Reinforcement Learning, Flappy Bird

Guruprerana Shabadi & Luca Bonengel

CSE204 Project


GitHub repository:
github.com/lucabonengel/FlappyProject

Outline of the presentation

I. The game, Flappy Bird

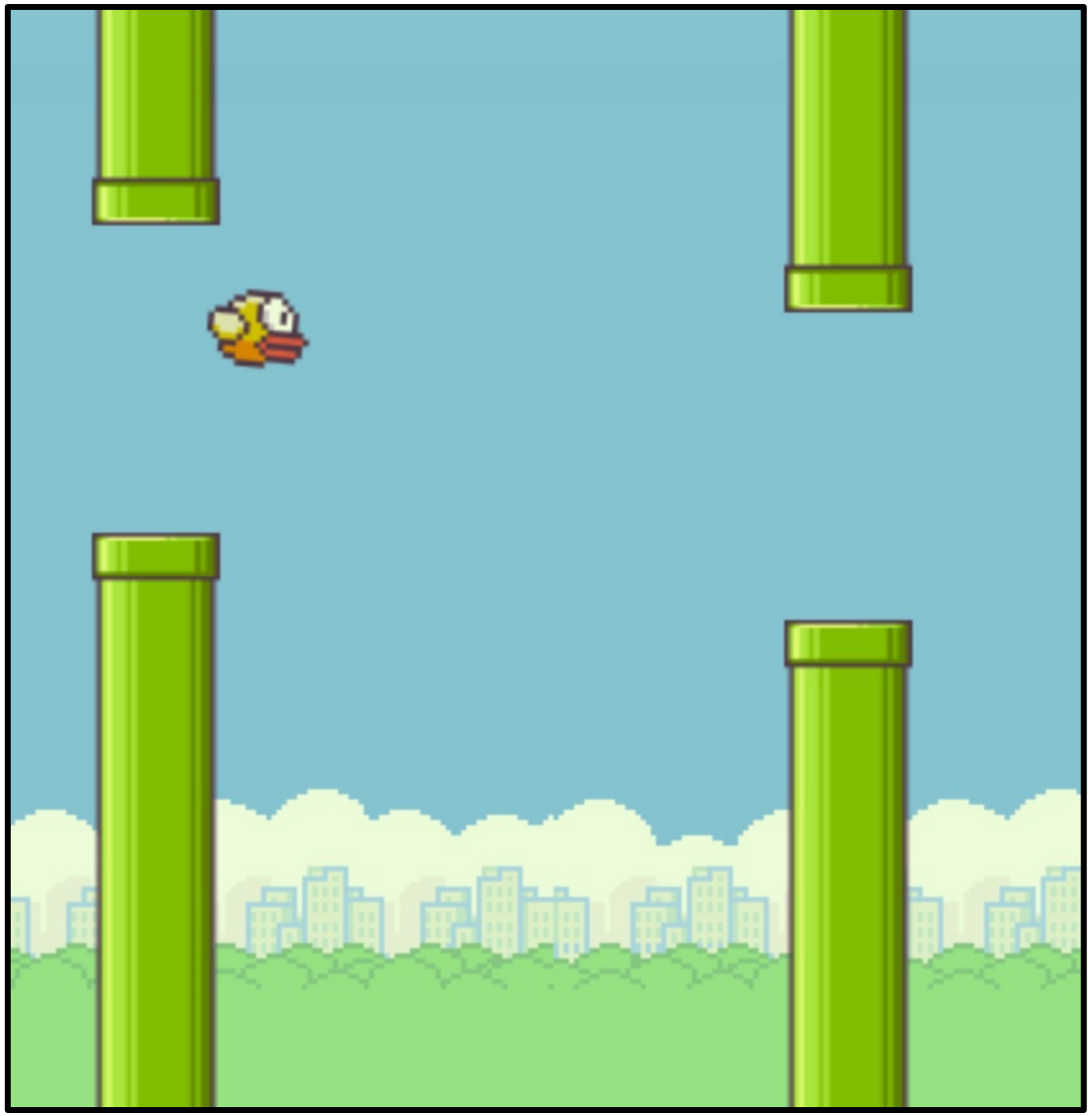
II. Algorithm 1: Q-learning algorithm
   - Description of the algorithm
   - Results

III. Algorithm 2: Neuroevolution algorithm
   - Description of the algorithm
   - Results

IV. Comparison of the two algorithms

I. The game, Flappy Bird

- An (in)famous game consisting in controlling a bird, attempting to fly between columns of green pipes without hitting them
- Game project in CSE104



II. Algorithm 1: Q-learning algorithm

- Simple model requiring no information about the environment
- Gradually build up the Q-Matrix at each instance of the gameplay
- Maximising cumulative rewards: each value in matrix represents expected reward given a state and action

Initialized

Q-Table		Actions					
		South (0)	North (1)	East (2)	West (3)	Pickup (4)	Dropoff (5)
States	0	0	0	0	0	0	0
	. . .	. . .	. . .	. . .	. . .	. . .	. . .
	327	0	0	0	0	0	0
	. . .	. . .	. . .	. . .	. . .	. . .	. . .
	499	0	0	0	0	0	0

