Snake Game AI: Movement Rating Functions and Evolutionary Algorithm-based Optimization

Jia-Fong Yeh, Pei-Hsiu Su, Shi-Heng Huang, and Tsung-Che Chiang
Department of Computer Science and Information Engineering, National Taiwan Normal University, Taiwan
{40247006S, 40247031S, 40247032S}@ntnu.edu.tw, tcchiang@ieee.org

***Abstract*—Snake game is a computer action game, whose goal is to control a snake to move and collect food in a map. In this paper we develop a controller based on movement rating functions considering smoothness, space, and food. Scores given by these functions are aggregated by linear weighted sum, and the snake takes the action that leads to the highest score. To find a set of good weight values, we apply an evolutionary algorithm. We examine several algorithm variants of different crossover and environmental selection operators. Experimental results show that our design method is able to generate smart controllers.**

Keywords—game; artificial intelligence (AI); snake; evolutionary algorithm

I. Introduction

Snake game is a classic computer action game, in which we control a snake to move and collect food in a map. In this paper we are interested in developing a controller with artificial intelligence (AI) using movement rating functions and evolutionary algorithms (EA). Before we elaborate how we design our controller, we first describe the snake game we implemented and define the objective.

The game environment is a 30×30 map. The snake starts with length one. At each time step the snake moves one step, and it can go straight, turn left, or turn right. There are two kinds of food in our game, apples and bananas. Eating an apple increases the length of the snake by one, whereas eating a banana decreases the length by two. There is always one apple and at most one banana in the map. One banana appears after every 30 apples are eaten. We can play the original snake game with a simple strategy: we move the snake up and down from the right to the left without touching the last row (the row at the bottom); when the snake is close to the leftmost boundary, go back to the rightmost boundary along the last row. In this way, the snake keeps alive until it occupies the whole environment.

To make our snake game more challenging and interesting, we set the initial score of an apple by 50 and decrease it to 25 as time passes. Besides, eating a banana decreases the score by 75. The game ends when any of the following four conditions is satisfied.

(1) The snake hits its own body or the wall.

(2) The length of the snake gets zero.

(3) The game score gets negative.

(4) The game score does not change after 900 time units.

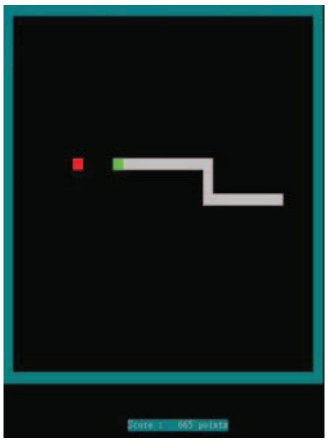


Fig. 1. A screenshot of the Snake game

The objective of our game is to maximize the score. The above-mentioned simple strategy may keep the snake alive, but without moving toward the apples efficiently it cannot get a high score. Thus, we need to design a more intelligent controller, which is the topic of this paper. The rest of this paper is organized as follows. Section II reviews related work about computer game design using EAs. The main component of our AI controller is a set of rating functions, which will be described in Section III. We aggregate these functions through weighted sum, and Section IV presents how we set weights through an EA. Section V gives the experiments and results. Conclusions are made in Section VI.

II. Related Work

EAs have shown many successful applications to design AI or generate contents for computer games. A useful paradigm is to use EAs to optimize the parameters of the controller/game. Gallager and Ryan [1] proposed a rule-based controller for playing the Ms. Pacman game. It is a game about moving a character (Ms. Pacman) to collect pills and to escape from ghosts in a maze. The controller first determines its state, *explore* or *retreat*, based on the distance between Pacman and the ghost. Then, it determines the new direction of movement based on the type of its location (e.g. corridor or intersection) probabilistically. The distance threshold and the probabilities form a set of 85 parameters of the controller. An evolution strategy (ES) was applied to optimize values of these parameters.

Cole et al. [2] designed an AI controller for Counter Strike, a popular first-person shooting (FPS) game. In this game human and computer players equip themselves with different types of weapons (e.g. AK-47 or sniper rifle) to complete specified missions (e.g. defuse bombs). Behaviors of controllers depend on the preference of weapons and the aggressivity (how aggressive the player is). Values of these parameters were optimized by a genetic algorithm (GA). Their experiments and results showed that the GA-assisted players perform similarly to human-tuned players.

