COMP4106 Final Project Report
Student Name: Yunkai Wang
Student Number: 100968473
Project name: Solving snake game with evolution

Description of the problem domain and the problem

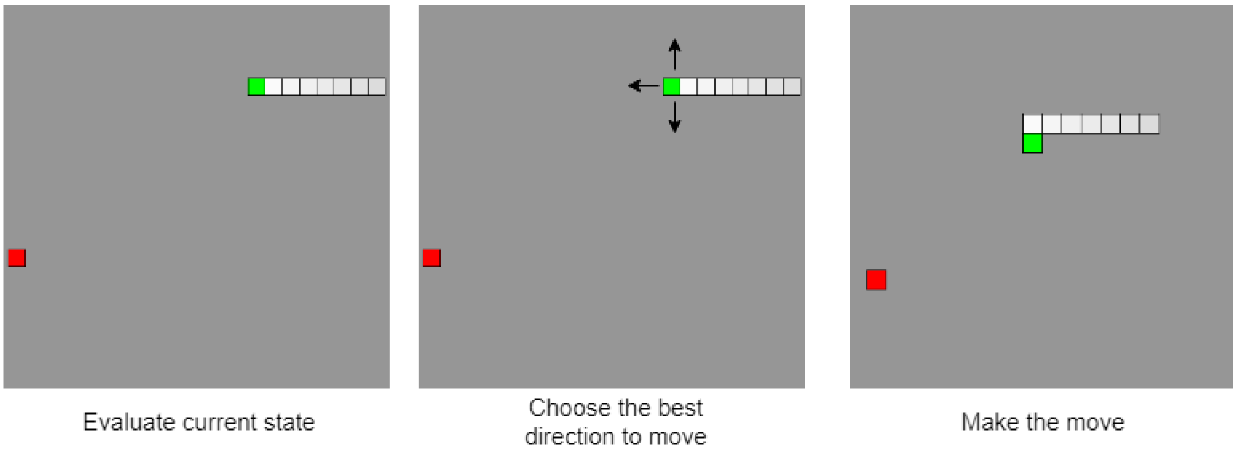
The snake game:

The snake game is a simple board game with the following rules:

1. The world is made up of multiple grids and the snake can only move orthogonally along the grids, it cannot move diagonally.
2. The snake will die if it hit the wall or its own body.
3. In each iteration of the game, there is a food in the world and the snake must try to eat the food in order to grow longer and get a better score.

Solving snake game with evolution

The goal of this project is to create an AI that learns to play the snake game. The AI should analyze the state of the world that the snake is currently living in, then the AI will choose the best direction to move in order to get a better score. The AI should gradually learn the best strategy to play this game without any given instruction.



The basic loop of the game

The motivation for the problem

Artificial intelligence has a growing importance in today's society, and it's not a novel thing that the computer can play a game. In 1997, Deep Blue, which was developed by IBM beat the best chess player in the world. However, the chess-playing computer was programmed by several good chess players, and the application itself will not evolve within a time frame. But the question is can we let the computer 'learns' to play a game without any previously defined game strategies or explicit algorithms? The answer is yes. This was proved to be doable, even on the most complicated board game Go, by the Google DeepMind company as they invented AlphaGo Zero, a version that learns to play Go game without any human data, but it was able to beat any previous human-champion-defeating version.

Solving the snake game using AI techniques has been studied for a while, and there are a lot of known algorithms and techniques that can be applied to solve the snake game like A * search. However, these algorithms explicit instruct the snake what to do, so the snake knows 'how' to play the snake game. People have also tried to use the neural network to solve the snake game, where a lot of training data are generated to train the neural network, and the snake became relatively 'smart' in the end. But what I want is let the snakes 'learn' by itself. Initially, the snake is 'silly', and it should know nothing about the game, but as time proceeds, it gradually learns how to play the game, and become an 'expert' of the snake game. Therefore, for this project, I will be focusing on the method: *genetic random mutation of a neural network.*

The AI techniques/algorithms used

For this project, I will be using two important AI techniques: **Genetic Algorithm** and **Neural Network**.

