it has learned how to pass through the small, medium and large gaps [9].

Since the number of pipe pair i.e. top and bottom pipe making single pair, in each scenario is three, $Scores_{max}$ = 3 and DT_{max} = 195, such that 195 is the DT for the agent that transposes three pairs of pipes. The reason behind three scenarios is supported by the fact that a second scenario allows a faster convergence, since, the agent is getting better in solving challenges.

In the context, three pairs of pipes were chosen as shown in Fig. 4. (a) and (b) shows a flat gap between pipes; (c) and (e)shows fall gaps; (d) and (f) show climb gaps. These types of gaps are composed with lots of pipe pairs with different gaps so that AI can execute in different scenario.

C. Network Phenotype, Parameters and Tools

The agent's phenotype is represented by an ANN that has three input neurons, one output neuron and with an internal structure that will be defined after the training of network by the neuroevolutionary algorithm. The output is theprobability of performing a jump by an agent. Fig. 5 shows all inputs in a frame, which are:

- Ay: Agent's y coordinate;
- By: Y coordinate of the tip of bottom pipe;
- Cy: Y coordinate of the center of the free spacebetween the pipe pair.

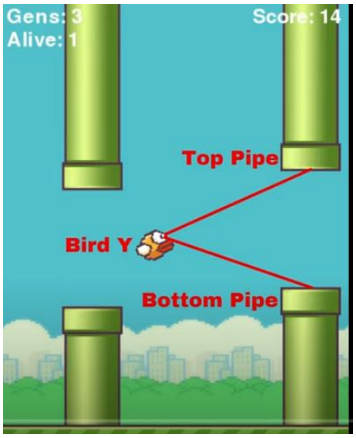
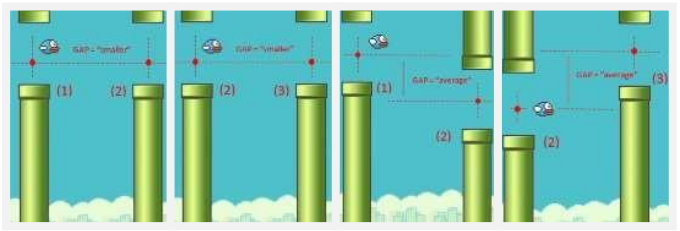
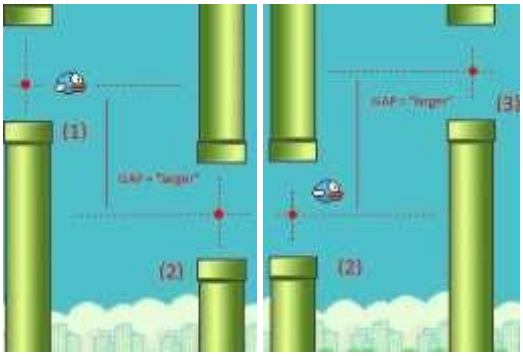


Figure 3. Representation



a b c d



e f

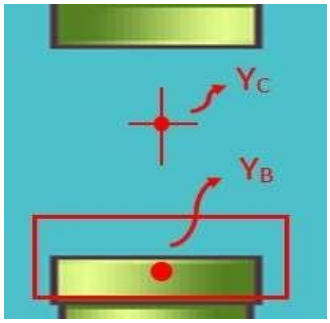


Figure 5. Inputs[9]

When the information given by the environment (Ay, By and Cy) is received by the agent, it gets processed by the net-work, brings about a probability of jump being executed. This probability determines the next step to execute according to the Algo: Action = Jump if (Out ≥b was 0.5.) else none, where b

In the configuration of the NEAT parameters, the following values were established, which can then be used to reproduce this work:

Population: a population of 20 individuals is used which enables a faster execution and is not a very big population, and also it is large enough to allow a genomic diversity which won't produce local minimawhich in turn would stop the evolution.

