Snake

Game play:

This is an old retro video game that became especially popular when it came bundled with the Nokia phones. In this game, a snake moves around the screen with arrow keys. The length of the snake increases when it eats apples and decreases when it eats bad cherry.

See this Wikipedia page for more information:

https://en.wikipedia.org/wiki/Snake_(video_game_genre)

If you want to play this game to get an idea of its features, click the link provided at the bottom of this chapter.

Download all my program files here.

Basic features:

I have implemented these features in my program and will explain how to design them in the following sections.

- The target food (apple) could be eaten up in 3 bites.
- The snake's speed increases slightly after eating an apple and decreases slightly after eating a bad cherry.
- The snake's length reduces if it eats the bad cherry. This cherry could appear periodically at random positions. The snake needs to avoid the cherry to eat the food. The cherry will not disappear though. Slowly along the way the screen will get populated with the cherries and the snake will have to avoid all of them to keep its size from reducing.
- Game is over if (a) the snake hits the obstacles or screen edges, or (b) loses all its body (length becomes 0).

Advanced features:

These are for you to explore and implement.

- The target food (apple) could be eaten up in 3 bites.
- Disallow eating itself
- Multicolor snake (striped along the length)
- Moving obstacles
- Timer

Concepts used in this chapter

Be sure to learn or revisit these concepts before you undertake this project.

- Algorithms
  - Abstraction
  - Using algorithms
  - Designing new algorithms
  - Pseudo-code
- Arithmetic
  - Expressions
  - Basic operators (+, -, *, /)
- Concurrency
  - Synchronization using broadcasting
- Conditional statements:
  - Conditions: YES/NO questions
  - Sensing touch
  - Relational operators (=, <, >)
  - Conditionals (IF)
  - Conditionals (If-Else)
  - Conditionals (Wait until)
  - Conditionals (nested IF)
  - Boolean operators (and, or, not)
- Data structures – list
  - List operations
  - List traversal
- Data types – basic
  - Integers
- Divide and conquer (program design technique)
- Events
- Looping (iteration)
  - Looping - simple (repeat, forever)
  - Looping - nested
  - Looping - conditional (repeat until)
- Motion
  - Motion - absolute
  - Motion - relative
  - Motion - smooth using repeat
- OOP
  - Clones
- Pen commands
  - Basic

- Random numbers
- Sequence
- Stopping scripts
- User input
    - Keyboard events
    - Keyboard events (polling)
- Variables
    - Simple
    - Properties (built-in)
    - Local/global scope
    - As timer
- XY Geometry

Version "starter":

In this version, we will try to implement these features:

- Snake moves around the screen by following the arrow keys.
- The 'Len' parameter allows snakes of different lengths.

Brainstorming:

We have dealt with challenges of eating prizes and avoiding penalties in other games. What is new in this game is the variable length of the Snake. The other challenge is how to show the Snake's body bending on turns. We can't use costumes because the Snake can bend in infinite number of ways.

Let's consider each challenge separately.

(1) How to make the Snake's length variable?

One idea that comes to mind is we could use Pen commands to actually draw the Snake. That way, we can have it of any length.

But how do we draw a moving Snake? The CLEAR command clears the whole screen. We need to somehow erase only the trailing part of the Snake's body. Can you think of how to achieve this?

How about having someone continuously eating the Snake's body as it moves?

"Erasing" can be considered as "painting with the background color", right? So, we could have another invisible sprite following the Snake's tail continuously and erasing the Snake's body as it moves.

With this idea, we can easily change the Snake's length by simply changing the distance between the Snake's head and the "eraser" sprite.

(2) How do we show the Snake's body bending on turns?

With a Snake that is drawn continuously, it is not too hard to show it bending on turns. As the "Head" sprite turns, it will obviously draw a body that is bent around the turn. The challenge is ensuring the "Eraser" sprite takes the exact same turns as the "Head" sprite. For this purpose, we could record the movements undertaken by "Head" in a list so that the "Eraser" will know how to follow correctly.

We will call this the 'starter' version. Let us get started with it.