Training

Q-Table		Actions					
		South (0)	North (1)	East (2)	West (3)	Pickup (4)	Dropoff (5)
States	0	0	0	0	0	0	0
	. . .	. . .	. . .	. . .	. . .	. . .	. . .
	328	-2.30108105	-1.97092096	-2.30357004	-2.20591839	-10.3607344	-8.5583017
	. . .	. . .	. . .	. . .	. . .	. . .	. . .
	499	9.96984239	4.02706992	12.96022777	29	3.32877873	3.38230603

The update equation

$$\mathcal{A} = \{\text{flap}, \text{no-flap}\}$$

$$R(s_t) = \begin{cases} 0, & \text{if bird is alive} \\ -1000 & \text{if bird is dead} \end{cases}$$

$$Q_{t+1}(s_t, a_t) \leftarrow Q_t(s_t, a_t) + \alpha \left[R(s_{t+1}) + \gamma \max_{a \in \mathcal{A}} Q_t(s_{t+1}, a) - Q_t(s_t, a_t) \right]$$

The set of states

Optimal inputs:

- y-coordinate of bird
- y-coordinate of next obstacle
- whether bird is alive

$$\mathcal{S} = [0, 500] \times [0, 500] \times \{\text{alive, not alive}\}$$

$$\text{shape}(Q) = \text{shape}(\mathcal{S} \times \mathcal{A}) = 500 \times 500 \times 2 \times 2$$