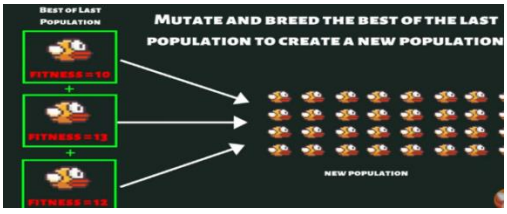


Figure 6. Best of last population

- Compatibility threshold: the value given to this term was 3.1, which was small enough to not completely prevent the formation of species within the population and it was large enough to not create many species initially. An interesting thing to note is that intersections between species can lead to problems in optimization when a large number of species in a population was present; also this was the case when the population was extremely small.
- Elitism: an elitism of 18 individuals was chosen, this value when compared to the previous parameter was equal to 60% of the population. This value allowed the preservation of innovations and it was small enough to not incur in slump or the absence of innovations
- Mutation rate: this parameter had a value of 0.05, which caused the connections to not activate immediately as the topology increased, they were activated gradually.
- Weight and Bias: the average initial generation of weights and bias were 0 and 0.01, respectively, with a standard deviation of 1.3 in both. These values allow a slightly more varied distribution when the weights are generated, thus getting closer to the topologies of better performance.
- Probabilities to add or remove connections: the likelihood of adding connections and the likelihood of removing connections were set to 0.7 and 0.2, Respectively. The values showed a greater fondness for a robust topology through connection creation, a proposal aimed at solving complex problems. If the topology does not increase, but the optimization finds good individuals it is because a simpler topology was sufficient for the problem, since NEAT starts from less complex to more complex topologies.
- Probabilities to add or remove nodes: the probability of adding nodes was set to 0.7 and the probability of removing nodes was set to 0.2, as in the above parameters, and they have a similar explanation

The algorithm ran for 100 generations using the activation function tanh . Given the simplicity of the agent's decision and since Descending Gradient is not part of the process, this function uses all of its gradient power without drawbacks.

In the calculations of the SFFs and the AFF the following values were used and gave a prominent result:

SFF (α = 1.0, β = 1.0, γ = 0.08): Note that the weight of the Y Factor is very low when compared to the others alpha and beta. This was due to two reasons:

- The Y Factor before being normalized gets a small value when compared to the other parameters during the generations and when the normalization is done this distance is lost, so a small value of γ allows a reduction of the *Delta Y* magnitude; and
- At the beginning of the generations the Y Factor has very high values, which is the opposite of its primary function of being a differentiation of similar fitness when fine tuning the adjustment in their scores, which can harm the convergence speed and thus slowing the whole program.

AFF (k_1 = 1.0, k_2 = 2.0, k_3 = 6.0): SFFs weights were given a very big value for increasing the strength in a more difficult environment. This was the reason why Scenario 2 had a slightly greater weight than Scenario 1 (easy) and Scenario 3 (hard) has a much greater weight than Scenario 2. This strategy allows a faster convergence of the algorithm since the agent finds the best performance faster [9].

This work was constructed using NEAT-Python [10] and PyGame Learning Environment (PLE) [11] .PLE is the library used for the agent's interaction in the environment, which is automatically executed when there is a need to calculate its fitness, thus making it communicate with the environment through his sensors and actuators. This relationship is best explained by Fig. 7.

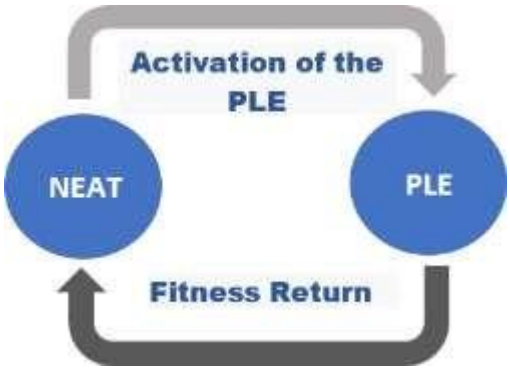


Figure 7. Relationship between NEAT and PLE.[9]

To execute the steps discussed above, some slight changes were made to the PLE to allow the definition of the structure of the scenario before the beginning of the optimization. These changes are thus important to build the three types of scenarios with different gaps between the pairs of pipes.

4. RESULTS

The results of fitness and scores are presented in Section IV-A. The speciation chart is shown in Section IV-B. Finally, the final network topology is presented in Section IV-C.

A. Fitness and Scores

Fig. 8 present the fitness results. The x-axis of the chart corresponds to generations, from the beginning going up to 100, while the y-axis corresponds to the average fitness (blue line) and the best fitness (red line) on every generation. It can be seen that in about generation 20 the fitness stabilizes until the end of the tests. The algorithm is able to achieve an optimal score since the first generations, showing the robustness of the applied strategy.

