AI LEARNING ON SNAKE PATTERN

Md. Nashiruddin Parvez[1] Umang Sharma[2] Sourav Kumar Behera[3] Geetesh Singh[4] S. Abinayaa[5]

1. SRM Institute Of Science and Technology, Chennai, India
2. SRM Institute Of Science and Technology, Chennai, India
3. SRM Institute Of Science and Technology, Chennai, India
4. SRM Institute Of Science and Technology, Chennai, India
5. SRM Institute Of Science and Technology, Chennai, India

Abstract - This paper describes the evolution of a genetic program to optimize a problem featuring task prioritization in a dynamic, randomly updated environment. The specific problem approached is the "snake game" in which a snake confined to a rectangular board attempts to avoid the walls and its own body while eating pieces of food. The problem is particularly interesting because as the snake eats the food, its body grows, causing the space through which the snake can navigate to become more confined. Furthermore, with each piece of food eaten, a new piece of food is generated in a random location in the playing field, adding an element of uncertainty to the program. This paper will focus on the development and analysis of a successful function set that will allow the evolution of a genetic program that causes the snake to eat the maximum possible pieces of food.

Key Words: AI, Machine Learning, Neural Network, Logical Reasoning

I. INTRODUCTION

Artificial intelligence (AI) techniques have been proven highly successful at the problems of navigation, task prioritization, and problem avoidance. Traditionally, humans have encoded rule-based AIs to create the behaviors necessary to allow an automaton to achieve a specific task or set of tasks. Genetic programming (GP), however, has been proven to allow a computer to create human-competitive results. Specifically, examples such as the wall-following robot (Koza 1992) and Pac Man(R) (Koza 1992) demonstrate the effectiveness of GP at evolving programs capable of navigation and task prioritization behaviors which are competitive with human-produced results.

In an original approach to demonstrating the effectiveness of GP at producing human-competitive results, this paper describes the evolution of a genetic program that can successfully achieve the maximum possible score in the "snake game." The problem posed by the snake game is of particular interest for two main reasons. First, the size and shape of the area through which the main game character, the snake, can move is constantly changing as the game progresses. Second, as the snake eats the single available piece of food on the game board, a new piece is generated in a random location. Because of these two factors, the snake game presents a unique challenge in the developing of a function and terminal set to allow GP to evolve an optimal solution that is generalized for successive runs of the snake game.

The "Background" section of this paper outlines the rules and discusses the specific details of the "snake game." Next, "Statement of the Problem" explains the problem being addressed by this paper. The "Methods" section provides the GP specifics of how the problem was approached. The "Results" section gives numerous examples of results produced by the GP runs along with a discussion and analysis of those results. The "Conclusion" section summarizes the ultimate results achieved by the paper. The "Future Work" section discusses potential for further study in line with the work discussed in this paper. Finally, the "References" section provides a bibliography for the paper.

II. BACKGROUND PROCESS

The "snake game" has been in existence for over a decade and seen incarnations on nearly every popular computing platform. The game begins with a snake having a fixed number of body segments confined to a rectangular board. With each time step that passes, the snake can either change direction to the right or left, or move forward. Hence the snake is always moving. Within the game board there is always one piece of food available. If the snake is able to maneuver its head onto the food, its tail will then grow by a single body segment and another piece of food will randomly appear in an open portion of the game board during the next time step. The game ends when the snake's head advances into a game square that is filled with either a snake body segment, or a section of the wall surrounding the game board. From a task prioritization standpoint, then, the snake's primary goal is to avoid running into an occupied square. To the extent that this first priority is being achieved, its second priority is to pursue the food.

