JETIR.ORG ISSN: 2349-5162 | ESTD Year : 2014 | Monthly Issue
JOURNAL OF EMERGING TECHNOLOGIES AND INNOVATIVE RESEARCH (JETIR)
An International Scholarly Open Access, Peer-reviewed, Refereed Journal

"A STUDY ON SNAKE GAME SOFTWARE"

KM ARACHANA KUMARI [1]

RAJKANYA CHAUDHARI[2]

Indrajeet Tiwari[3]

Dr. Praveen Kumar[4]

Department Computer Science and Engineering

DELHI INSTITUTE OF ENGINEERING AND TECHNOLOGY , MEERUT UP. INDIA

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: algorithm, functions, section, operators, game, snake etc.

INTRODUCTION

Snake is a video game that originated during the late 1970s in arcades becoming something of a classic. It became the standard pre-loaded game on Nokia phones in 1998. The player controls a long, thin creature, resembling a snake, which roams around on a bordered plane, picking up food (or some other item), trying to avoid hitting its own tail or the edges of the playing area. Each time the snake eats a piece of food, its tail grows longer, making the game increasingly difficult. The user controls the direction of the snake's head (up, down, left, or right), and the snake's body follows.

Here is a quick overview of how it is designed. In the snippets below, DOT_SIZE is the width (and height) of the section of the snake say 20 pixels. (You can think of this as something similar to a square on a game board.) MAX_DOTS constant defines the maximum number of possible dots on the board. The size 600x400 defines the size of the board in pixels, so dividing that number by DOT_SIZE*DOT_SIZE computes the number of DOTS (i.e. squares) on the board. Going forward, a part of the snake will occupy one of these dots.

1. DISPLAYING THE BOARD AND A STILL SNAKE

First, we need to display the game board and the snake. Start by creating the file snakegame.html. This will contain all of our code. Next, open the file in your preferred browser.

To be able to create our game, we have to make use of the HTML <canvas>, which is used to draw graphics with JavaScript.

The id is what identifies the canvas; it should always be specified. The id takes the dimensions width and height as props.

Until now, the browser will not display anything since the canvas has no default background. To make our canvas visible, we can give it a border by writing some JavaScript code.

To do that, we need to insert <script> and </script> tags after the </canvas>.

2. MAKING THE CANVAS

Now we can make the canvas, or the game board, for our snake to navigate. First, we get the canvas element using the id gameCanvas (specified earlier).

Next, we get the canvas “2d context", which means that it will be drawn into a 2D space.

We will then make a 400 x 400 white rectangle with a black border, which will cover the entire canvas starting from the top left corner (0, 0).

3. MAKING THE SNAKE

Now, for the snake! We need to specify the initial location of our snake on the canvas by representing the snake as an array of coordinates.

Thus, to create a horizontal snake in the middle of the canvas, at (200, 200), we list the co-ordinate of each body part of the snake.

The number of coordinates in the object will be equal to the length of the snake.

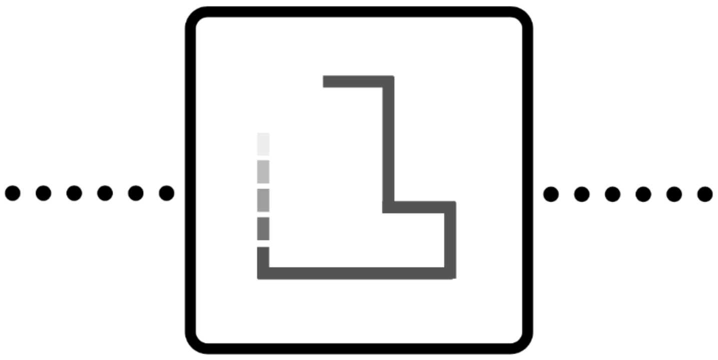
The *yy*-coordinate for all parts is always 200. The *xx*-coordinate is at decrements of 10 to represent different parts of the snake’s body. The very first coordinate represents the snake’s head.

Now, to display the snake on the canvas, we can write a function to draw a rectangle for each pair of coordinates.

Putting step 1 together

Check out the code and click on the Output tab to see the result.

OutputHTML



(Fig.1.1)

4. MAKING THE SNAKE MOVE AUTOMATICALLY

We have our canvas and our snake, but we need the snake to move so it can navigate the game space in all directions.