The score chart in Fig. 9 is very similar. The red color line is the best score achieved in every generation while the blue

color line is the average score. The stabilization of the scores occurs again around the 20th generation, agreeing with the fitness. In both figures the mean values stabilize in values that represent around 2/3 of the maximum values in each chart. The maximum score is 2 in the chart, which occurs when DT = 195, *scores* = 3 and ΔY = 0. This means that the agent was able to pass through all three pairs of pipes and did not shock into anything. This implies that the $SFF = \alpha + \beta = 1.0 + 1.0 = 2.0$ leading to $AFF = ⨍(1.0 \times 2.0 + 2.0 \times 2.0 + 6.0 \times 2.0)/(1.0 + 2.0 + 6.0)] = (18.0/9.0) = 2.0$, since $k1 = 1.0$, $k2 = 2.0$ and $k3 = 6.0$.[9]

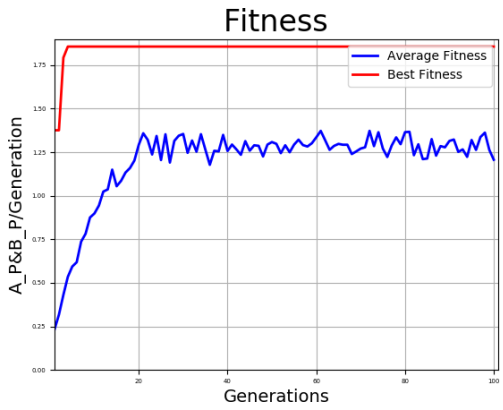


Figure 8. Agent fitness chart [9]

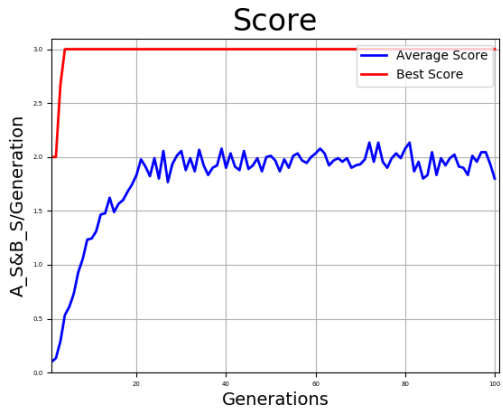


Figure 9. Agent fitness chart [9]

On the other hand, the maximum score possible is equal to three, which equates to the transpose of the three pairs of pipes. When it plays the game with random objects, an agent transposes all pipes continuously which is clear in Figure 10 in which an agent got a score higher than thousand and is currently playing the game. Minimal training strategy was very successfully to generate agents with optimal behaviors

B. Speciation

Figure 11 shows the speciation chart, in which the x-axis informs the generations and the y-axis informs the size of the species that is at most 20. Since only a single species was enough to converge and solve the game, only a single color is shown.

C. Topology

The phenotype of the best agent of the last generation is shown in Fig. 12. The phenotypeis a perceptron, a very simple model of an artificial neural network. The network weights computed for each input are:

- **W Ay:** 0.6285.

Figure 10. Agent achieving a score of 1136 and still counting

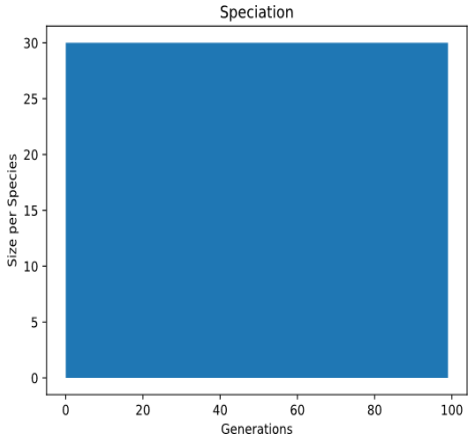


Figure 11.Speciation Chart. [9]

- **W By:** -1.5107.
- **W Cy:** 1.6638.
- **Bias Node:** -1.0770.

Ay ensures that the agentis with a small value of the y coordinate, i.e. the agent is far above the passage and the tanh function may result in a very low jump chance. This implies that the agent will tend to go down by the action of gravity. When theagent has a high value of the y coordinate, i.e. the agent isunder the passage, the tanh function will tend to result in a very big jump chance, which will lead the agent goes up.

