SnakePit

January 2018

SnakePit	1
Purpose Capture Designer Clock1	2
	3
	3
Graphics	4
cellPX	4
R	4
Cell storage	4
row	4
columns	5
rows	5
column	5
cell	5
centerX	6
centerY	6
erase	6
draw	6
global_snake_storage	7
Link storage	7
Screen1_Initialize	8
seed	8
next	8
prev	9
Clock1_Timer	9
hatch_a_random_egg	9
eggs	10
randomColor	10
move_a_snake	10
move	11
snake	11
location	11

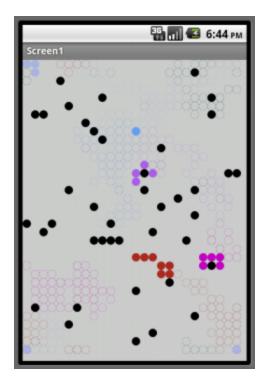
Other Projects	19
Gallery link	19
move_to	18
last	18
kill	17
revert_all	17
color	16
attack	16
eat	14
first	14
link	14
owner	13
try_target	13
wrap	12
neighbors	12

Purpose

This project demonstrates how to use lists to run a multiple Snake simulation on a Canvas, without using any Sprites or Balls. Instead, it draws Circles on grid squares (cells) for snake eggs, and hatches a few of them into colored snakes that move randomly and grow in length as they eat free standing snake eggs. When a snake bites another snake, the tail breaks off and reverts to individual snake eggs.

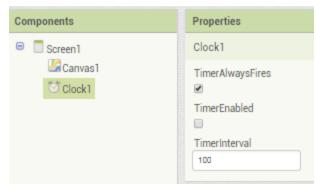
This is an advanced project, and employs doubly linked lists.

Capture



The black dots are snake eggs. The colored dots are snakes, fairly tightly coiled because of the random motion automatic movement routine. The brown snake has eaten the most eggs so far, and the light blue snake has eaten no eggs yet. The colored bubbles are artifacts of the drawing and erasing of the colored dots that make up the snakes, as the snakes move. (I rather like them, so I have left them as is.)

Designer Clock1



The Designer is as simple as can be, with just the Canvas and a Clock.

The Clock is disabled, to allow clean startup, and hard wired for 100 milliseconds per tick. Feel free to run the clock faster to experiment.

Graphics

cellPX

```
to cellPX result 12
```

This function returns the number of pixels in the side of each grid cell.

R

```
to R result call cellPX / (2)
```

R is the radius of the circles used to draw the snake bits. Anything other than half the cell size leaves debris on the canvas or makes the snakes look too loose.

Cell storage

Cells are identified by row and column number, 1 based. To combine both the row and column number into a single value, we assume no more than 999 rows or columns, and express a (row, column) as a single number, rrrccc, using multiplication by 1000. To go back and forth between cell identifiers and rows and columns, we do modulo math.

We don't actually keep a 1,000 by 1,000 array. We just keep the locations of the small fixed set of eggs that we start with, that serve as links in the chains that are our snakes.

row

```
result quotient of get cell ÷ (1000)
```

Given a cell ID, return the row number of that cell.

columns

```
to columns
result floor Canvas1 . Width fall cellPX
```

Return how many columns will fit onto the Canvas based on its Width and the number of pixels per cell.

rows

```
result floor Canvas1 . Height / Call cellPX
```

Return how many rows will fit onto the Canvas based on its Height and the number of pixels per cell.

column

```
to column cell
result remainder of get cell ÷ 1000
```

Extract the column number from a cell ID.

cell

Build a cell ID from a row number and column number.

centerX

```
to centerX column
result 
get column - - (0.5) × call cellPX -
```

Return the x value (in pixels) for the center of a circle at a given column number.

centerY

```
result | O | call cellPX - | call cellPX - |
```

Return the y value (in pixels) for the center of a circle at a given row number.