So, let’s learn how to make our snake move automatically on the canvas.

Horizontal movement

To make the snake move one step (10px) to the right, we can increase the *xx*-coordinate of every part of the snake by 10px (dx = +10).

To make the snake move to the left, we can decrease the x-coordinate of every part of the snake by 10px (dx = -10).

dx is the horizontal velocity of the snake. We need to create a function move_snake that will update the snake.

In the function above, we created a new head for the snake. We then added the new head to the beginning of the snake using snake.unshift and removed the last element of the snake using snake.pop.

This way, all of the other snake parts shift into plane.

Vertical movement

To move our snake vertically, we cannot alter all the *yy*-coordinates by 10px as that would shift the whole snake up and down. Only the *yy*-coordinate of the head needs to be altered.

Decreasing it by 10px to move the snake up and increasing it by 10px to move the snake down will move the snake correctly.

To implement this, we have to update the move_snake method to also increase the y-coordinate of the head by dy (vertical velocity of the snake).

Automatic movement

In order to move the snake, say 50px to the right, we will have to call move_snake(x) 5 times. However, calling the method 5 times will make the snake jump to the +50px position, instead of moving step-by-step towards that point.

To move the snake how we want, we can add a slight delay between each call with setTimeout. We also need to make sure to call drawSnake every time we call move_Snake, as shown below. If we don't, we won't be able to see the intermediate steps that show the snake moving.

clearCanvas() is called inside setTimeout to remove all previous positions of the snake.

Although there is still a problem, nothing tells the program that it has to wait for setTimeout before moving to the next setTimeout. This means that the snake will still jump 50px forward but only after a slight delay.

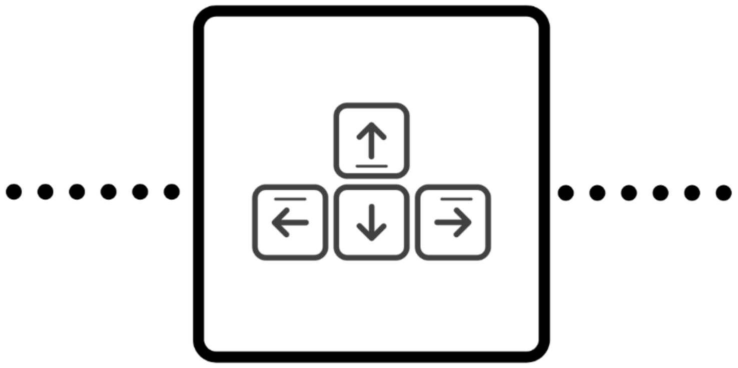
To fix that, we have to wrap our code inside functions. Instead of creating an infinite number of functions that call each other, we can instead create one function (main) and call it over and over again.

Putting step 2 together

Check out the code and click on the Output tab to see the result.

OutputHTML

Now our snake can move! However, once the snake's position moves beyond the canvas boundary, it keeps going forever. We need to fix this by incorporating the use of arrow keys to change the snake's direction.



(Fig.1.2)

5. USING ARROW KEYS TO CHANGE THE SNAKE'S DIRECTION

We have a moving snake, but our next task is to make the snake change direction when one of the arrow keys is pressed.

CHANGING DIRECTION

Let's make the function change_direction. This will check if the pressed key matches one of the arrow keys. If it does, we will change the vertical and horizontal velocity. Look at the function below.

We also need to check if the snake is moving in the opposite direction of the new, intended direction. This will prevent our snake from reversing, such as when you press the right arrow key when the snake is moving to the left.

To incorporate the change_direction function, we can use the addEventListener on the document to listen for when a key is pressed; then we can call change_direction with the keydown event.

ADDING BOUNDARY CONDITION

To prevent our snake from moving infinitely, we need to add boundary conditions. For this, let's make the function has_game_ended, which returns true when the game has ended and false if otherwise.

There are two cases in which the game can end:

- The head of the snake collides with its body.
- The head of the snake collides with the canvas boundary.

These two conditions are incorporated in the code below:

First, there is a check which looks to see if the head has collided with any of the body parts.

If it has not, there is a further check for all of the boundary walls.

Putting step 3 together

Check out the code and click on the Output tab to see the result.

OutputHTML

The snake is now able to change direction when we press the arrow keys. The point of the game is to eat as much food a possible, so we will now learn how to incorporate food and score into the game.