Böhm et al. [3] solved the Tetris game by rating functions and EA-based optimization. Tetris is a block falling game, in which players need to put a series of blocks of different shapes properly to get high scores. Böhm et al. evaluated different positions of placing blocks by many criteria such as pile height and removed lines. These criteria were aggregated, and the best position was determined by the aggregated value. The EA was responsible for optimizing the weight values. The framework of our controller is similar to theirs.

Schichel et al. [4] applied genetic programming (GP) to design the controller for playing Robocode, a game about driving a tank and shooting bullets to destroy other tanks. Their controller consists of four parameters, including moving distance, moving direction, and so on. GP is a kind of EA, featured by representing chromosomes as trees. In their trees, non-leaf nodes (so-called functions) represent arithmetic and logic functions; leaf nodes (so-called terminals) represent game attributes (e.g. tank positions and energy) and numeric constants. They used GP to generate suitable expressions to calculate required parameter values in their controller.

Muñoz et al. [5] aimed to tune 22 parameters (e.g. wing angles, wheel pressure, gearbox ratios, etc.) of racing cars in a car setup optimization competition. They used two EAs to search for the optimal set of parameter values. Both EAs evaluated individuals based on best-lap time, top speed, and raced distance. The second EA also considered car damage. A key difference is that one EA aggregated the concerned objective values into a single value but the other followed the Pareto dominance relationship. Experimental results showed that the dominance-based EA outperformed the aggregation-based one. It was also the best among the five participants in the competition in 2010.

In addition to designing AI controllers to play games, EA can be utilized to generate game contents. Super Mario is a game in which a character (Mario or Luis) moves, runs, and jumps to pass levels composed of blocks, enemies, and many kinds of items (e.g. coins). Different players could have different preferences when playing this game. For example, some players like to walk and collect coins but some others run to pass the level as soon as possible. Cheng et al. [6] intended to generate interesting levels based on users' behaviors. Their level generator generated a level by concatenating five types of zones. The appearing ratios of these zones were controlled by base ratios and users' tendency. The base ratios of zones were optimized by interactive evolutionary optimization, where human users play the role of evaluation. Their generator is the winner of the Level Generation Track of the Mario AI Competition held in IEEE Conference on Computational Intelligence and Games in 2012.

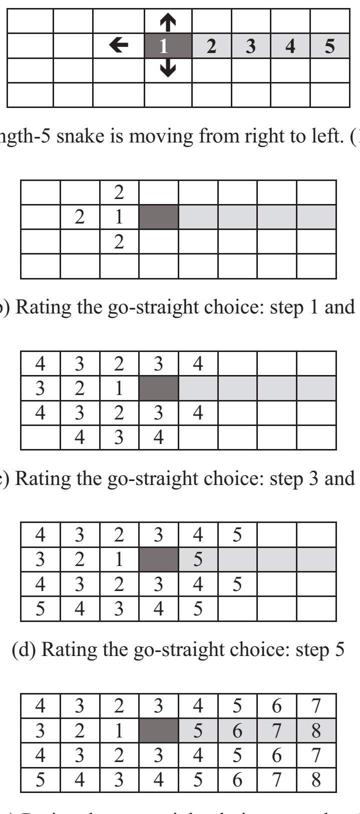
Although EAs have been popularly used in game designs, there was little research about using EAs to develop snake game AI. Ehis [7] used GP and investigated the function set. The terminal set consisted of three actions. The first version of his function set only included five functions such as ifFoodAhead and ifDangerAhead. However, the preliminary tests showed that GP cannot generate good controllers with these functions. Thus, he extended the function set with more functions such as ifDangerTwoAhead and ifMovingUp. The extended GP generated controllers with interesting behaviors. Note that our game in this paper is more challenging due to time-dependent scores and the existence of bananas.

III. Rating Functions and Direction Control

The AI controller is responsible for deciding the moving direction of the snake. Simply speaking, it has to choose one from three possible actions: go straight, turn left, or turn right. Our controller rates each of the three possible next positions of the snake head by four rating functions – smoothness, space, apple, and banana. We will detail all of them in the following sub-sections.