Genetic algorithm

A genetic algorithm is a search heuristic that is inspired by the theory of natural evolution. This algorithm reflects the process of natural selection, where for each generation, the fittest individuals are selected for reproduction in order to produce the next generation. A genetic algorithm is made up of 5 phases:

1. Initial population: A population consists of a set of individuals, where each individual represents a solution to the given problem. Each individual is made up with a set of parameters known as Genes, the genes are joined into a single string to form a Chromosome.
2. Fitness function: The fitness function determines how fit an individual is, it calculates a fitness score for every individual, and the fitness score represents the probability that the individual will be selected for reproduction.
3. Selection: When producing every new individual in the new generation, two individuals from the old generation are selected based on their fitness score. Therefore, the selection process needs to make sure that the fittest individuals are selected and let them pass their genes to the next generation.
4. Crossover: Crossover is the most significant phase in the genetic algorithm. For every pair of parents to be mated, a crossover point is chosen at random, then the child (or new individual) is created by exchanging the genes of parents among themselves until the crossover point is reached.
5. Mutation: During reproduction, some of the new individual's genes can be subjected to a mutation with a low probability. The mutation is also important since it occurs to maintain diversity within the new population and prevent premature convergence.

Neural network

Artificial neural networks (ANNs) are computing systems inspired by the biological neural networks that constitute animal brains. It can be used to determine the abstract relationship between input data and their output. By training the network with training data, the network can identify the aspect of the input data with a high accuracy by adjusting parameters in the network using learning algorithm. A neural network consists of these important terms:

1. Neuron: A neuron receives an input from predecessor neurons, and produces a single output based on the input and the threshold value that the neuron is currently holding, the threshold value is usually fixed unless changed by the learning function. A neuron with no predecessor is called an input neuron, it serves as the input interface of the whole network. Similarly, an output neuron has no successor and it serves as the output interface of the whole network.
2. Connections and weights: The network contains connections, each connection transfers the output of a neuron I to the input of a neuron j, where I is the predecessor of j, and every connection is assigned with a weight W_{ij}.
3. Propagation function: The propagation function computes the input P_j to the neuron j from output O_i of the predecessor neurons of j typically has the form:

$$P_j = \sum_i O_i * W_{ij}$$

4. Learning rule: Learning rule is usually an algorithm which modifies the parameters in the neural network, such that the neural network produced the desired output based on the given input. The learning algorithm usually modifies the weights and threshold values that the neurons are holding within the network.

The design choices made

Since I am evolving the neural network using the genetic algorithm, so I don't need to find a pool of training data to train the neural network as the genetic algorithm is responsible for choosing the best snakes of the current population and use those snakes to create the next generation. But we still need to design the neural network and genetic algorithm for the system to work. Therefore, let's start by designing the genetic algorithm.

Genetic algorithm

1. **Initial population**: For the first population, all the parameters in the neural network are randomly chosen between [-1, 1], and 100 snakes with random parameters will be created to serve as the first generation.
2. **Fitness function**: The simplest fitness function can be the same as the original snake game's scoring mechanism, where every food the snake has eaten adds 1 to the snake's score. However, as the first population is all generated at random, so most of the snakes will be 'silly', and there is a low probability that they will know they need to eat the food. Therefore, most of the snakes in the first generation will have a score of 0, so we have no way of knowing which snake is better than the others. This can lead to a problem where the bad snakes are selected for reproduction. To solve this problem, I designed the following scoring scheme which can eliminate this problem:
    1. +1 if the snake has made a 'good' move
    2. -2 if the snake has made a 'bad' move
    3. +10 if the snake has eaten a food

A ‘good’ move means that the snake is moving towards the food, so the distance between the head and food is less than the previous distance, and a ‘bad’ move means the opposite, the snake is moving away from the food. The separation is needed since sometimes the snake will be trapped in a circle, so the snake never dies, and its score will become infinity if we simply reward 1 score for every move. However, it’s clear that the snake is unhealthy since it’s not eating the food. So, with the separation, such snake will have a negative score over time, and any snake who has a negative score of -50 will be considered as ‘died’ so the game will not run forever. The fitness function will simply return the score that the snake has scored in one game.