6. INCORPORATING FOOD AND SCORE

Now that we have a fully functional snake, it's time to incorporate food and score into our game.

Food

For the food that our snake will eat, we want to generate a random set of coordinates. Let's make the function random_food to randomly generate an *xx*-coordinate and a *yy*-coordinate for the food's positions. We also have to ensure that the food is not located where the snake currently is.

If it is, then we have to generate a new food location. See the functions below:

We *also* need a function to actually draw the food on the canvas and update main to incorporate the drawFood function.

Growing the snake

The snake will grow whenever the head of the snake is at the same position as the food. Instead of adding a body part to the snake's body every time that happens, we can skip removing a body part in the move_snake function.

Score

Incorporating a score is actually quite simple. We need to initialize a score variable and increment it every time the snake eats the food. To display the score, we will need a new div before the canvas.

We need to further update the move_snake method to incorporate the score:

Putting all the steps together

Check out the code and click on the Output tab to see the result.

OUTPUTJAVASCRIPT

There you go! We now have a fully functional snake game with automatic movement, arrow keys, food, and score. We now have a fully functioning .

7. FUTURE SCOPE

The snake game is one of the simplest game concepts ever, and just like Tetris it's very addictive. Your goal is to move the snake and eat as many "food" blocks as possible. There is only one food block at any given time. When the food is eaten, the snake grows in length.

8. CONCLUSION AND RECOMMENDATIONS OF PROJECT

The coding of Snake was extremely difficult with many errors arising. Many systems had to be written numerous ways before a final working solution was found. For example, two different movement methods were used prior to final version; however, even the final version is flawed as vertical movement causes the snake to change scale. There were also issues with the food – snake collision detection. While the final version resulted in a snake that could eat food, the movement glitch caused the food to cause further size issues.

Despite the fact that the game could not truly be played due to the fact no score could be given, the game is still satisfying. With the exception of the size glitch when turning, the snake responds to user input and moves around the screen as directed. Given longer to work on this, the collision detection with the movement would be the first thing fixed. By fixing this, all other sections of code that are currently not working would run. The leaderboard would work as there would be correct scores input, and the snake would grow as the food would cause it to only increase by one and not varying numbers based on direction. In addition, fixing the movement would allow for the snake to die when colliding with itself. In the current state, the snake moves as a matrix so it can not kill itself as it would be impossible to move in any direction. This failure to establish a perfect movement system was the biggest disappointment of the game as all other problems stemmed from it.

For these reasons, it is recommended that anyone who wishes to recreate this game starts simply when writing the code. It is advisable that they first perfect the snakes movement controls before messing with the food generation. By taking the code in small sections, it is easier to get individual features to work. Building off this, use functions to contain each aspect of the game. Using functions made it easier to determine where errors were occurring when debugging the code. It also kept the code more organized.

9. REFERENCES

[1]. BLATTNER, M.M. AND DANNENBERG, R.B., 1990, CHI '90 Workshop on Multimedia and Multimodal Interface Design, SIGCHI 22(2), October 1990, 54-58.

[2]. CRANE, G. AND MYLONAS, E., 1988, The Perseus Project: An Interactive Curriculum on Classical Greek Civilisation, Educational Technology 28(11), November 1988, 25- 32.

[3].FRITZ, J.M., 1991, HyperCard Applications for Teaching Information Systems, SIGCSE Bulletin, 23(1), March 1991, 55-61.

[4].GRIMES, J. AND POTEL, M., 1991, What is Multimedia?, IEEE Computer Graphics and Applications, 11(1), January 1991, 49-52.

[5].KATZ, E.E. AND PORTER, H.S., 1991, HyperTalk as an Overture to CS1, SIGCSE Bulletin, 23(1), March 1991, 48-54.

[6].MARCHIONINI, G., 1988, Hypermedia and Learning: Freedom and Chaos, Educational Technology 28(11), November 1988, 8-12.

[7].MARCHIONINI, G. AND SHNEIDERMAN, B., 1988, Finding Facts vs. Browsing Knowledge in Hypertext Systems, Computer, January 1988, 70-80.

[8].MORARIU, J., 1988, Hypermedia in Instruction and Training: The Power and the Promise, Educational Technology 28(11), November 1988, 17-20