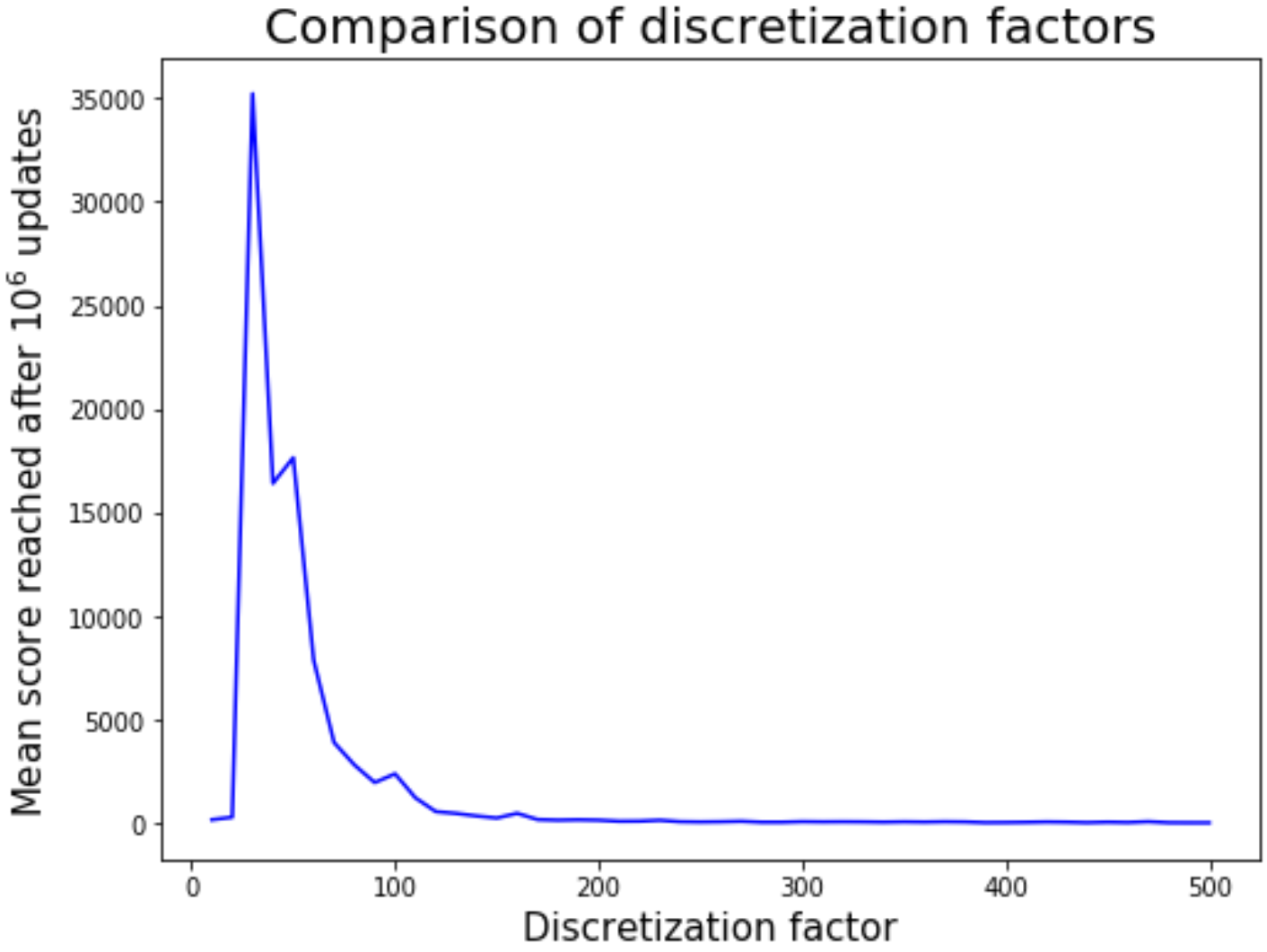
$$30 \times 30 \times 2 \times 2.$$

Big Q-matrix = **long training**

Small Q-matrix = **fast training**

BIG IDEA: Cut up grid into blocks

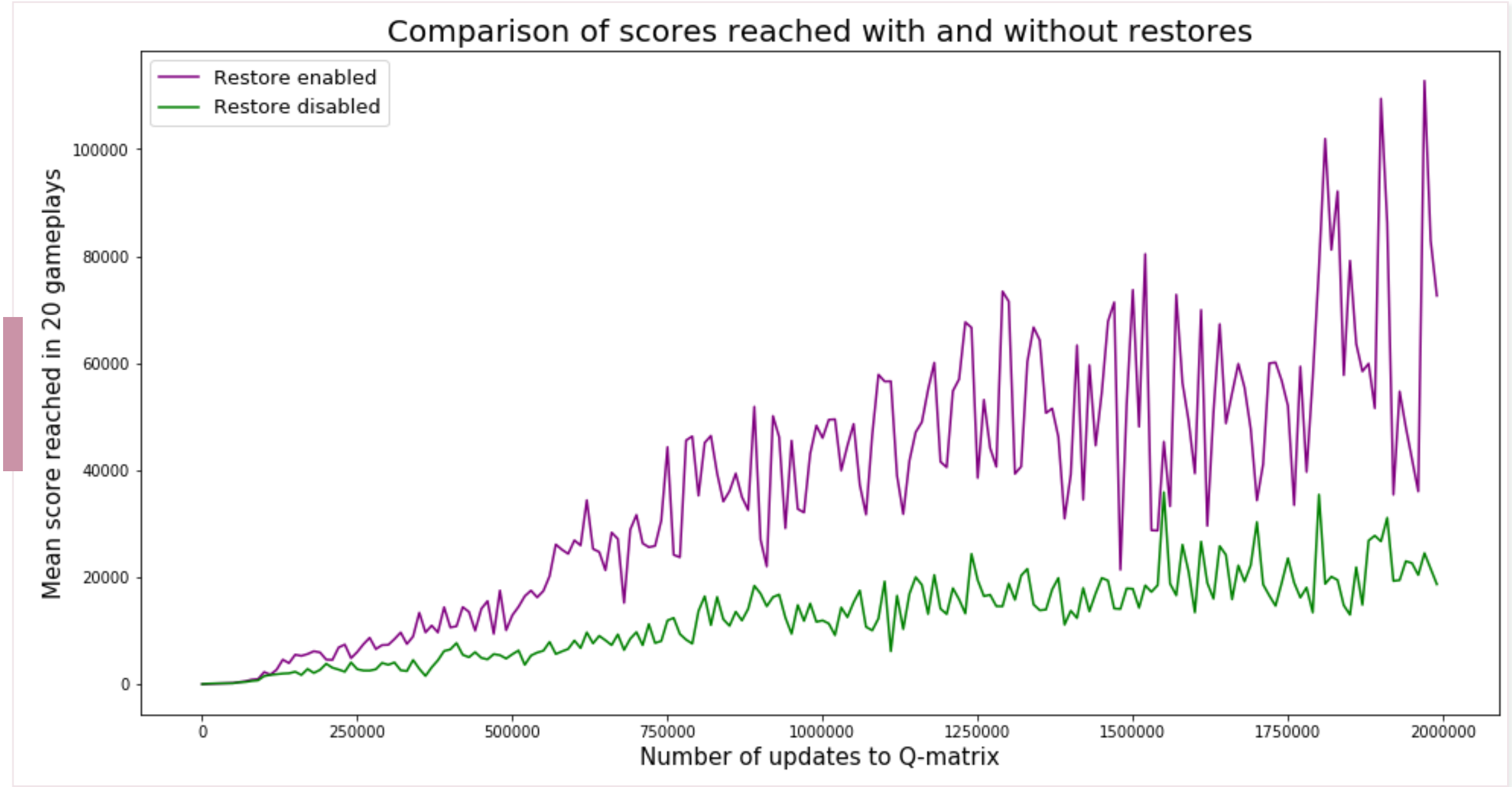
Why cut into 30 blocks?