3. **Selection**: Once all the snakes in current generation have died (either hit the wall/tail or scored -50), then the 10 snakes with the highest scores will be added to the mating pool which will be used to create the next generation.
4. **Crossover**: To create a new snake child, two parents are randomly chosen from the mating pool, and a random crossover point is chosen, where all parameters before the crossover point are inherited from the first parent, and all other parameters are inherited from the second parent.
5. **Mutation**: Every parameter in the newly created snake child will be mutated using a global mutation rate, a parameter is mutated by adding a number which has gaussian distribution with a mean of 0 and a variance of 1 divided by 5, but it cannot exceed the limit -1 and 1.

Neural Network

1. The first important question regarding the neural network is what should be the input data? We can provide the neural network with all the information, which is the color of every pixel in the grid, whether it’s a food, snake’s own body or it’s empty. This approach is possible, but it’s much more complicated than it should be, as more input neurons leads to a larger number of connections, and the snake will need to learn everything, what is the food, how to find it, etc. Since the neural network is very complicated, random parameters seems insufficient for this problem, so it will take much longer to converge to ‘smart’ snake. Therefore, we need to simplify the neural network. Imagine that we are the snake and we are trying to keep alive and eat more food, then the only thing that we must know is: *What direction can I move without dying* and *What direction is the food*. The answer to the first question can help us to live longer, and the second question can help us to eat more food, therefore, only provide input for these two questions is sufficient for the snake to fulfill our defined goal. The input to the neural networks will be:
   1) Is it clear to the left?
   2) Is it clear straight ahead?
   3) Is it clear to the right?
   4) Is food to the left?
   5) Is food straight ahead?
   6) Is food to the right?

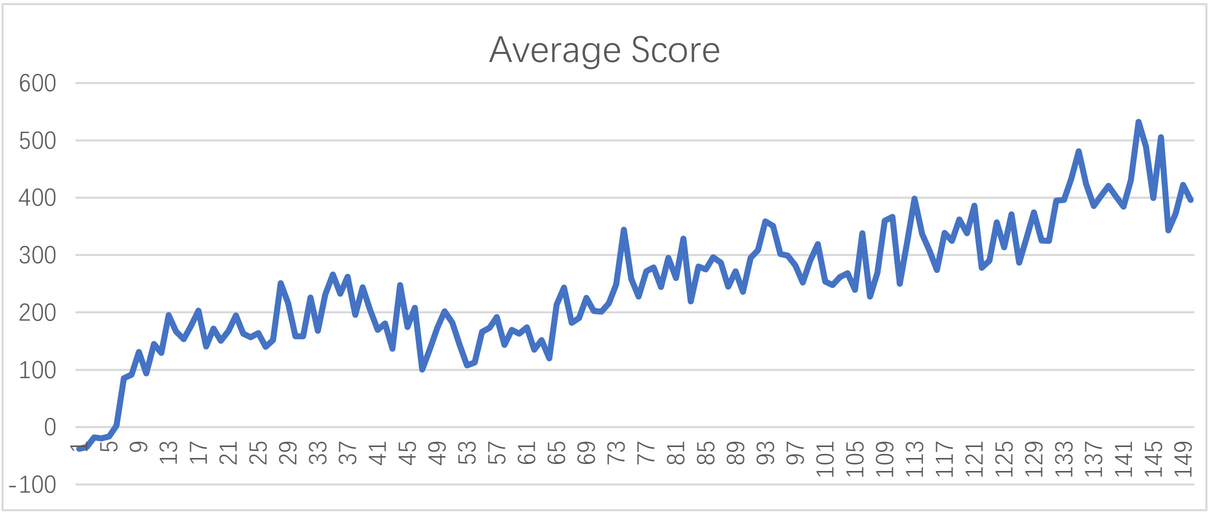
   Every question will have an answer of 1 (yes) or a 0 (no). The system decides whether the directions are clean by looking at the cell that next to the snake’s head, if the snake will die after moving to that cell, then that direction will be considered as ‘bad’. Note that the system doesn’t tell the snake why that direction is bad, it only tells the snake that it cannot move in that direction. The system decides whether the food is to the left or to the right by calculating the degree between the vector formed by the head and the food, and the direction that the snake is moving, if the degree is in the range (-45, 45), then the food will be considered as straight ahead. If the degree is in the range [-45, -180], then the food will be considered as on the left, otherwise, it's considered as on the right of the snake.