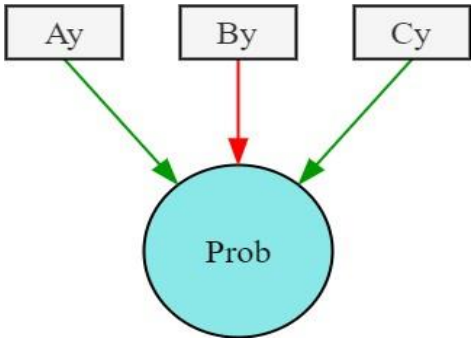


Figure 12. Topology found by NEAT.[9]

Evaluating all of the three results together it can be seen that, our methodology turned the game into a problem simple to solve by the algorithm. Those results show that training agents using a very restricted training environment, only three types of obstacles, can lead to optimal behaviors.

5. CONCLUSION

In this work, a efficient training strategy to generate agents capable of achieving excellent scores in the game Flappy Bird by using only three different types of scenarios with different types of obstacles to train the agents by evolving a neural network to stay alive without dying in an environment with unlimited pair of pipes with random heights. The fitness calculation used ensured that the objective of an agent in a reduced environment represented its goal in a real run of the game.

Because the obstacles used have gaps near the environment borders and an intermediate one, when the agent manages to maximize its result in these three cases it then masters all possible variations within these limits. Considering that NEAT always searches for the simplest solution to a problem and that the fitness presented together with the division into three scenarios help the NEAT find a perceptron network architecture using a single species, this solution is the simplest. The techniques discussed in this work helped the algorithm to find this solution in a short time,thus proving its effectiveness.

Using a population of only 20 individuals, the evolution- ary algorithm was able to converge to an optimal behavior after about twenty generations. At this point, an agent is able to play the game indefinitely. This shows that this strategy can find optimal solutions in a short number of generations. As future works, this minimal training strategy can be tested in simple platform games that show some kind of action repetition through their stages, and also with different types of learning algorithms

REFERENCES

[1] K. O. Stanley and R. Miikkulainen, “Evolving neural networks through augmenting topologies,” Evolutionary Computation, vol. 10, no. 2, pp. 99–127, 2002. [Online] Available: http://nn.cs.utexas.edu/?stanley:ec02

[2] R. S. Sutton and A. G. Barto, Introduction to Reinforcement Learning, 1st ed. Cambridge, MA, USA: MIT Press , 1998.

[3] Sallab, Ahmad & Abdou, Mohammed & Perot, Etienne & Yogamani, Senthil. (2017). Deep Reinforcement Learning framework for Autonomous Driving. Electronic Imaging. 2017. 70-76. 10.2352/ISSN.2470-1173.2017.19.AVM-023.

[4] Song, Seungmoon & Kidziński, Łukasz & Peng, Xue & Ong, Carmichael & Hicks, Jennifer & Levine, Serge & Atkeson, Christopher & Delp, Scot. (2020). Deep reinforcement learning for modeling human locomotion control in neuromechanical simulation. 10.1101/2020.08.11.246801.

[5] Lingaraj, Haldurai. (2016). A Study on Genetic Algorithm and its Applications. International Journal of Computer Sciences and Engineering. 4. 139-143.

[6] Felipe Petroski Such, Vashisht Madhavan, Edoardo Conti Joel Lehman Kenneth O. Stanley, Jeff Clune(20 April 2018) ,”Deep Neuroevolution: Genetic Algorithms are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning”, arXiv:1712.06567v3

[7] Joel Lehman and Risto Miikkulainen (2013) Neuro-evolution. Scholarpedia, 8(6):30977.

[8] C. Rosset, C. Cevallos, and I. Mukherjee, “Cooperative multi-agent reinforcement learning for flappy bird *,” Semantic Scholar, 2016.

[9] M. G. Cordeiro, P. B. S. Serafim, Y. L. B. Nogueira,C. A. Vidal and J. B. Cavalcante Neto, "A Minimal Training Strategy to Play Flappy Bird Indefinitely with NEAT," 2019 18th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames), Rio de Janeiro, Brazil, 2019, pp. 21-28, doi:10.1109/SBGames.2019.00014

[10] A. McIntyre, M. Kallada, C. G. Miguel, and C. F. da Silva, “neat-python,” https://github.com/CodeReclaimers/neat-python.

[11] N. Tasfi, “Pygame learning environment,” https://github.com/ ntasfi/PyGame-Learning-Environment, 2016