Feature idea #1: Head sprite

Have the Snake head follow arrow keys to move around the screen. It will draw the Snake's body as it moves and record its moves in a list.

Design:

The following screenshot shows a sample of a Snake head moving around the screen.

Since we don't have the "Eraser" sprite yet, the Snake will continuously grow in length as the Head moves around. We will use a list called 'Movements' to keep track of every turn and distance traveled on that turn. The format of this list is explained in Feature idea #2 below.

Here is the algorithm for the Head sprite.

```
Algorithm for head:
Given:
```

```
Len: Snake's current length
Movements: list to record Head's movements. It will initially have 1 item
(= Len) because that's the distance 'Head' needs to move to draw the full
Snake.

When arrow key pressed:
If arrow direction is different from current heading
      Add new heading to 'movements'
      Append 0 to 'movements'
End if
Move
Increment last location in 'movements'
Tell 'eraser' to follow (send a broadcast)
```

The last step above would be useful only after we implement Feature idea #2 below.

Feature idea #2: Eraser sprite

Have an Eraser sprite follow the Snake's tail as 'Head' moves around the screen. Eraser will erase the Snake's body beyond the tail.

Design:

The 'movements' list is essentially for the 'Eraser' sprite to know how to follow 'Head'. Hence, this list will contain pairs of data: each pair consisting of distance (traveled before a turn) and heading (indicating which way the turn was taken). For the sake of our discussion, let us assume that Snake's length is 100. Let us assume every keypress causes a movement of 5 steps. 'Movements' will already contain one item (100) because that's how much 'Head' has already travelled to draw the body.

Here is an example of how 'head' might travel after the game begins:

```
Initially facing east.
Go 50 steps.(right arrow key 10 times)
Go 110 steps south. (down arrow key 22 times)
Go 10 steps east. (right arrow key 2 times)
Go 5 steps north. (up arrow key once)
```

The resulting list would be as follows. Explanation is given in parentheses.

```
150 (100 Snake's length plus 50)
180 (south)
110
90 (east)
10
0 (north)
25
```

'Eraser' will just follow as per the above list. It will also update the list as it moves. Items once taken care of (moves and turns) would be deleted.

Here is the algorithm for 'Eraser'.

```
Algorithm Eraser Follow:
Given:
Movements: list of moves

After Head moves on arrow key (it will send a broadcast):
If 'movements' is empty
      Error: movements must have at least 1 item
Else
      If 1st item in 'movements' is 0 (which means current path has been
covered)
            Turn as per 2nd item in 'movements'
            Delete first pair of items in 'movements'(since they are no
longer needed)
      End if
      Move
      Decrement 1st item in 'movements'
End if
```

Feature idea #3: Setup

Set up the sprites at the start of the game.

Design:

Initially, we will position 'Head' and 'Eraser' somewhere in the left area of the screen. Both would face east with their pens down. 'Head' would move to draw the Snake, but 'Eraser' will stay put. Eraser's pen color would match the background. We will keep Eraser's pen thickness slightly more than Head's to ensure the erasing is clean. We will initialize all the variables.

Save version:

We will save this version as snake-starter. All it does is lets you move the Snake around using your arrow keys.

Version 1:

In this version we will include the following features:

- Score variable
- "Apple" sprite is placed randomly on screen, when eaten it disappears, score goes up. It then reappears at another random location.

- Include "bad cherry": placed randomly on screen, when eaten score goes down. Multiple cherries continue appearing all over the screen.
- Game is over if 'Head' touches screen edges.

Feature idea #4: Apple and bad cherries

Step 1: An "Apple" sprite is placed randomly on screen. When eaten by the Snake it disappears and 'score' goes up. Apple reappears at another random location.

Design:

This is straightforward. We will repeatedly place the Apply at a random location on the screen. It will hide after being touched by the Snake and reappear at another location. 'Score' will go up every time.

```
Algorithm for Apple (Food):
Forever
      Appear at a random location
      Wait until touching Head
      Increment Score
      Hide
End
```

Step 2: Include a "bad cherry" sprite placed randomly on screen, when eaten score goes down. Multiple cherries continue appearing all over the screen.