Still slow to learn: requires next big *idea*

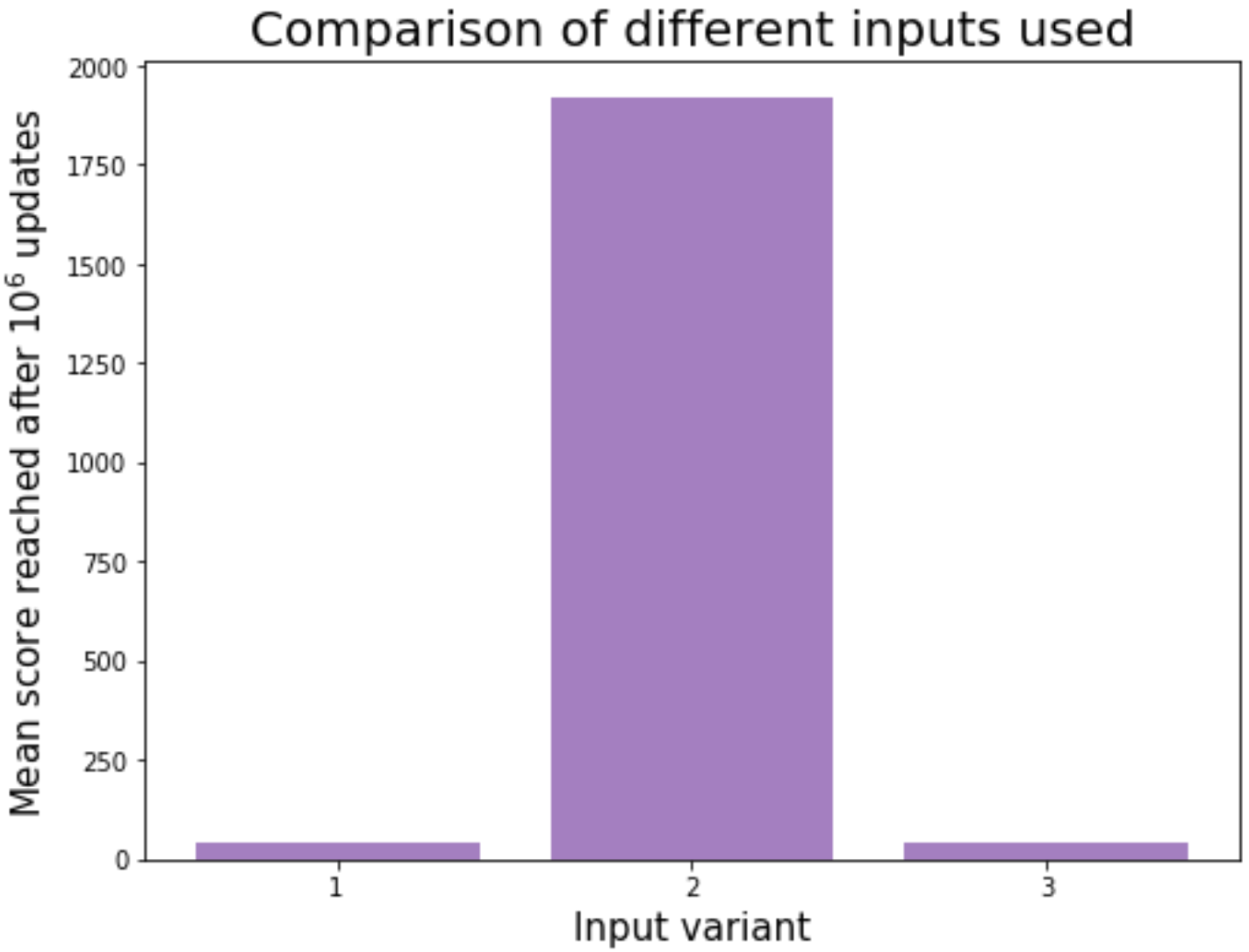
```
initialize Q-Matrix

while True:
    Use Q-matrix to determine next action
    Take action
    Update state
    Update Q-matrix

    if bird dies:
        restart the game
```

```
initialize Q-Matrix

while True:
    Use Q-matrix to determine next action
    Take action
    Update state
    Update Q-matrix

    if bird dies:
        rewind the game by 70 instances
```


Comparison of scores reached with and without restores
Restore enabled
Restore disabled
Mean score reached in 20 gameplays
0
20000
40000
60000
80000
100000
0
250000
500000
750000
1000000
1250000
1500000
1750000
2000000
Number of updates to Q-matrix