erase

```
to erase cell
do call draw cell get cell color Canvas1 BackgroundColor
```

Erase the circle at a given rrrccc cell. This technique leaves a bubble on the Canvas. I could have changed this to draw a fat line to fill the entire rectangular cell, to eliminate the bubble, but I like the bubble trail left by the snake.

draw

```
to draw cell color

do set Canvas1 . PaintColor to get color .

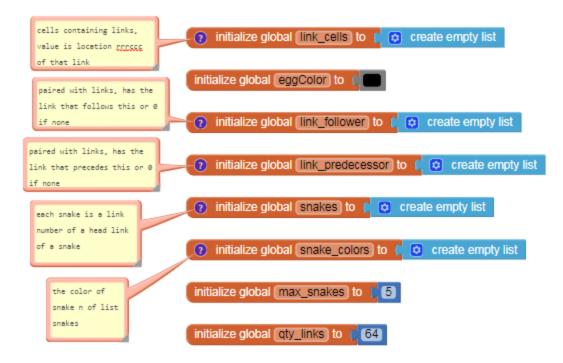
ci initialize local x to call centerx column call column cell get cell .

initialize local y to call centery row call row cell get cell .

in call Canvas1 . DrawCircle centerx get x centery get y radius get call R fill true .
```

Several layers of conversion take the <u>row</u> and <u>column</u> numbers from the cell ID, then extract the <u>centerX</u> and <u>centerY</u> graphic coordinates for the center of that cell. Then we draw a circle of radius <u>R</u> and the given color at that location.

global_snake_storage



Link storage

Snakes are chains of links in a doubly linked list of constant size, once the board has been seeded with eggs. The constant **qty_links** controls how many links (rrrccc items in list **link_cells**) will be produced, and that number will remain unchanged after the seeding operation. Because that number remains unchanged, we can make parallel lists of predecessor and follower link numbers (indices into **link_cells**). An egg at position 23014 (row 23, column 14) whose link is at index 9 in list **link_cells** will start out with 0 at index 9 of lists **link_follower** and **link_predecessor**, since it is a chain of length 1.

When we hatch an egg, we add its starting link number to list **snakes**. We limit the number of snakes for ecological reasons, based on constant **max_snakes**. List **snakes** has a matching list **snake_colors**, holding the color of snake **i** in list **snakes**. As snakes are born and die, we have to keep those lists in parallel.

A third list **snake_directions** might be used to keep persistent snake directions, to allow faster movement than the current Brownian Motion random model.

Screen1_Initialize

```
when Screen1 · Initialize

do call seed · n get global qty_links ·

set Clock1 · TimerEnabled · to true ·
```

At app startup, we <u>seed</u> the lists of links and their structures, and paint them as eggs. We then start the Clock Timer to get some snakes hatched and moving.

seed

```
to seed
    for each (number) from (1) to (
                                    get n by (
         initialize local 🕜 to 🚺
                                random integer from
                                                              call rows -
            initialize local 🕻 to 🕻
                                random integer from
                                                              call columns -
              add items to list list |
                                     get global link_cells •
                                                              item call cell r
                                                                                  get 📭 c ( get 🖙
              add items to list
                               list 7 get global link_follower -
                                                                 item 0
              🙎 add items to list 🛚 list
                                     get global link_predecessor *
                                                                     item 🖟 🛈
             call draw -
                     cell
                           call cell r get r c get c -
                           get global eggColor -
                   color
```

To seed the board, we generate the required number of links, pointing to random row and column locations, and draw an egg at each location. Being eggs, they have no followers and predecessors for the chains they would be part of if they were parts of snakes.