Design:

Unlike the apple, cherries continue to appear continuously even if not eaten by the Snake. We could either use STAMP or clones. Since we need the cherries to disappear after being eaten by Snake, "clones" is the only practical idea.

Once again, we will have a loop that will create cherry clones and place them at random locations. Each clone will then wait to be eaten up by Snake. 'Score' will go down every time.

```
Algorithm for bad cherry:
Create clones continuously:
      Appear at random location
      Each clone waits until touching Head
      Decrement Score
      Delete clone
```

Step 3: Game stops (Snake dies) when it touches any of the screen edges.

This is just a matter of using the 'wait until touching edge' command.

Save version:

We will save this version as snake-1. It lets you move the Snake around using your arrow keys. When it eats apples the score goes up, when it eats bad cherries the score goes down. The game is over when the Snake touches a screen edge.

Version 2:

We will add the following features in our next version.

- Have a variable for Snake's width.
- Use a 'Speed' variable to control Snake's speed. It should increase slightly when food is eaten and decrease when bad cherry is eaten.
- Snake's length increases/decreases on eating food/poison.
- Game is over when Snake loses all its body (length = 0).

Brainstorming:

Having a variable 'width' will allow us to control Snake's width. We will use it as 'pen width' that draws (and erases) the Snake.

Using 'speed' to control the Snake's speed is straightforward: just insert it in all 'move' commands of both 'Head' and 'Eraser'.

How will the Snake's length change? Let's think about decrease/increase separately.

(1) Increase Snake's length:

To make the Snake look longer, we will move 'Head' but not 'Eraser'. We should increment the appropriate item in 'movements' to indicate this movement. We will also increment the 'Len' variable.

(2) Decrease Snake's length:

We will decrement the 'Len' variable. How will we visually reduce the Snake's length?

The simplest way is to move 'Eraser' forward. We can re-use Eraser's "follow head" script for this purpose, even though 'Head' hasn't moved.

The next question is:

(3) By how much should we increase or decrease Snake's length?

Changing it by 1 step would show very little visible change on the screen.

So, let us say we change by it by some number C. We need to figure out if there is any relation/constraint between C and Len. Let us take an example to understand this better.

Len=50, C=3

Let's say the Snake's length continues to decrease. Will it ever become 0? It won't because 'Len' is not a multiple of C.

Thus, the first constraint that we must account for is: **Len must be a multiple of C.**

How about 'speed'? So far, our 'Len' and 'speed' parameters were fixed, and in fact, we took 'Len' as a multiple of 'speed'. Now that 'Len' may vary (and even 'speed' will vary), will there be any impact on the motions of 'Head' or 'Eraser'? Let us investigate.

Let us say right at the beginning of the game we have:
Len=77, speed=5

Let us now take 'Head' through the following set of moves:

```
3 right arrow keys (will travel 15 steps)
2 down arrow keys (will travel 10 steps)
```

This will create the following 'Movements' list:

```
92 (distance 'eraser' must travel before turning south)
180 (south heading)
10
```

Now, clearly 'Eraser' will not cover a distance of 92 correctly because every move takes 5 steps. After 18 correct moves (18x5=90) the nineteenth move will go wrong. **We will need to modify the algorithm of 'Eraser'** such that it compares each move with the distance in 'movements' and breaks it up if necessary. In the case of 92, 'Eraser' should go 2 steps east and then 3 steps south.

Feature idea #5: Speed and width paramters

Use variables for Snake's width and speed.

Design:

This is straightforward. Having a variable 'width' will allow us to control Snake's width. We will use it in the command 'set pen width' for both 'Head' and 'Eraser'.

Using ‘speed’ to control the Snake’s speed is straightforward: just insert it in all ‘move’ commands of both ‘Head’ and ‘Eraser’.

Every time an apple is eaten ‘speed’ will increase slightly, and it will decrease slightly when a bad cherry is eaten. We will use percent change instead of arithmetic change. The following lines cause a 5% change:

```
Speed = speed x 1.05 (increase)
Speed = speed x 0.95 (decrease)
```

Feature idea #6: Snake length should vary

Snake’s length should increase/decrease on eating food/poison. The game should terminate when Snake loses all its body (length = 0).