Inputs to use

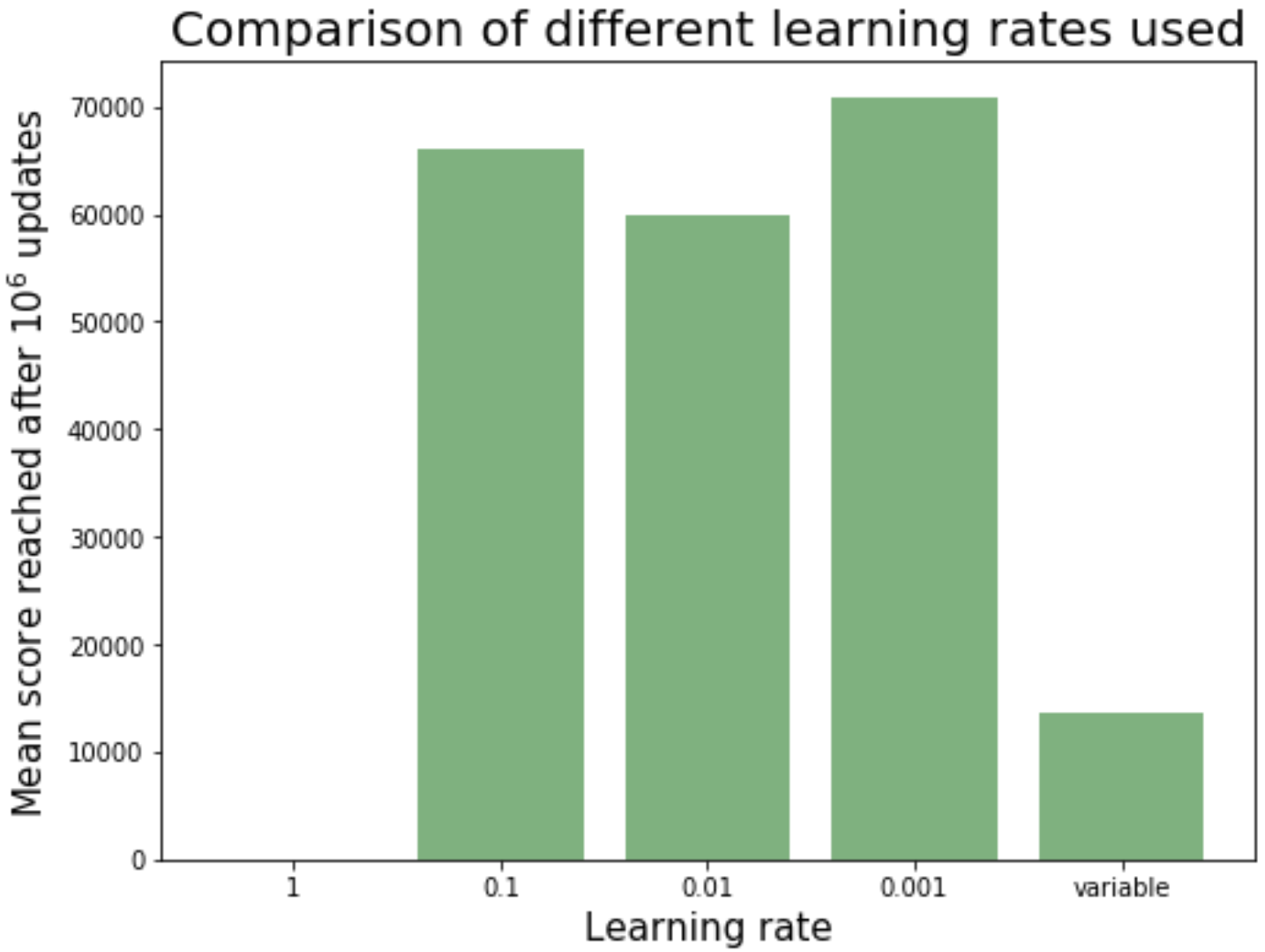
- **Variant 1**: y-coordinate of the bird, vertical distance to the next obstacle (pipe), and horizontal distance to the next obstacle
- **Variant 2**: y-coordinate of the bird, and y-coordinate of the next obstacle
- **Variant 3**: y-coordinate of the bird, horizontal distance to the next obstacle, and y-coordinate of the next obstacle



Comparison of learning rates

$$\alpha_t = \frac{1}{1 + \sum_{k=0}^{t} \mathbb{1}_{\{s_k = s_t, a_k = a_t\}}}$$

Constant works better.



Results

1 million updates to the Q-matrix = very high scores of above **100,000** easily.

10 million updates = a ***perfect bird which never dies***.

Big training time: approx. **30 min** on our computers

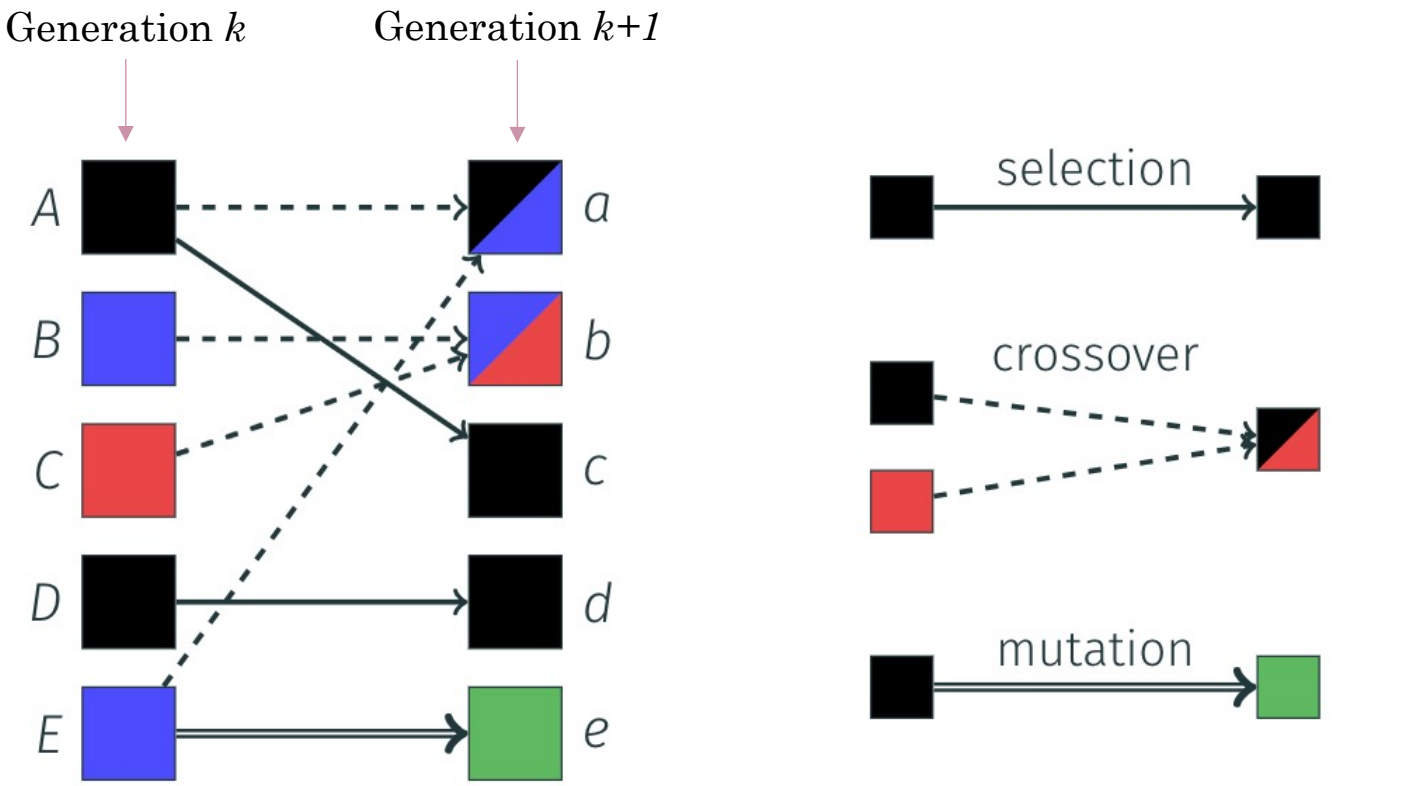
III. Algorithm 2: Neuroevolution algorithm

