Smart Snake
Solving the game of Snake with AI

Peter Kosorin

CTU–FIT

kosorpet@fit.cvut.cz

May 12, 2022

1 Introduction

Snake is a game in which the player controls a snake with the aim of maximizing their score by eating food which appears on the screen one at a time. The game is over if the snake crashes into a wall or its own body. The goal of this project was to create an autonomous snake agent that plays the game and compare the performance of different control algorithms used to achieve that goal.

2 Algorithms used

The following project implements and compares three solvers. The first is a domain specific solver (the snake "knows" the foods location) implementing the **A* pathfinding algorithm** and two general solvers, one using **reinforcement learning** methods and the other using a **genetic algorithm** - both utilizing neural networks. The following is a brief description of the algorithms used and their implementation.

3 A* search algorithm

A* is a path search algorithm which is often used due to its optimal efficiency.[2] The algorithm aims to find a path to a given goal node having the smallest cost (least distance travelled). It does this by maintaining a tree of paths originating at the start node (in our case the snake's head) and extending those paths one edge at a time until its termination criterion is satisfied (the food is reached). At each iteration, A* needs to determine which of its paths to extend. It does so based on the cost of the path and an estimate of the cost required to extend the path all the way to the goal. Specifically, A* selects the path that minimizes f(n):

$$f(n) = g(n) + h(n)$$

where n is the next node on the path, g(n) is the cost of the path from the start node to n, and h(n) is a heuristic function that estimates the cost of the cheapest path from n to the goal. The heuristic chosen for this implementation was the Manhattan distance between the snake's head and the food as the game is played on a square grid. The path is recalculated at every step of the game. When no viable path is found by A*, the snake chooses a free tile to step on in hopes that a path will open.

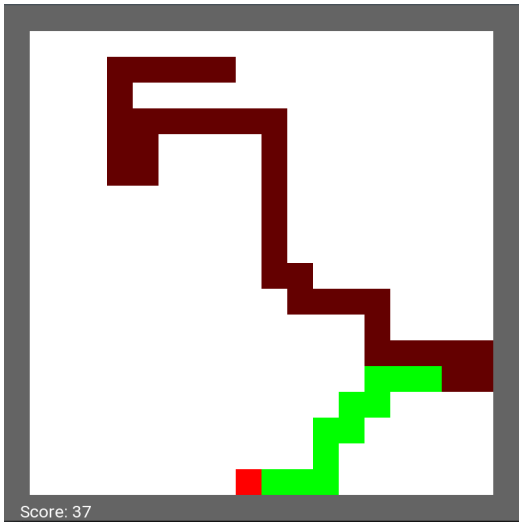


Figure 1: Path (green) found by A* from the snakes head to the food (red)

4 Reinforcement (Deep Q) learning

The concept behind Reinforcement Learning is simple: an agent learns by interacting with its environment. The agent chooses an action, and receives feedback from the environment in the form of states and rewards. This cycle continues until the agent ends in a terminal state, in our case crashing into a wall or itself. Then a new episode of learning starts. The goal of the agent is to maximize rewards during play. In the beginning of the learning phase the agent explores a lot: it tries different (random) actions given the same input state. It needs this information to find the optimal action for a given state. As the learning continues, exploration decreases. Instead, the agent exploits his moves: this

means it chooses the action learned based on its prior experience.

4.1 Game state

The game state is represented as a vector of 11 binary values: whether there is a obstacle left/right/in front of the snake's head relative to movement direction, the direction the snake is moving (left, right, up, down) and the position of the food relative to the snake's head (above, below, left, right).

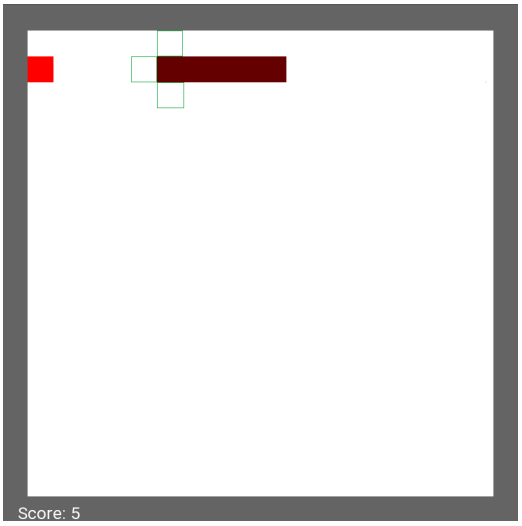


Figure 2: Example game state

Figure 2 shows an example game state, the vector representation would be [0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0] - the tiles around the snake's head have no obstacles (highlighted green), the snake is moving left and the food is straight ahead.