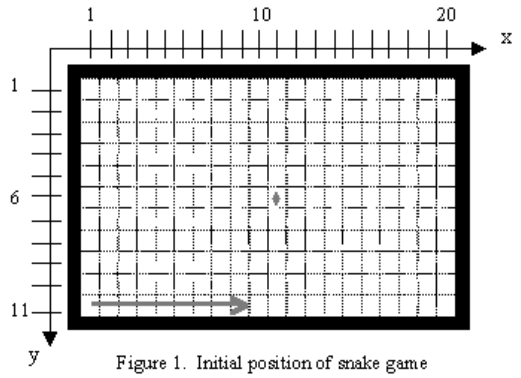


Figure 1. Initial position of snake game

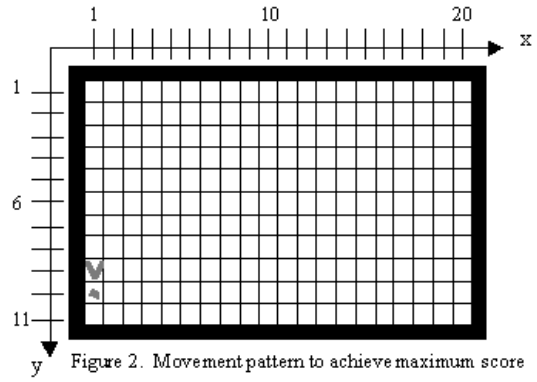


Figure 2. Movement pattern to achieve maximum score

The version of the game used for this paper, shown in figure 1, is a replica of the game as it currently exists on Nokia cell phones. In this version, which is available for play online at www.nokia.com/snake, the game board is made up of 220 total squares, 20 horizontal and 11 vertical, and the food begins in position (11,6) on the game board, represented by a diamond in the figure. The snake is initially made up of 9 body segments, occupying positions (1,11)-(9,11) on the board, with the head in position (9,11) and the snake moving to the right, represented by the arrow in the figure. The maximum number of pieces of food that can be eaten is the size of the game board minus the initial size of the snake. With the given parameters, then, this equates to 220-9=211 pieces of food. This is because with each piece of food eaten, the snake grows by a body segment, reducing the amount of free space in which it can move. Hence when it has eaten 211 pieces of food, its body will fill the entire game board, rendering any further movement impossible. One critical piece of information is whether or not it is even possible for the snake to eat the maximum amount of food. Indeed it is conceivable that after eating a certain amount of food, the snake will have grown so large that it restricts itself from access to a portion of the board. Upon close inspection, however, the reader will note that by tracing certain patterns repeatedly over the board, it is possible for the snake to cover every square exactly once and return to its initial position. One such pattern is shown in figure 2, which features a snake of 210 body segments about to eat the final piece of food. Hence by continually tracing the pattern shown, the snake can eat the maximum possible pieces of food.

III. HARDWARE SPECIFICATIONS

- Display
- Keyboard
- Hard Disk - 1GB
- RAM - 512MB
- Processor - Any Pentium Version

IV. SOFTWARE SPECIFICATIONS

- Windows Operating System
- Python
- Pytest
- Numpy
- Matplotlib
- Tensorflow

V. MODULE DESCRIPTION

Building of Game: In the Initial phase game development will take place using java framework.

Building of AI: Artificial intelligence (AI) is an area of computer science that emphasizes the creation of intelligent machines that work and react like humans. Some of the activities computers with artificial intelligence are designed for include:

Integration of AI With Game: In this phase the AI will learn to play the game and improves itself game by game and eventually reach at it's maximum scoring efficiency.

VI. METHODS

Terminals: The terminal set chosen for the problem was right, left, and forward. Each terminal was a macro that would cause the snake to take the corresponding action during a time step as follows:

Right: the snake would change its current direction, making a move to the right.

Left: the snake would change its current direction, making a move to the left.

Forward: the snake would maintain its current direction, and move forward. This is the same as a no-op, as the snake must make a move. These three terminals represent the minimal terminal set with which the snake can effectively navigate its surroundings. While some problems consisting of navigation in a two-dimensional grid can be successfully navigated by way of only one direction changing terminal, that is impractical for the snake game because the facts that the game board is enclosed and that the snake has an extended body that is impassible necessitate the ability for the snake to move in either direction in order to avoid death. More advance terminals, such as moving the snake along the shortest path to the food, were not implemented. Rather, the function set was constructed in such a manner that the GP could evolve the necessary capabilities to achieve the maximum score.

Functions: Initially the snake was given very limited functionality. One function gave it information about the location of the food, three other functions gave it information about any immediately accessible danger, and progn2 was provided as connective "glue" to allow a function tree to make multiple moves in a single pass. All functions were implemented as macros of arity two, and therefore would only execute one of their arguments depending on the current state of the game, except for progn2, which would execute both of its arguments. Even though no expressions evolved from this initial function and terminal set were able to achieve the optimum score of 211 pieces of food, this set served as a baseline by which to evaluate progress and determine enhancements the would lead to the eventual optimal solution. Following is a description of the initial function set:

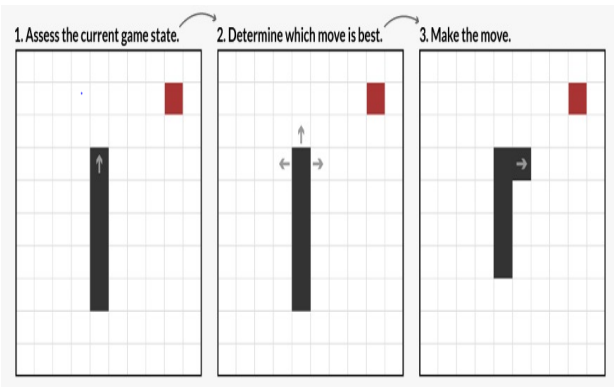
ifFoodAhead: If there is food in line with the snake's current direction, this function will execute its first argument, otherwise it will execute the second argument. This was the only initial function that gave the snake information beyond its immediate surroundings.

ifDangerAhead: If the game square immediately in front of the snake is occupied with either a snake body segment or the wall, this function will execute its first argument, otherwise it will execute its second argument.

ifDangerRight: If the game square immediately to the right of the snake is occupied with either a snake body segment or the wall, this function will execute its first argument, otherwise it will execute its second argument.

ifDangerLeft: If the game square immediately to the left of the snake is occupied with either a snake body segment or the wall, this function will execute its first argument, otherwise it will execute its second argument.

progn2: This is a connectivity function that will first execute its right argument, then its left. It is the only function that allows execution of more than one terminal in a single parse of the function tree. Although this function will always execute both of its arguments, it was necessary to implement it as a macro because of the way that the software used to make GP runs, Dave's Genetic Programming in C (DGPC), evaluated functions vs. macros. To avoid unnecessary modification of DGPC, implementing progn2 as a macro proved the simplest option.



As mentioned previously, no GP runs performed with the initial function set were able to score greater than 123 hits. In order to increase the probability of evolving a function tree capable of achieving the maximum number of hits, the initial function set was enhanced. Functions were added to extend the snake's capabilities for detecting food and danger, as well functions that were conditional on the snake's current movement direction. Following is a discussion of the additional functions that, along with the initial function set, make up the final function set.

Additional Functions, all of arity 2:

ifDangerTwoAhead: If the game square two spaces immediately in front of the snake is occupied by either the wall or a segment of the snake's body, this function will execute the first parameter, otherwise it will execute the second.

ifFoodUp: If the current piece of food on the board is closer to the top of the game board than the snake's head, then the first parameter of this function will be executed, otherwise the second parameter will be executed.

ifFoodRight: If the current piece of food on the board is further to the right of the game board than the snake's head, then the first parameter of this function will be executed, otherwise the second parameter will be executed.

ifMovingRight: If the snake is moving right, then the first parameter of this function will be executed, otherwise the second parameter will be executed.

ifMovingLeft: If the snake is moving left, then the first parameter of this function will be executed, otherwise the second parameter will be executed.

ifMovingUp: If the snake is moving upward, then the first parameter of this function will be executed, otherwise the second parameter will be executed.

ifMovingDown: If the snake is moving downward, then the first parameter of this function will be executed, otherwise the second parameter will be executed.

VII. CONCLUSION

This paper has presented the development and evaluation of a function set capable of evolving an optimal solution to the snake game. An initial function set was presented and evaluated, but proved unsuccessful at evolving an optimal solution. The initial function set was then expanded upon to create the successful final function set, and consistently optimal solutions were generated using primed GP runs. A comparison was made of the results achieved by each function set, as well as by the primed GP runs. Examples of commonly evolved strategies were presented and evaluated, and a final analysis of a consistently successful optimal solution was given.

VIII. FUTURE SCOPE

The work presented in this paper provides innumerable opportunities for further investigation into the evolution of a task prioritization scheme within a dynamically changing, randomly updated environment. Specific to the snake problem, modifications can be made to create completely new and interesting problems, such as a non-rectangular game board, obstacles within the game board, or multiple pieces of food. Multiple snakes could be co-evolved to competitively pursue the food. The function set could be modified to feature enhanced detection capabilities and more advanced navigational options. The techniques used for navigating the snake could be generalized to apply to various other problems of interest. Possibilities include automated navigation of multiple robots through a crowded workspace, an automaton for tracking fleeing police suspects through harsh environments, or a control scheme for an exploratory vehicle seeking a particular goal on a harsh alien planet. The possibilities are only limited by the imagination.

IX. REFERENCES

Koza, John R. 1992. Genetic Programming: On the Programming of Computers by Means of Natural Selection. Cambridge, Massachusetts: The MIT Press.