2. The next question is what should be the output neuron? The snake can move in three directions: turn left, keep straight, or turn right. So, it's reasonable to create an output neuron that represents each direction. Therefore, in every iteration of the game, we will evaluate the state of the game, create the inputs based on previously defined rules, and feed the input to the neural network, then the three output neurons will have a different weight, and the neuron with the heaviest weight will be chosen as the output, and the direction that corresponds to that neuron will be the next direction that the snake will move.
3. For this project, I don't need to know what parameters for the neural network are better, since all the connections and weights in the neural network should be learned by the genetic algorithms, so all I need is give the neural network with enough neurons such that it will be able to learn. For this project, I used a neural network with 3 layers, the first input layer contains 6 input neurons, the hidden layer contains 6 neurons, and the output layer contains 3 neurons.
4. The learning algorithm is the genetic algorithm that I have already defined above, and I don't need an extra learning algorithm for this neural network.

The results of applying the above-mentioned techniques to the problem

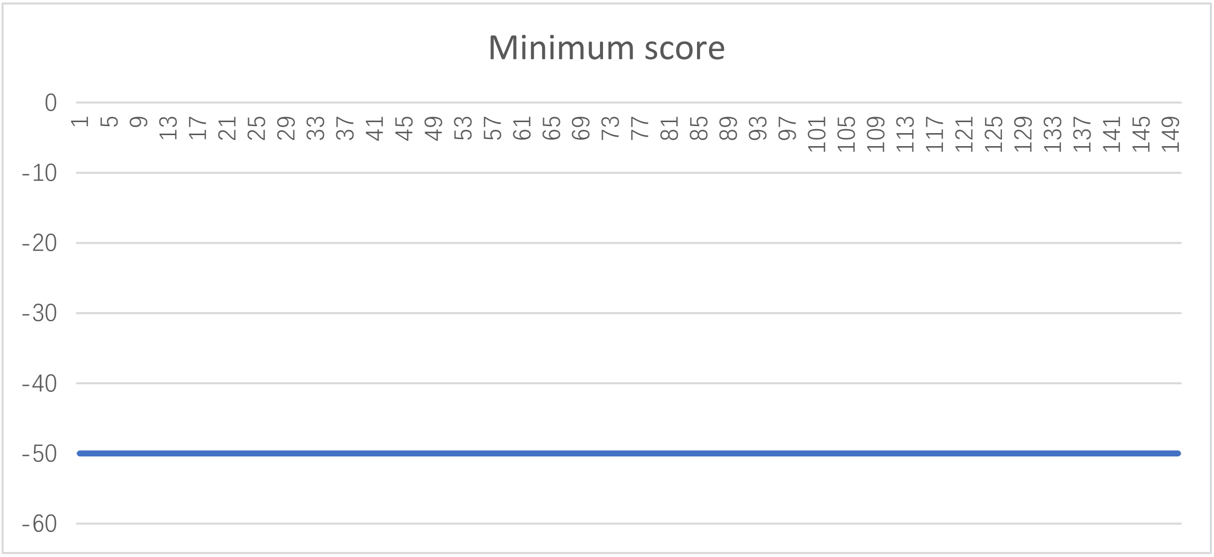
Average score

The following chart shows the average score of all the snakes in the population from generation #1 to generation #150. The average score increases rapidly in the first 10 generations, then the increase rate becomes smaller and smaller, and as time proceeds, it will converge to an average score of 400. The reason of this convergence will be discussed in the problem section. But overall, most of the snakes are getting smarter.