(Bug alert: I did not check for random duplicates here.)

next

```
to next link

result select list item list get global link_follower index get link
```

To concisely navigate chains of links, we provide <u>next</u> and <u>prev</u> functions accepting a link (index into <u>link_cells</u>).

prev

```
to prev link

result select list item list get global link_predecessor index get link
```

Clock1 Timer

```
when Clock1 - .Timer

do if length of list list get global snakes - set get global max_snakes - then call hatch_a_random_egg - call move_a_snake -
```

The Clock ticks rapidly, and does just a little bit at each tick, hatching one snake if there aren't enough snakes, and regardless, moving one snake.

hatch a random egg

```
to (hatch_a_random_egg)
initialize local eggs to call eggs
      initialize local egg to
      initialize local cell to
     initialize local color to 📗
                               call (randomColor -
                        is list empty? list
      then
                              pick a random item list
                                                         get eggs
                              select list item list 🍍
                                                   get global link_cells -
                                                                            index 🥛 get (egg)
                 add items to list
                                         get global snakes
                                                                       get egg -
             add items to list
                                        get global snake colors
                                                                     item 📜 get color -
             call draw - cell
                                  get cell
                                              color
                                                       get color
```

The first step in hatching a random egg is to go through the lists of links, and collect the links that have no followers or predecessors, i.e. eggs. It's possible that there are none left, because they are parts of snakes now. In that case, we do nothing. If there is at least one egg on our list of eggs, we pick a random egg from that list. Since the returned egg is from a list of link indices in list link cells, we need to look up the cell (rrrccc) for that link to draw it with a random color.

We record the head link of the new snake in global list <u>snakes</u>, and its color in the matching list <u>snake colors</u>.

eggs

```
to eggs
result
         initialize local eggs to
                                      create empty list
                    for each (link) from (11)
                                             to | length of list list | get | global link_cells -
                         if
                                                call prev - link
                                                                   get (link -
                                    and 🕶
                                                                   get (link -
                                                call next - link
                                add items to list list get eggs -
                         then
                                                   item
                                                           get (link -
               result
                        get eggs
```

Eggs have no previous or next links.

randomColor

move a snake

```
to move_a_snake

do call move snake_index random integer from 1 to length of list list 2 get global snakes 1
```

To move a random snake, we pick a random snake from list <u>snakes</u> and <u>move</u> it.

move

```
to move snake_index

do initialize local head to call snake snake_index get snake_index

initialize local target to 0

in set target to pick a random item list call neighbors cell call location link get head call try_target snake_index get snake_index target get target.
```

To move a snake, we first need to look up the head link of that snake by its snake index, using our <u>snake</u> function. We then extract the rrrccc <u>location</u> of that link, and generate a list of its <u>neighbors'</u> rrrccc values. For the simplest possible implementation, we settle for picking a random neighbor rrrccc to try, using routine <u>try_target</u>.

snake

```
to snake snake_index

result select list item list get global snakes index get snake_index
```

This is a simple lookup in global list snakes.

location

```
result select list item list get global link_cells index get link
```

Links are indices into the list of rrrccc cells, global list link cells.

neighbors

```
result initialize local c to call column cell get cell initialize local c to call column cell get cell initialize local c to call column cell get cell initialize local c to call column cell get cell initialize local result to c create empty list in do for each dR from 1 to 1 by 1 do for each dC from 1 to 1 by 1 do do add items to list list get result item call cell cell call wrap n 2 get result call rows null move c call wrap n 2 get c + get dC imit call columns result get result get result index 5
```

The cell ID is broken down into its <u>row</u> and <u>column</u> numbers, and its 8 surrounding cells are calculated, <u>wrapped</u> around the edges of the board, and reassembled into rrrccc <u>cell</u> IDs and returned.

wrap

```
to wrap
                                                    limit
                           result
                                                get n = < -
                                                                  1
force n into limit of 1
... limit, by wrapping
                                              get (limit)
                                     then
                                     else
                                                                   > •
                                                                           get (limit)
                                                         get n -
                                                      1
                                              then
                                              else
                                                       get n -
```

We deal with the edge of the board by using doubly cylindrical geometry.