A. Smoothness

The smoothness rating function calculates the shortest distance to all reachable positions and takes the largest value. We use this value to estimate how long the snake can move smoothly without making many turns. (Making turns is difficult and sometimes dangerous for the snake.)



(a) A length-5 snake is moving from right to left. (1: head)

(b) Rating the go-straight choice: step 1 and 2

(c) Rating the go-straight choice: step 3 and 4

(d) Rating the go-straight choice: step 5

(e) Rating the go-straight choice: completed

Fig. 2. Illustration of the smoothness rating function (assuming map size 4×8)

Fig. 2 illustrates the rating process. Assume that a lengh-5 snake is moving horizontally from left to right in Fig. 2(a). It has three choices, and now we rate the go-straight choice in Fig. 2(b). The numbers in Fig. 2(b) mark the reachable positions and represent the distance within two move steps. Keeping the above process (doing a breadth-first search, BFS, on the map), we have the state in Fig. 2(c). Note that we need a check condition to know if we can move to a position where the snake originally occupied. Whenever the snake moves one step, each of the indices of the snake (including its head and body) increases by one. Thus, the condition that the snake can move to where it occupied implies that the body disappears:

$$\textit{BodyIndex} + \textit{MoveSteps} > \textit{SnakeLength}. \tag{1}$$

In Fig. 2(c), the snake head cannot move to the position where the snake head (index 1) originally occupied in Fig. 2(a) since the condition is not satisfied (1 + 4 is not greater than 5). In Fig. 2(d), the snake head can move to the position where snake body with index 2 originally occupied since the condition is satisfied (2 + 5 > 5), leading to a distance of five. Finally, Fig. 2(e) shows that the snake can move 8 steps with only one turn if the decision is to go straight in Fig. 2(a).

B. Space

The smoothness rating function helps to identify the way to live easily. However, sometimes we do need to choose a way to live even though the life is tough. Take Fig. 3 as an example. The smoothness of turning right is seven and leads to a dead end. Although the smoothness of turning left is six (smaller than seven), but there is a chance for the snake to keep alive by making many turns. The space rating function returns the number of positions reachable. The smoothness and space rating functions are complementary and can find the safe way for the snake together.

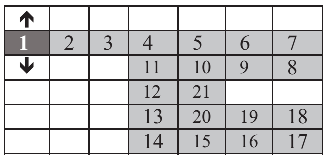


Fig. 3. Illustration of the space rating function (assuming map size 6×7)

C. Food: Apple and Banana

The apple and banana rating functions are identical except that the target is different – one is the apple and the other is the banana. To rate a choice (go straight, turn left, or turn right), we check whether the target is still reachable after taking that choice. If it is not reachable, the distance is infinite and the score is zero; otherwise, we calculate the score by

$$f(\textit{target}) = \textit{space} \,/\, \textit{distance}(\textit{target}). \tag{2}$$

Take Fig. 4 as an example. The star symbol represents an apple. Although turning left gets closer to the apple, the space is small (the snake will die), leading to a low food score 3/1 = 3. By contrast, turning right gets farther but makes it possible to eat the apple and survive. The food score of this choice is 16/3 = 5.33, higher than 3. Thus, the snake will choose to turn right.

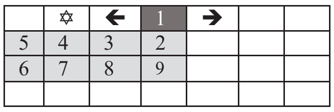


Fig. 4. Illustration of the food rating function (assuming map size 4×7)

D. Direction Control

For each direction choice, we rate it by four functions described in previous three sub-sections. Let $f_i(c)$ denote the rating functions, where $i \in \{1, 2, 3, 4\}$ and c is a choice. Since different rating functions may have different importance, we use weighted sum to aggregate these scores:

$$F(c) = \sum_{i=1}^{4} f_i(c) \cdot \left(w_{2i-1} + w_{2i}\,\frac{l}{900}\right) \tag{3}$$

The choice that maximizes the final score is made.