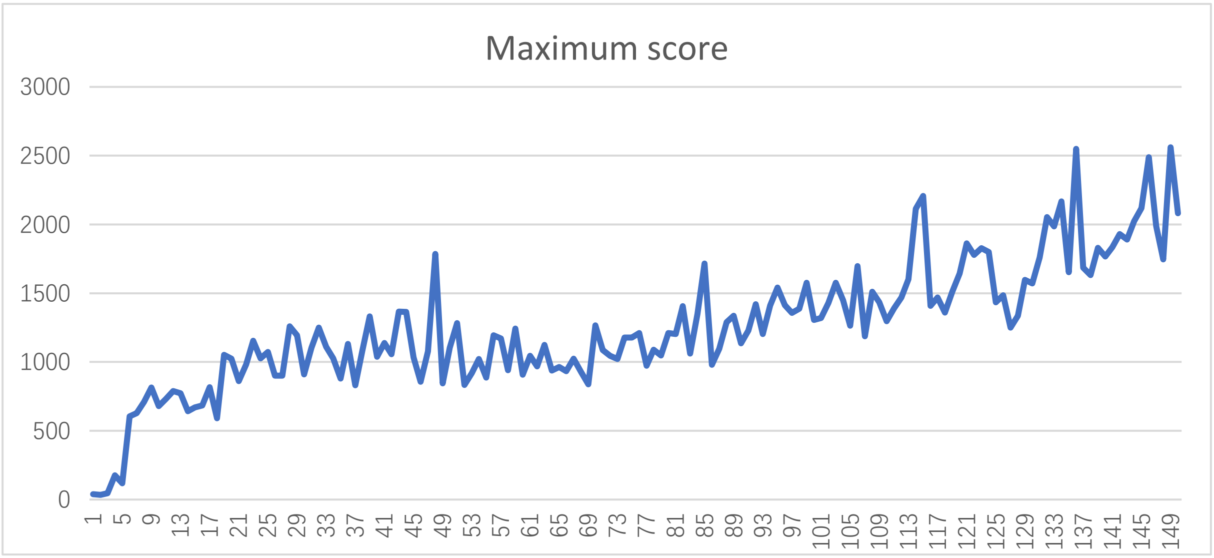


Average score of the entire population for the first 150 generation

The next two charts show the score of the worst and the best snake in the entire population, and it will help us to understand the result better. The first chart shows that the worst snake in all populations is still 'silly', it still tries to run away from the food. This result is expected because the genetic algorithm only which snake is overall better than other snakes, but the genetic algorithm doesn't know which parameter of the snake is 'good', so the child may inherit bad behaviors from its parents. The second chart has a similar growing rate as the chart of average score. The best snake is growing better over generations, but the score of the best snake may decrease in some generations, the reason is that the snakes are not good at escaping from their own tails.