Neuroevolution is a machine learning technique that applies evolutionary algorithms to construct neural networks.

- Population of neural networks evolves in order to find a network that solves the given task.
- 1 neural network for 1 bird.
- First generation: weights initialised with random values in [-1, 1].

Generation of the next populations

- **Elitism** (selection) consists in keeping the best neural networks as they are. These birds automatically survive to the next generation.
- **Crossovers** are created by mixing the neural networks of a pair of well performing parents and adding some mutations (slightly changing some weights).
- **New random neural networks** are also generated.

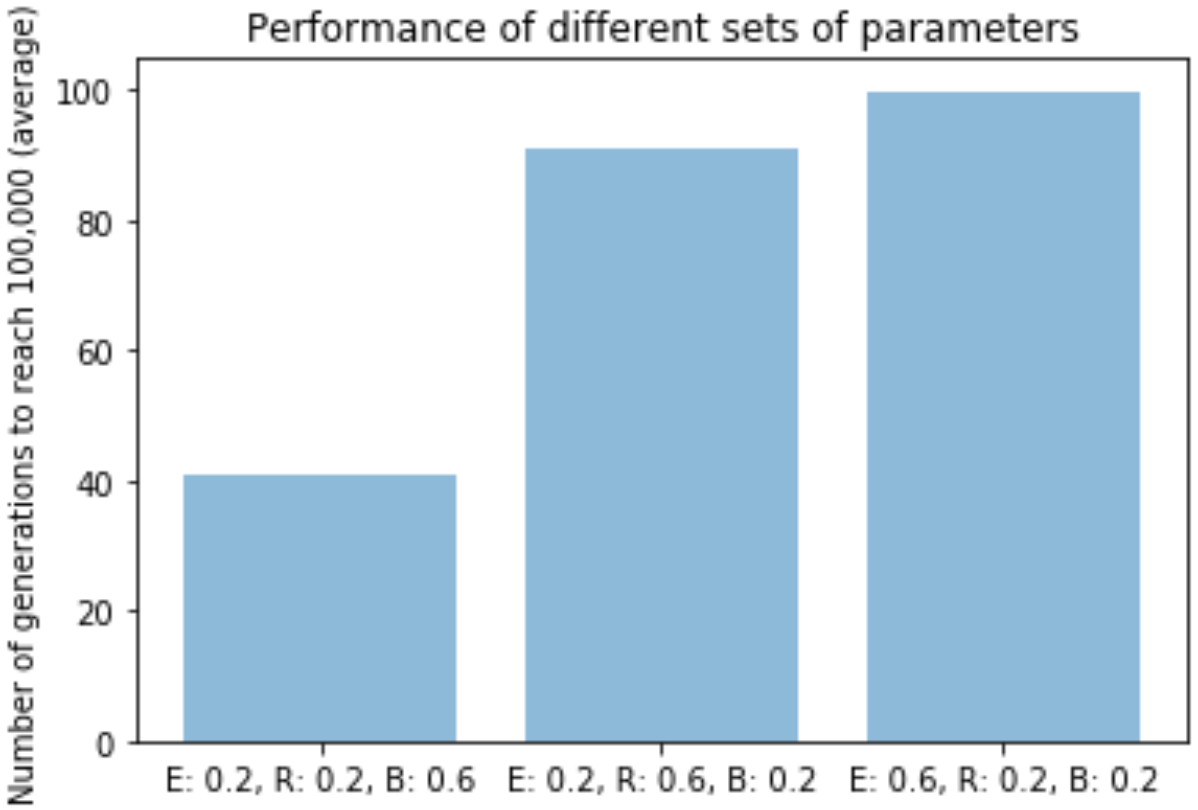


Methods for creating the neural networks of the next generations

Many other ways of creating neural networks for the next generation (e.g. slighty mutating a good performing one)

What proportions should we use?