Note that in (3) we have two weights for each function. One weight w_{2i-1} determines the base effect of function $f_i(c)$. The symbol l denotes the length of the snake. The other weight w_{2i} represents the effect as the length of the snake grows. This helps us to express some complicated principles. For example, we are able to express that one rating function is important when the snake is short (eating apples is good at the beginning) but becomes relatively less important as the snake gets long (keeping alive is more important) by a large w_{2i-1} and a small w_{2i}. Here we divide l by 900, which is the size of the map (30×30) and also the maximal length of the snake. In other words, as the snake grows its length to the maximal value, w_{2i} increases its effect to be equal to w_{2i-1}.

IV. Evolutionary Algorithm-based Optimization

In Section III we explain that our controller chooses the best direction that maximizes $F(c)$ in (3). The next question is how to determine the weight values (i.e. the importance of each rating function). It could be a very tiring and time-consuming job for a human game designer to tune these weight values manually. In this paper, we propose an EA to do this job automatically and intelligently.

A. Chromosome Representation and Initialization

The goal of our EA is to find good values of eight weight parameters in (3). Therefore, the chromosome representation in our EA is a real vector of eight variables. The range of each variable is [-15, 15]. The value of each gene is set by zero, r, or $-r$ in equal probability, where r is a random value in [0, 1].

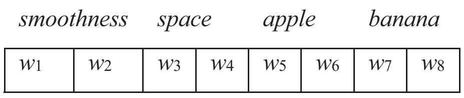


Fig. 5. Chromosome representation

B. Evaluation

Each individual is evaluated by playing the snake game with the proposed direction control procedure (Section III-D) using weight values encoded on the individual. Fig. 6 illustrates the relationship between direction control, EA, and the game. At first, the EA sends a set of weight values to the direction control module. Then, the game interacts with the direction control module by sending game states and receiving actions. Finally, as the game ends, the game sends a score to the EA as the fitness of the individual to be evaluated.

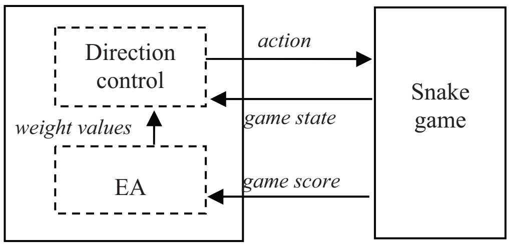


Fig. 6. Chromosome evaluation

C. Reproduction

At each generation, new individuals (offspring) are produced through mating selection, crossover, mutation, and environmental selection. In our EA, we use 2-tournament to select parents. We test two crossover operators: the single-point crossover and a random crossover. The single-point crossover cuts the parents into two sections by a random point and exchanges the first sections between the parents. The random crossover exchanges half random genes between the parents. For mutation, each gene adds a random value in [-1, 1] in probability p_m, where p_m, the mutation rate, is a parameter.

Given a population of size N_p, N_p offspring will be produced in each generation. Then, we have to decide which individuals can survive to the next generation. Here we test two methods: the generational model and the (n+n) method. When the generational model is adopted, all N_p new offspring survive and replace the old population. As for the (n+n) method, it merges the old population and newly generated offspring and removes the half with worse (smaller) fitness values.

V. Experiments and Results

A. Experimental Setting

We carried out four experiments. In the first experiment, we examined four controllers, each of which used only a single rating function. This experiment attempted to justify the need of multiple rating functions. In the second experiment we examined whether different crossover and environmental selection operators have impact on the quality of the evolved controller. Snake-length-related weights (w_{2i} in (3) and Fig. 5, $1 \le i \le 4$) were disabled (not encoded in the chromosome) in this experiment. The third experiment enabled these weights and investigated whether the length-related weights are beneficial. The fourth experiment compared the EA-generated controller with a heuristic controller.

Our EA has three parameters, and we did not tune parameter values particularly to enhance the solution quality. Instead, we set the values in a reasonable range. The population size N_p was 20, the number of generations N_g was 50, and the mutation rate p_m was 0.05. The snake game and the AI controller were implemented by C++. Each run of EA took about 30-60 minutes.