4.2 Rewards

Each action executed by the agent receives a reward: when the snake reaches the food, the reward is +10 and when the snake dies, the reward is -10. When the snake does neither, the reward is 0.

4.3 The model

The model controlling the snake is a dense feed forward neural network with 3 layers: an input layer of 11 neurons, a hidden layer of 256 neurons, and an output layer containing 3 neurons. The model receives the game state vector as an input and outputs an action - the direction the snake should turn. The model is visualized in Figure 3.

4.4 Training the model

The training loop consists of the following steps:

- Get the current state of the environment.

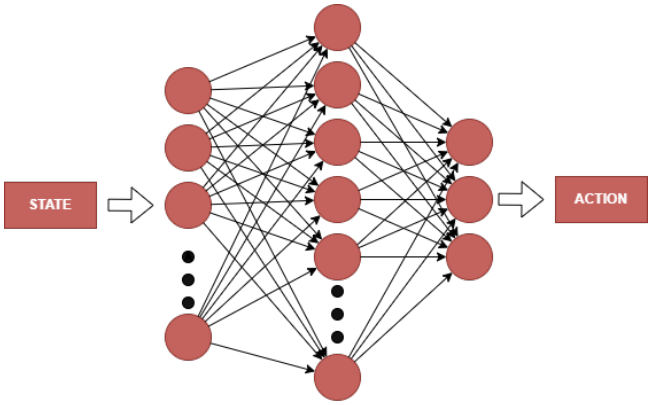
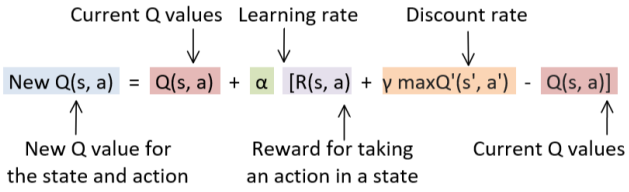


Figure 3: Model visualization

- Get the next move from the model, or a random move.
- Play the step.
- Remember the original state, the move, the state reached after performing the move, the reward and whether the game is over or not (to be used for training later).
- Train the model based on the move performed and the reward obtained.
- If the game ends, train the model based on all the moves executed previously.

During the first phase of the training, the system often chooses random actions to maximize exploration. Later on, the system relies more and more on its neural network.
When the agent chooses and performs the action, the environment gives a reward. Then, the agent reaches the new state and it updates its Q-value according to the Bellman equation:



The actual training consists of calculating the loss function (mean squared error) between the previous Q value and new Q value:

$$loss = (Q_{new} - Q)^2$$

and backpropagating that loss to update the models parameters using the Adam optimizer.

5 Genetic algorithm

A genetic algorithm is a metaheuristic inspired by the process of natural selection. The process of

natural selection starts with the selection of the fittest individuals from a population. They produce offspring which inherit the characteristics of the parents and are added to the next generation. The algorithm consists of 5 stages[3]:

1. Creating the initial population
2. Calculating the fitness of each individual
3. Selection
4. Crossover
5. Mutation

Steps 2 to 5 are repeated until a suitable individual is found.

5.1 The individual

An individual is characterized by a set of parameters known as genes. Genes are joined to form a Chromosome. In our case the genes of a individual are the parameters (weights and biases) of the neural net model controlling the snake. The model architecture is the exact same as in the reinforcement learning model: 3 layers - 11, 256, 3 neurons each. The parameters are treated as a single vector of values.
As seen in Figure 4, the population size chosen was 36 as larger populations become hard to visualize.

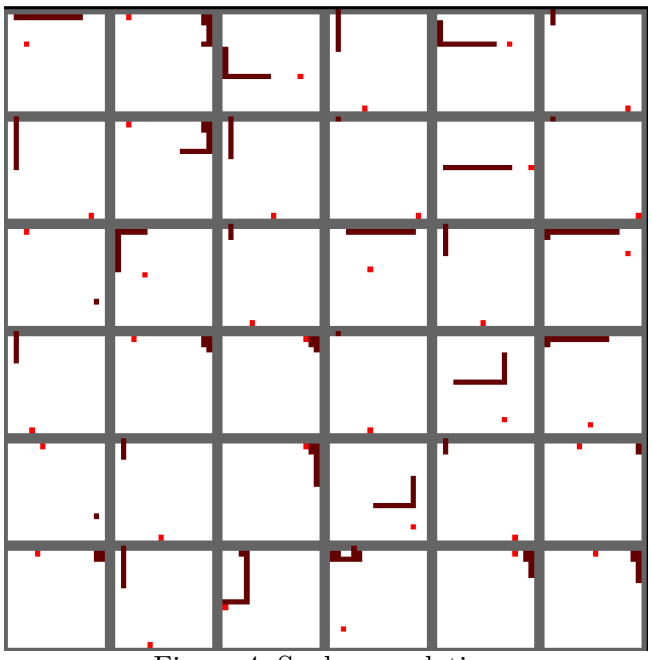


Figure 4: Snake population

5.2 Fitness

At the end of every generation, the fitness of each individual is calculated. The fitness function used:

$$2^{snake\_length} + 4$$

encourages snake growth, especially in the early generations.

5.3 Selection

The selection phase of the algorithm consists of selecting the best individuals from the previous generation to continue to the next one. The method chosen for this implementation is the Tournament selection method. Tournament selection involves running several "tournaments" among a few individuals chosen at random from the population. The winner of each tournament (the one with the best fitness) is selected.

5.4 Crossover

Crossover is the most significant phase in a genetic algorithm. For each pair of parents to be mated, an exchange of DNA is performed, creating offspring. The crossover method chosen was uniform crossover: each gene is chosen from either parent with equal probability of 50%.

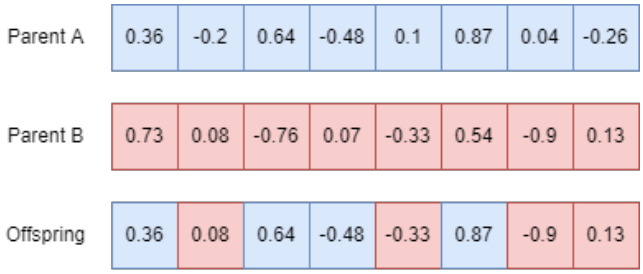


Figure 5: Uniform crossover

5.5 Mutation

Some of the newly formed offspring can be subject to mutation. Each of their genes has a chance of being mutated: a random value between -0.35 and 0.35 is added. Mutation occurs to maintain genetic diversity within the population and prevent the population being stuck in a local maximum.

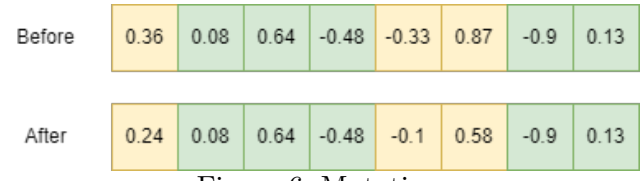


Figure 6: Mutation

6 Results

6.1 Human player

As a control, a human player played 20 games of snake using the keyboard. The highest score achieved was 25, with and average score of 8.2.

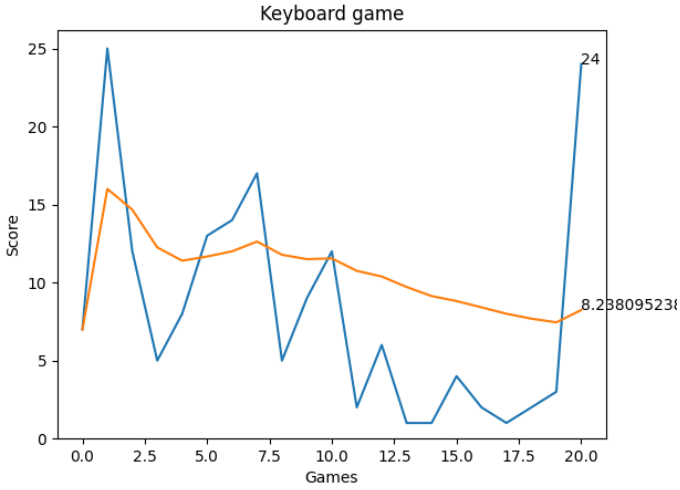


Figure 7: Human player results

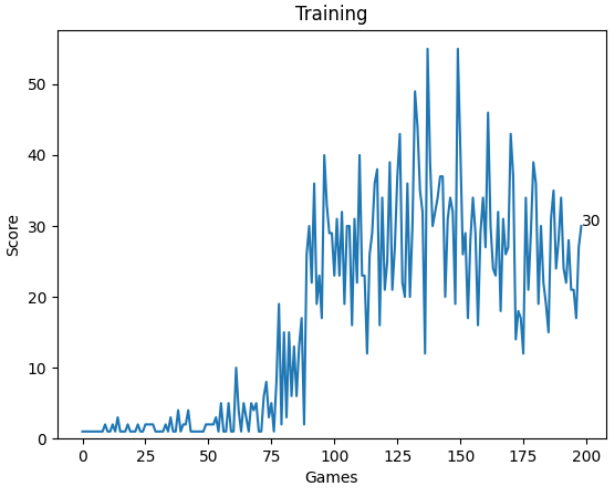


Figure 9: Reinforcement learning performance

6.2 A*

The A* pathfinding algorithm easily outperformed the human player over the span of 20 games, reaching a highscore of 82 with an impressive 63.8 average. The only limitation of A* is its lack of foresight: the only way the snake dies is if it is encircled by its own body.