try_target

```
to try_target
                  snake_index
     initialize local (target_snake_index) to
                                               call owner - cell
                                                                    get (target
                                     get (target -
                                                          get global link_cells
         then
                if
                              get target snake index -
                                                               get snake_index
                then
                       get target_snake_index -
                                                                      0
                       then
                              call eat snake_index
                                                                             target
                                                         get snake_index -
                                                                                      get (target
                                   attack - snake_index
                                                           get snake_index -
                                                                                target
                                                                                         get (target)
         else
                call [move_to +
                    snake_index
                                   get snake_index •
                      target_cell
                                   get (target 🔻
```

There are three possibilities when we want to move a snake onto a target cell:

- 1. The target has an egg, a link that's not part of a snake (no owner). Eat it.
- 2. The target has a link that's part of another snake. Attack it.
- 3. The target has no link, so it's empty. Move to there.

owner

```
result in initialize local link to call link cell get cell in if get link > 0 then index in list thing call first link get link get link get global snakes else 0
```

To find the owner (snake, if any) of an rrrccc cell, we first look up the <u>link</u> (if any) pointing to that cell. If a link is found (non-zero), then we look up the <u>first</u> link in whatever chain that link might be part of, then try to look up that head link in our list of snake heads (global **snakes**). Failure at any stage returns 0.

link

```
result index in list thing get cell list get global link_cells
```

This function accepts a cell (row, column) and returns the link (index into links list) of the link on that cell, or 0 if the cell is empty.

first

```
to (first)
              link
result
         initialize local first to
                                        get (link -
             initialize local prev to
                     set prev to
                                        call prev - link
                                                            get first -
                                                            0
                                       get prev -
                           set first - to
                                             get prev -
                           set prev - to
                                              call prev - link
                                                                  get (first -
                          get (first -
                result
```

This function accepts a link number, and follows the <u>previous</u> links to the very head of the chain (snake) holding this link.

```
to eat
         snake_index
                        target
initialize local target_link to
                                    call (link - cell
                                                       get (target -
          initialize local head to
                                    call snake - snake_index
                                                                  get snake_index -
    call draw -
              cell
                      get (target -
             color
                      call color - snake_index
                                                   get snake_index -
      replace list item list
                            get global link_predecessor -
                   index
                            get (head -
            replacement
                            get [target_link -
      replace list item list
                            get global link_follower -
                   index
                            get [target_link +
            replacement
                            get (head -
                            get global snakes -
      replace list item list
                            get snake_index -
                   index
             replacement
                            get [target_link -
```

This procedure is for the case where a snake encounters an egg, a solitary link, and absorbs it into itself (eats it.) The **target** parameter is the cell (location) holding the target <u>link</u> to be eaten. The mouth of the snake is in its **head**, the link we can look up in the <u>snake</u> table.

To show that the link has been eaten, we change its <u>color</u> to that of the snake eating it. To absorb the link into the snake,

- we make it the predecessor of the old head link,
- Point the follower of the new head back to the old head, and
- Update the snakes list to point to the new snake head link.

attack

```
to attack snake_index
                           target
 initialize local (target_snake_index) to
                                              call owner - cell
                                                                    get (target
              initialize local (target_link) to
                                              call link - cell
                                                                 get (target -
         initialize local (new_target_tail) to (
                                              call prev - link
                                                                  call link - cell
                                                                                      get (target -
 in [
      🔯 if
                    get new_target_tail - > -
             replace list item list
                                    get global link_follower -
      then
                           index |
                                    get new_target_tail -
                    replacement
             call kill snake_index
                                         get_target_snake_index
      call revert all - link
                               get [target_link -
```

This is the case where a snake bites another snake at a target cell location. First we need to identify the victim, the <u>owner</u> of the target cell. We also identify which <u>link</u> is at the target cell. Because we are merciful, we will allow the target snake to escape and heal if it has been bitten anywhere behind its head, so we will need to identify the link that will serve as its new tail link. That link is the <u>previous link</u> to the bitten link.