B. Experiment 1: Need of Multiple Rating Functions

This experiment examines the behaviors of four controllers that consider only one rating function. Each controller played the same snake game (same positions of apples and bananas) for one time. Scores of all controllers are given in Table I. The controller considering only *smoothness* kept the snake circling along the boundary of the map, and the snake did not eat any apple. According to the ending conditions mentioned in Section I, the game ended after 900 time units. The controller considering only *space* had a similar behavior. This is due to the fixed order of checking the three action choices (go straight, turn right, and then turn left) in our design. At the beginning of the game, the resulting space of every action is the same and the snake keeps going straight. As it is going to bump into the boundary, it turns right. Then, it moves along the boundary and cycles. When the controller considers only eating *apples*, it got some but died early as its length got long. As for the controller considering only *bananas*, it indeed ate the banana with the first priority and then diminished to end the game. Based on the results, we know that a controller using a single rating function cannot perform well in our snake game.

TABLE I. Scores of Controllers using a Single Rating Function

	smoothness	space	apple	banana
score	0	0	5954	**-75**

C. Experiment 2: Crossover and Environmental Selection

We tested four EA variants with combinations of two crossover and two environmental selection operators. In this experiment the positions of apples in the game were fixed. In Table II, RC and SP denote random crossover and single-point crossover, respectively; N+N and GM denote (n+n) method and generational model, respectively. Here, the snake-length-related weights (w_2, w_4, w_6, and w_8) were not encoded, and thus the length of chromosome was 4. Each EA variant was run for ten times. The maximum, average, and minimum scores over ten runs are reported in Table II. *Succ* denotes the number of runs in which the snake grows to the maximum length 900. *AvgLen* denotes the average length when the game ends. The best performance is marked in bold.

TABLE II. Scores of Controllers using Differnet Crossover and Environmental Selection Operators (Length-related Weights Disabled)

	RC _N+N	RC_GM	SP_N+N	SP_GM
Max	**44721**	42657	43935	36522
Avg	**28080**	20795	22190	19387
Min	**5991**	0	5959	5952
Succ	**1**	0	0	0
AvgLen	**567**	420	448	393

We have some observations from the results. First, the best-case performance of the tested variants is close except that SP_GM is much worse than the other three. On average the RC crossover is better than the SP crossover, and the N+N method is better than the GM method. The RC crossover can exchange consecutive or non-consecutive genes between parents, but the SP crossover can exchange only consecutive genes. The higher variety of produced offspring might be the reason why RC is better than SP. The N+N method realizes the elitism and keeps good individuals in the population. Its better performance than the GM method implies that in this problem good individuals are not easy to find and they should be kept once they are found. Comparing with their counterparts, RC and GM tend to search in a diversified manner. This feature makes the combination of them (RC_GM) lose search direction sometimes, resulting in the zero score. The worst-case performance of the other three variants is almost the same. Actually, the evolved controllers in these cases were the controllers that only considered eating apples, i.e., only w_5 was positive but the remaining three weights were zeros. These solutions form the local optima in the search space.

D. Experiment 3: Need of Length-related Weights

The goal of this experiment is to verify the need of length-related weights, i.e., w_2, w_4, w_6, and w_8 in (3) and Fig. 5. Here, they joined the evolution process, and thus the length of chromosome increased from 4 to 8. Scores of the four EA variants are listed in Table III.

TABLE III. SCORES OF CONTROLLERS USING DIFFERNET CROSSOVER AND ENVIRONMENTAL SELECTION OPERATORS (LENGTH-RELATED WEIGHTS ENABLED)

	RC_N+N	RC_GM	SP_N+N	SP_GM
Max	44803	44781	**44805**	44790
Avg	**44055**	40004	41016	33934
Min	**42744**	19876	23381	13100
Succ	**3**	**3**	2	2
AvgLen	**886**	805	827	683

From Tables II and III, we can clearly see that the introduction of length-related weights significantly improves the performance of the controller. The average score increases by at least 14,547 points, and the worst score increases by 7,148 points. The life time of the snake also gets much longer, and the snake can grow to the maximum length in at least two of the ten runs for all EA variants. The addition of length-related weights doubles the chromosome length and enlarges the search space. However, the experimental results show that many solutions in the enlarged space are good and easy to find.