Design:

The scripts for increasing or decreasing Snake’s length will be called by Apple and Bad cherty sprites when they touch ‘Head’. As discussed in the ‘brainstorming’ section the amount of increase/decrease will be some number C.

```
Algorithm to increase Snake’s length:
Increment the ‘Len’ variable by C
Move ‘Head’ by C
Increment the last item in ‘movements’ by C.

Algorithm to decrease Snake’s length:
Decrement the ‘Len’ variable.
Ask Eraser to run its “follow head” script
```

As discussed in the ‘brainstorming’ section above, 'Len' must be a multiple of 'C'. During the initial set up we can ensure this as follows:

```
if remainder of (Len/C) > 0
     Len = integer division of (Len/C) * C
```

The game termination should now include the Snake’s length becoming 0.
Actually, due to the widths of ‘Head’ and ‘Eraser’ sprites, the actual length for which Snake’s body disappears completely is > 0. In my program, it’s about 20.

Finally, we will need to modify the algorithm of ‘Eraser’ such that it compares each move with the distance in ‘movements’ and breaks it up if necessary.

```
Algorithm Eraser follow (updated):
Given:
Movements: list of moves
```

```
After Head moves on arrow key (it will send a broadcast):
If 'movements' is empty
      Error: movements must have at least 1 item
Else
      If 1st item in 'movements' < speed
            Temp = speed – 1st item in 'movements'
            Call Move Eraser with input=1st item in 'movements'
            Call Move Eraser with input=Temp
      Else
            Call Move (speed)
      End if
End if

Algorithm Move Eraser (used by the above algorithm):
Input: distance d
If 1st item in 'movements' is 0 (current path has been covered)
      Turn as per 2nd item in 'movements'
      Delete first 2 items in 'movements'(they are no longer needed)
End if
Move d
Decrement 1st item in 'movements' by d
```

Save version:

We will save this version as snake-2. It lets you move the Snake around using your arrow keys. When it eats apples the score goes up and its length increases. When it eats bad cherries the score goes down and its length decreases. The game is over when the Snake touches a screen edge or when its length becomes 0.

Version final:

We will include the following features in the next version.

- Include obstacles (different every time) that Snake must avoid. Game terminates when 'Head' touches an obstacle.
- Placement of apple and cherries should avoid obstacles.
- Help screen.
- Snake should stop moving after game is over. (In current version arrow keys can move it.)

Brainstorming:

Having a Help screen is just a matter of creating a separate sprite which shows up in the very beginning.

Including obstacles is a matter of designing obstacle sprites and placing them randomly on the screen. We could use a basic shape and create its clones and then use a simple algorithm to create a layout of obstacles around the screen.

Now that we have these obstacles, we need to place apples and cherries such that they don't overlap obstacles. This is a simple matter of "trying" random placement on the screen until a spot is found where they are not touching any of the obstacles.

Finally, how do we ensure that Snake stops moving when the game is over? Right now, we are using Event-driven scripts which will continue running even after the game is over. The solution is to replace these with "polling-based" scripts.

Let's implement these features one by one.

Feature idea #7: Obstacles

Insert obstacles (different every time) that Snake must avoid. Game terminates when 'Head' touches an obstacle. Placement of apple and cherries should not overlap with obstacles.

Design:

Including obstacles is a matter of designing obstacle sprites and placing them randomly on the screen. We will use the following basic shape as an obstacle:

We will use a simple algorithm to create multiple such obstacles around the screen. Following is our algorithm which creates clones of this sprite and places them in a matrix format. The Random operator helps us in two ways: (1) To get different orientation of this obstacle. (2) To pick some of the obstacles.

```
Algorithm obstacle placement
Draws layout in a 3x4 matrix (row/column) format
Go to x1, y1 (some place in left upper corner)
Repeat 3
      Repeat 4
            Coin toss (using random)
            If Head
                  Create clone
```

```
            End if
            Go to next column
            Select heading at random
      End repeat
      Go to beginning of next row
End repeat
```

As discussed in 'brainstorming' section we will "try" random placement of apples and cherries on the screen until a non-overlapping spot is found:

```
Go to random position
Repeat until NOT touching obstacles
      Go to random position
End repeat
```

Feature idea #8: Stopping snake movements

Snake should stop moving after game is over. (Pressing arrow keys should not cause it to move.)