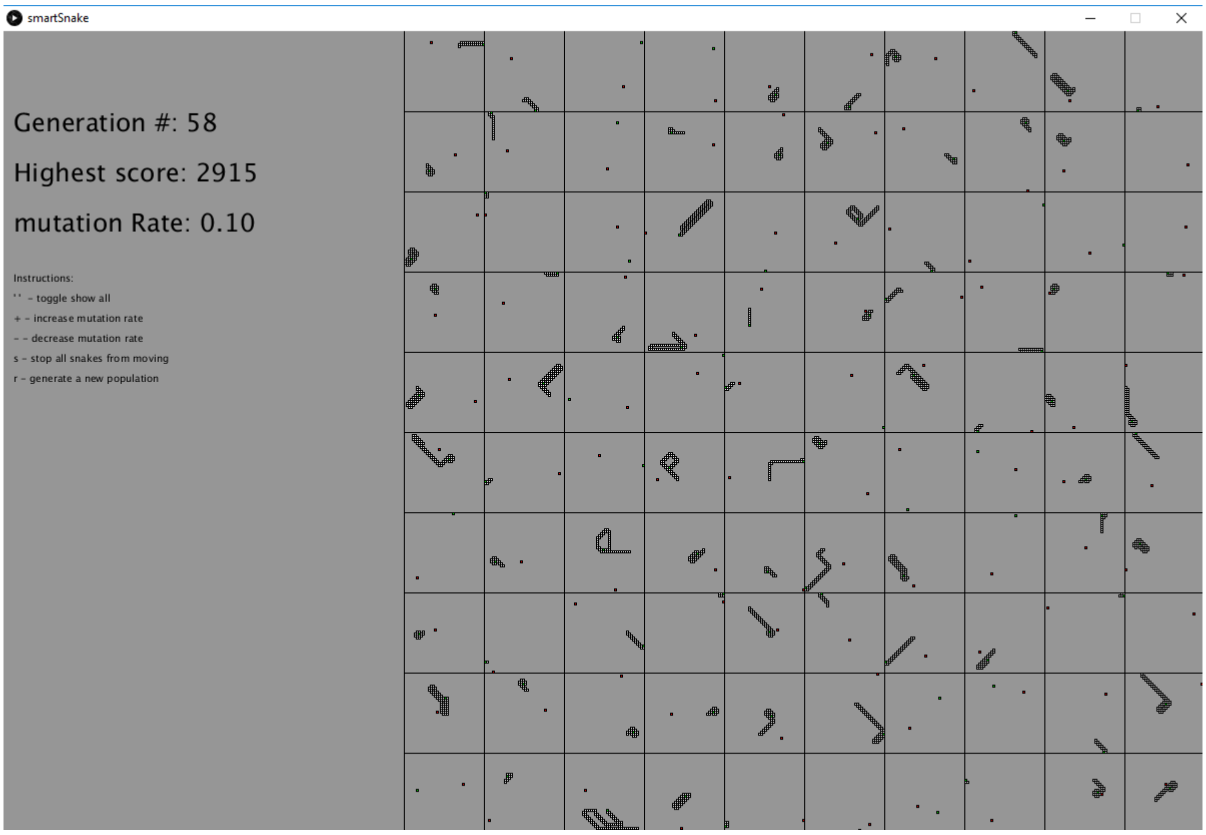
Minimum score of the entire population for first 150 generations



Maximum score of the entire population for first 150 generations

Observation

Below is a snapshot that I have taken after the program has been running for a while. Each cell represents a single snake in the population, it's clear that the snakes are getting smarter as we have more long snakes. The snakes in the first several generations only know that they need to eat the food, they discard the information about whether the directions are clear or not, so they will hit their own body or the wall. But after a certain number of generations, the snakes will start to realize that they will die if they move to one of the bad directions, so they will try to pick directions that are clear and also brings them closer to the food. It's interesting to notice that the snakes prefer to move diagonally (up->left->up->left, etc.) since these snakes don't know the exact position of the food, they only try to follow the approximate direction as given by the input.



Generation #: 58

Highest score: 2915

mutation Rate: 0.10

Instructions:

' ' – toggle show all

+ – increase mutation rate

- – decrease mutation rate

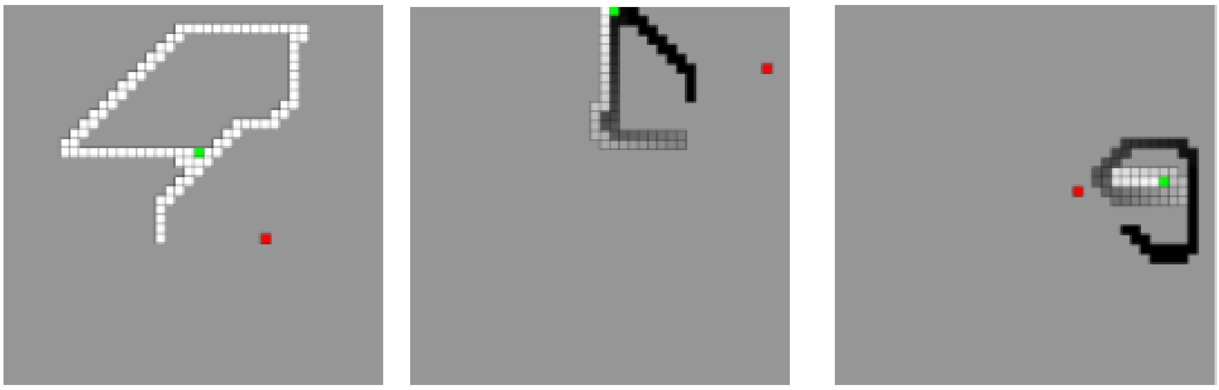
s – stop all snakes from moving

r – generate a new population

Screenshot after the system has been running for a while

Problems that occurs

Even though the snakes get smarter, but the average snakes' score will converge to a certain value (~400). The reason is that the snake only knows the approximate direction of the food and whether the cells next to its head is clear or not (inputs to the neural network). Even though the snake is trying to eat the food and escape from those 'bad' directions, but the snake doesn't know whether there is an obstacle on its chosen path to the food. If there is an obstacle, then the snake is very unlikely to escape from the obstacle as it only sees it when it's next to the obstacle. Therefore, after the snake has grown longer, it becomes harder and harder for the snake to escape from hitting its own body and the wall. As it can be shown in the screenshots below, the snake always takes the shortest path to the food (the green cell represents the head of the snake and the red cell is the food), but there is an obstacle (its own body) on the path, so the snake dies by hitting its own body. This is normal since the input to the neural network doesn't provide enough information for the snakes to predict the best path to eat the food. Therefore, the average score will always converge.



Screenshots of snakes that hit their own body when trying to eat food

The next problem that I think of is the new scoring system is still not perfect. The reason is that clearly in these examples provided, the snakes need to make several moves away from the food so that it can keep alive and eat the food. But the scoring system will penalize the snake for doing that, then the snake will have a lower probability to be selected for producing the next generation, so we have missed some good snakes. Therefore, the scoring system should separate snakes that are running away from the food with the snakes that are running away from the food to escape from the obstacles. But it will increase the algorithm's complexity as it's hard to identify these two types of snakes.

The possible enhancements to the study

To solve the problem that I just mentioned above, I will list some possible solutions that may solve the problem:

1. Provide the neural network with all the information required for the snake to become 'ultimate' snake, like the state of all the cells on the grid, so that the snake will be able to find the best path to eat the food. In this case, the snake will be able to escape from its own tails. However, as the complexity of neural network increase dramatically, the algorithms will run much and much longer before it can converge to a 'smart' snake. However, this seems to be the only direction to evolve an 'ultimate' snake.
2. Instead of providing the neural network with all the information, only provide several extra information. For instance, right now the snake only 'sees' the cells next to its head, we can update the information provided to the distance to the next obstacle in that direction. Then we also provide the distance to the food. Therefore, the snake will be able to know it may hit the wall if it simply tries to move closer to the food. However, this solution still cannot solve the problem since even though the snake knows how many moves can make in each direction, it doesn't know how many moves it needs to move before the obstacle disappears. Therefore, this won't make a big improvement in this project.
3. Every time the snakes dies, make a copy of current state, and when a new generation is created, use all the memorized games to train the snakes' brains so that they may identify ways they can do better. This is similar to training the neural network using training data, but the difference is that the snake creates its own training data, the programmer is still not responsible for collecting training data. However, this will increase the complexity of this project, and it's not originally designed as part of this project, so it will not be applied.

References

Micheal A. Nielsen, “Neural Networks and Deep Learning”, Determination Press, 2015

Mallawaarachichi, V. 2017. Introduction to Genetic Algorithms – Including Example Code. [https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3]. Accessed April 16, 2018.

Korolev, S. 2017. Neural Network to play a snake game. [https://towardsdatascience.com/today-im-going-to-talk-about-a-small-practical-example-of-using-neural-networks-training-one-to-6b2cbd6efdb3]. Accessed April 16, 2018.

Binggeser, P. 2017. Designing AI: Solving Snake with Evolution. [https://becominghuman.ai/designing-ai-solving-snake-with-evolution-f3dd6a9da867]. Accesses Aril 16, 2018

Appendix

To run the code, you need to have Processing installed on your computer. You can download it from https://processing.org/. After installation is complete, then you can simply open the application and click “Run” button at the top.