If we have a new tail for the target snake, we seal it off by setting its follower link to 0. Otherwise, we have eaten the head of another snake, and must kill that snake entirely.

Regardless, the bitten target link and its followers in that snake are now loose food, and must revert to eggs.

color

```
result select list item list get global snake_colors index get snake_index
```

This function returns the color of a snake, based on the lookup table **snake_colors**.

```
to revert_all
 initialize local (links_to_revert) to create empty list
               initialize local (next) to
                                         get (link -
     while test
                      get next - > -
                                       get [links_to_revert -
           add items to list list
                               item
                                       get next -
            set next to
                             call next - link
                                                 get next -
      for each (link) in list
                             get [links_to_revert -
           replace list item list
                                  get global link_follower
                                  get (link -
                         index
                  replacement
                                  0
                                  get global link_predecessor
            replace list item list
                                  get (link -
                         index
                  replacement 0
            call draw -
                            call location - link
                                                  get (link -
                   color
                           get global eggColor -
```

This procedure accepts the head link of a chain of links, that need to be de-chained and reverted back to eggs. It is a two phase operation:

- Gather the links in the chain into a list, following the next links,
- Revert each link in the list to an egg, by clearing its predecessor and follower links and redrawing it black.

kill

```
to kill snake_index

remove list item list get global snakes index get snake_index

remove list item list get global snake_colors index get snake_index

remove list item list get global snake_colors index get snake_index
```

This procedure kills a snake, by removing it from the snakes list, and cleaning up any matching lists. It does no link manipulation; that's left to the calling <u>attack</u> routine.

```
to (last)
result
             initialize local last to
                                        get (link -
             initialize local next to
                     set next - to
                                        call next - link
                                                            get [last -
                     while test
                                                           0
                                       get next -
                                             get next
                           set last to
                           set next - to
                                              call next - link
                                                                   get (last -
                          get (last -
```

This routine follows the follower chain of a link, returning the last link in the chain.

move to

```
to move_to snake_index target_cell
                                          initialize local (head) to
                                                                           call snake - snake index
                                                                                                            get snake_index -
                                           initialize local [last_link] to [ 0
                                               set [last_link - to ]
                                                                      call [last - link
                                                                                           get head -
          ve of a snake involves
                                               call draw -
      series of slides from back
                                                                 get [target_cell -
                                                         cell
      to front, ending at the
                                                       color
                                                                 call color - snake_index
                                                                                                get snake_index -
      target cell. The snake head
       emains the same link,
                                               call erase - cell
                                                                      call (location - link
                                                                                               get [last_link -
      though its location
      changed.
                                                                                                           > • (
                                                                 call prev - link
                                                                                       get [last_link]
                                                     replace list item list
                                                                              get global link_cells
                                                                              get [last_link -
                                                             replacement
                                                                              call location - link
                                                                                                       call prev - link
                                                                                                                             get [last_link -
                                                      set (last_link - to
                                                                             call prev - link
                                                                                                  get [last_link -
                                                replace list item list
                                                                        get global link_cells -
                                                                        get [last_link -
                                                       replacement
                                                                        get [target_cell]
8 0
rnings
```

This is complex, simulating the movement of a caterpillar. First the drawing is handled at the new head of the snake, and at its old tail. Then we start a wave at the tail, lifting each link and dropping it onto the location of its predecessor, pushing the wave forward towards the head of

the snake, until we reach the head of the snake (prev = 0). All this movement happens in the list <u>link cells</u>, which maps the link numbers into their cell locations.

Gallery link

http://ai2.appinventor.mit.edu/?galleryld=5592523749457920

Other Projects

 $\underline{https://docs.google.com/document/d/1acg2M5KdunKjJgM3Rxpy \ Rf6vT6OozxdIWglgbmzroA/ed} \\ \underline{it?usp=sharing}$