Design:

Right now, we are using Event-driven scripts which will continue running even after the game is over. The solution is to replace these with "polling-based" scripts. See an example below:

```
Forever
      If right arrow down
            Move right
      End if
End forever
```

Save version:

We will save this version as snake-final. It provides all features that we had planned.

MIT Scratch site: https://scratch.mit.edu/projects/599732199/

How to run:

1. Click Green flag. Read help and click to continue.
2. Use arrow keys to move the snake.
3. Eat apples to gain points and increase length and speed. Eating bad cherries loses points and causes length and speed to decrease.
4. Game terminates when Snake touches any of the screen edges or obstacles or loses all its body.

Advanced version:

Let us see if we can have a more colorful snake, maybe one with a stripe in the middle.

The idea is simple: just like we have a hidden eraser sprite that follows the Snake's head at a distance and erases the Snake's body, we could have another hidden sprite following the Snake's head and drawing a thin line. This would create the effect of a stripe on the Snake's body.

For this to work we need to modify the algorithm used by 'Eraser' to follow 'Head', because in the current algorithm, 'Eraser' modifies and deletes the record of 'Head's movements. Since we have another sprite that would need this record, we need an algorithm that keeps 'Movements' intact.

Instead of modifying 'Movements' the follower sprite could simply traverse the list using its own index. See a possible algorithm below:

```
Algorithm Follow at a distance 'd':
Given:
a record of moves - list 'Movements' of distance and turns
Signal every time sprite moves (broadcast)
Local index i into 'Movements', initially 1
Local d = distance covered, initially 0

Steps:
If d >= Movements[i]
      new heading = Movements [i+1]
      i = i + 2
      d = 0
End if
move s steps
d += s
```

The beauty of this algorithm is that it can be used by both 'Eraser' and 'Stripe' sprites. The only difference is they would be following at different distances ('d'). And of course, they would be performing different functions.

One problem is that the list 'Movements' will continue to build up indefinitely since nobody is modifying/deleting it. This is clearly a waste of memory since once all 'follower' sprites have traversed the items they are no longer needed.

One solution is to have the last follower sprite do a "cleanup" operation every once in a while. For instance, 'Eraser' could delete the first 40 items of 'Movements' when it is at item 50, because everyone else would also be well past the first 40 items. See the addition below:

```
Algorithm Follow at a distance 'd':
Given:
a record of moves - list 'Movements' of distance and turns
Signal every time sprite moves (broadcast)
Local index I into 'Movements', initially 1
Local d = distance covered, initially 0
```

```
Steps:
If d >= Movements[I]
      new heading = Movements [I+1]
      I = I + 2
      d = 0
End if
move s steps
d += s
If I > 25
      delete first 20 items in 'Movements'
      ask everyone to reset their indexes
End if
```

All follower sprites when they get 'reset' signal:

```
I = I - 20
```

Other minor changes:

To ensure that 'Eraser' is the last one to follow 'Head', it should first tell others (in this case only 'Stripe') do complete their following.

When Snake's length increases, we move 'Head' ahead a bit. The 'stripe' must also be moved accordingly. 'Head' will initiate this by sending a message *after* it has moved ahead. In response, 'Stripe' must use its 'Follow Head' procedure to ensure all variables are properly updated.

Save version:

Congratulations! Save this version as snake-advanced. It provides some of the advanced features that we had planned.

How to run:

1. Click Green flag. Read help and click to continue.
2. Use arrow keys to move the snake.
3. Eat apples to gain points and increase length and speed. Eating bad cherries loses points and causes length and speed to decrease.
4. Game terminates when Snake touches any of the screen edges or obstacles or loses all its body.