Until now we evaluated individuals with the same deterministic snake game based on two reasons: (1) we want to evaluate individuals in a fair manner; (2) the time to play a game is long. Next, we took the best controllers obtained in experiment 2 (length 4) and in this experiment (length 8). We let each of them play 100 random snake games (in which apples appeared in random positions) and calculated the average score. On average the controllers with length-related weights get 4000+ score than those without length-related weights. Evidently, the dynamic rating strategy has higher capability of dealing with various conditions in the snake game. Its performance is also more stable.

TABLE IV. AVERAGE SCORES OVER 100 RANDOM GAMES OF CONTROLLERS WITH LENGTH-RELATED WEIGHTS DISABLED AND ENABLED

	DISABLED	ENABLED
RC_N+N-Best	18158	20539
RC+GM-Best	15768	21491
SP_N+N-Best	12152	21557
SP_GM-Best	20768	20670
Average	16712	21064

E. Experiment 4: Evolved Controller vs. Heuristic Controller

In the last experiment we want to compare the evolved controller with a heuristic controller [8]. The heuristic controller was designed based on two ideas. First, the snake is only allowed to move to the positions that there is a path to reach its own tail; if there are more than one position, move to the position closer to the apple. Second, the snake keeps the unoccupied positions connected. To achieve this, the snake moves away from its tail. If turning and going straight lead to the same distance from the tail, the snake turns. This kind of movement prevents the snake from circling itself to separate the space into two regions with the apple unreachable in the inner region.

We let the heuristic controller play the deterministic game we used to evolve controllers. It grew the snake to the maximum length and got 44609 points. Checking the results in Table III, all the four EA variants can evolve controllers that outperform the heuristic controller. We also let the heuristic controller play 100 random games. It still grew the snake to the maximum length. The results show that our EA can generate good controllers for a given game but they are not general enough to deal with unknown games. This is an important topic in our next study.

VI. CONCLUSIONS

In this paper we proposed to generate AI controllers for the snake game through four rating functions and EA-based optimization of their weights. The rating functions consider smoothness, space, and food. We also proposed the idea about dynamically adjusting the weights according to the length of the snake. In the experiments, we first examined the need of combining multiple rating functions. Then, we compared four EA variants and identified good crossover operator and environmental method. The benefit of length-related weights in the controller was also confirmed by results of EA optimization and playing random games. Finally, we compared our evolved controller and a heuristic controller. Although our controllers can play better in a given game, they cannot play well in unknown games. In our future work, we will address this weakness by using multiple games in the evaluation of individuals in EA. This could be time consuming, and thus we will rely on parallel EAs. The idea in the heuristic controller such as the connectedness will be included in our future design. We will also investigate better reproduction operators such as those in differential evolution (DE) and particle swarm optimization (PSO).

REFERENCES

[1] M. Gallager and A. Ryan, “Learning to play Pac-Man: an evolutionary, rule-based approach,” *Proceedings of IEEE Congress on Evolutionary Computation*, pp. 2462–2469, 2003.

[2] N. Cole, S. J. Louis, and C. Miles, “Using a genetic algorithm to tune first-person shooter bots,” *Proceedings of IEEE Congress on Evolutionary Computation*, pp. 139–145, 2004.

[3] N. Böhm, G. Kókai, and S. Mandl, “An evolution approach to Tetris,” *Proceedings of The Sixth Metaheuristics International Conference*, 2005.

[4] Y. Shichel, E. Ziserman, and M. Sipper, “GP-Robocode: using genetic programming to evolve Robocode players,” *Lecture Notes in Computer Science*, vol. 3447, pp. 143–154, 2005.

[5] J. Muñoz, G. Gutierrez, and A. Sanchis, “Multi-objective evolution for car setup optimization,” *UK Workshop on Computational Intelligence (UKCI)*, 2010.

[6] C.-Y. Cheng, Y.-H. Chen, and T.-C. Chiang, “Intelligent level generation for Super Mario using interactive evolutionnary computation,” *The 27th Annual Conference of the Japanese Society for Artificial Intelligence*, 2013.

[7] T. Ehlis, “Application of genetic programming to the snake game,” *Gamedev.Net*, 2000.

[8] A heuristic snake AI controller (written in Chinese), http://blog.csdn.net/fox64194167/article/details/19965069