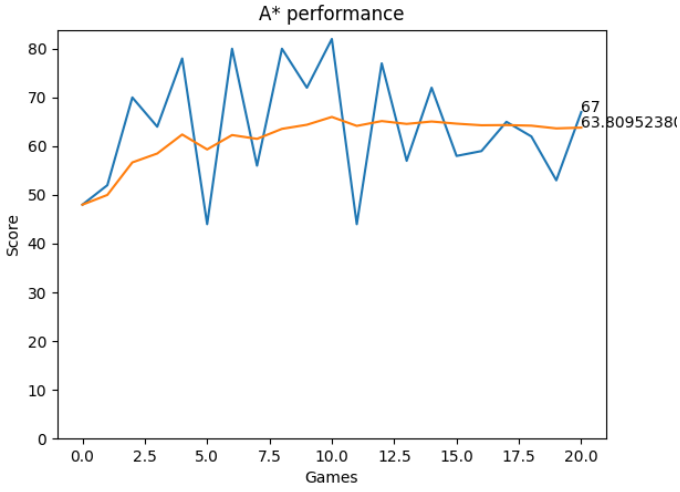


Figure 8: A* preformance

6.3 Reinforcement learning

The reinforcement learning model starts slow, making a lot of random moves at the early stages of training. After around game 100 as the exploration decreases, the model starts consistently hitting scores above 30, with a highscore of 56. The limitation of this model lies in its limited awareness of its surroundings - the snake only "sees" one tile around its head. This could be improved by providing the model with more information in the game state vector, for example the distances to food and obstacles.

6.4 Genetic algorithm

The performance of the genetic algorithm varies widely based on chance. The population size of 36 is quite small - the initial genetic diversity can be lacking. It oftentimes takes many generations to see any significant progress but over time, the results are impressive reaching scores as high as 80. On the other hand, when a little bit of luck is involved, the genetic algorithm produces a viable individual in less than 50 generations.

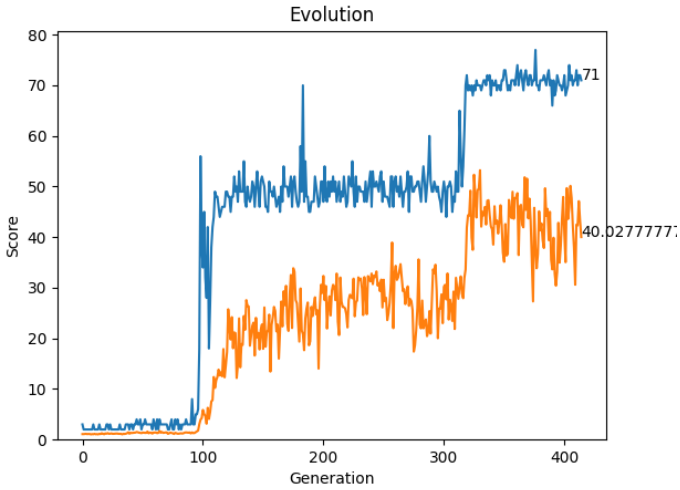


Figure 10: Genetic algorithm performance.
blue - highest score in generation,
orange - average score in generation

7 Conclusion

All of the compared algorithms consistently outperformed a human player. The downside of the reinforcement learning and genetic models is training time - running the genetic algorithm for 400 generations took over an hour. The performance

of the neural net models could be improved by changing the hyperparameters or by extending the state vector the game is represented by. Overall, the performance of these algorithms exceeded my expectations.

References

[1] P. Loeber. Teach ai to play snake - reinforcement learning tutorial with pytorch and pygame. online, 2020. https://www.youtube.com/watch?v=PJl4iabBEz0.

[2] S. Russell and P. Norvig. Artificial intelligence: A modern approach. Pearson, 2018. Boston.

[3] D. Shiffman. The nature of code. Magic Book Project, 2012. Chapter 9. The Evolution of Code.