- E: elitism proportion
- R: random networks proportion
- B: breeding proportion

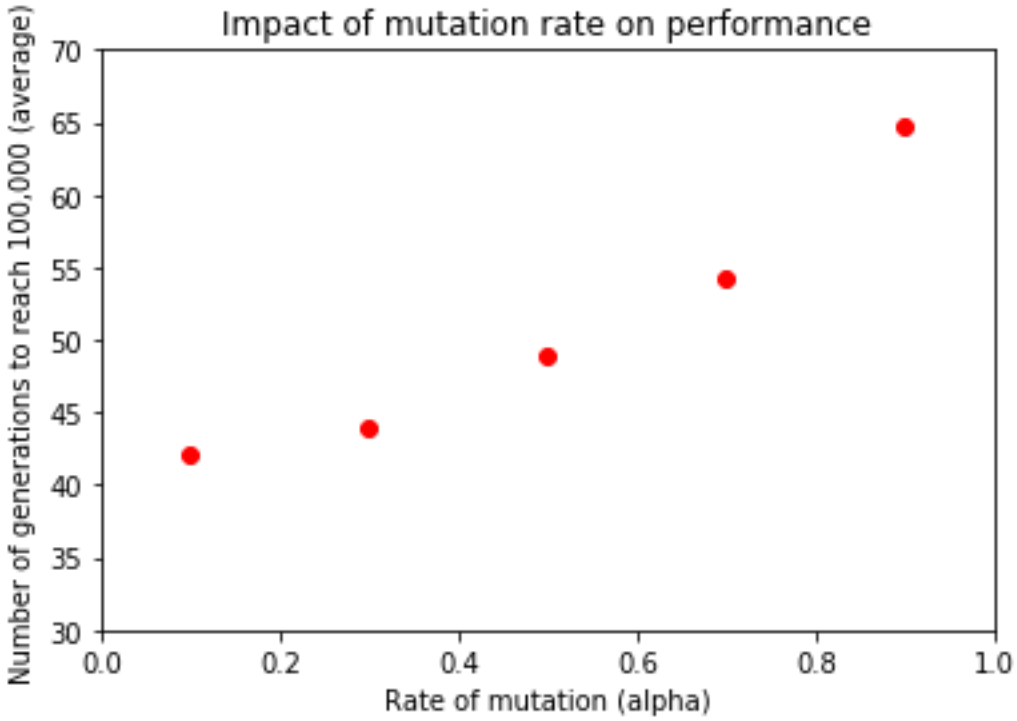


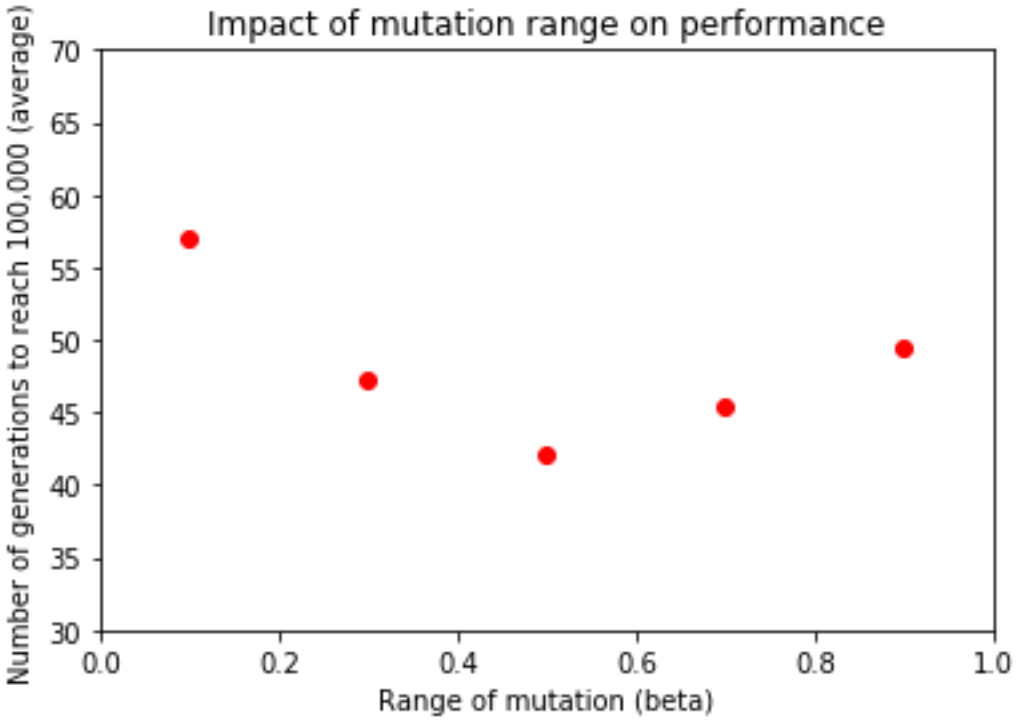
Fixing parameters

We fix the following parameters for the following slides:

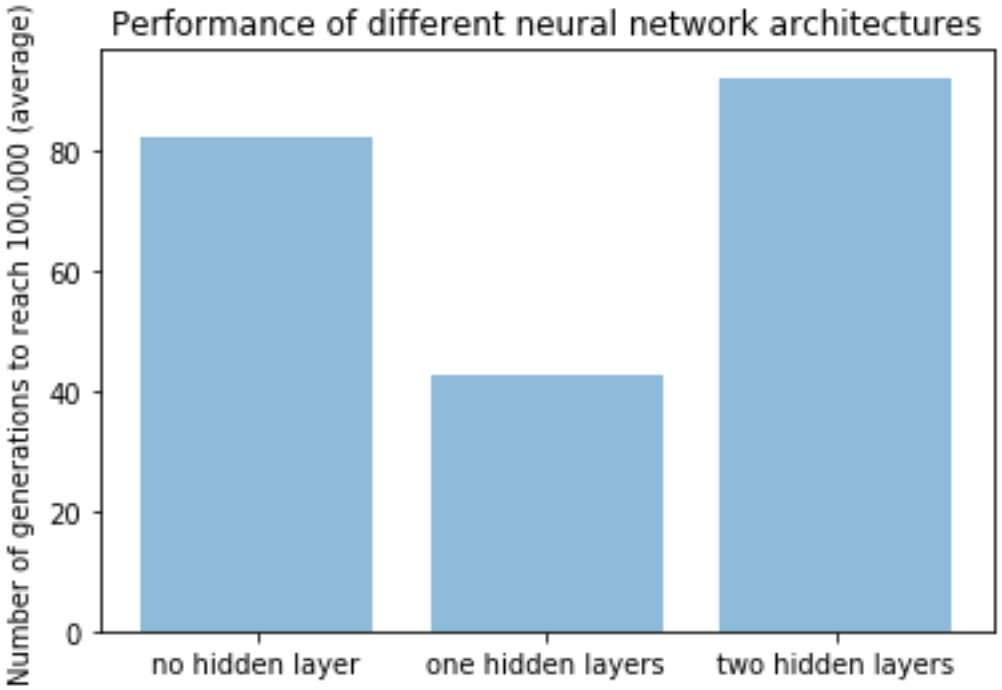
- 20% elitism,
- 20% of new random birds,
- 60% of breeding and additional random mutations.

The importance of randomness

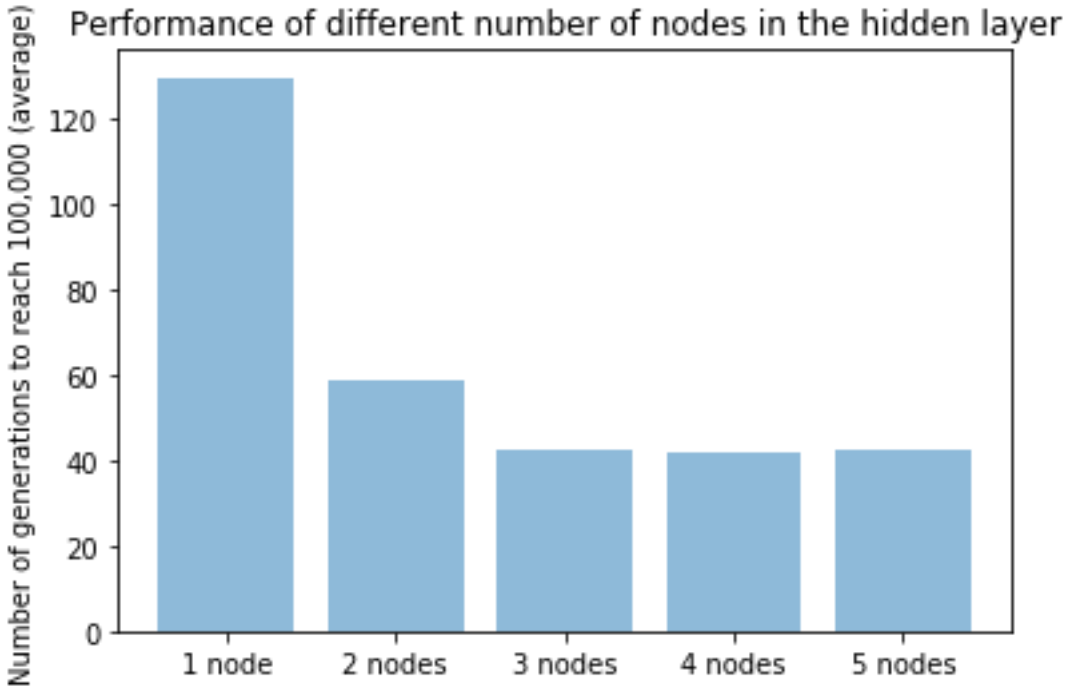




Neural network structure



(all hidden layers contain 3 nodes)



(1 hidden layer)

IV. Comparison of the two algorithms

Neuroevolution requires much lower storage and memory than Q-learning, which has to store big matrices.

Neuroevolution is much faster.

Some things we are proud of

Implemented the game in JavaScript from scratch.

Implemented both algorithms without the aid of any external library.

Q-Learning was built with only the theory in mind and without looking at any other code sources.

References

Neuroevolution:

1. The concept of Neuroevolution: Scholarpedia.
2. Inspiration for the structure of the algorithm (some code come from this repository): [xviniette's repository]
3. Inspiration, similar project: [kishorliv's repository]

Q-Learning:

1. Convergence of Q-learning: [Watkins and Dayan in 1992]
2. Q-learning applied to flappy bird: [Moritz Ebeling-Rump, Manfred Kao, Zachary Hervieux-Moore]
3. Inspiration for discretization: [Cihan Ceyhan's repository]
4. Inspiration for restoring game states for training: [kyokin78's repository]