Animal Prison Puzzle Game

How to play:

Click green flag or “New Game” to start playing. Press h to see help. You can play the game yourself by clicking on the animals. Or, you can click "Solve" any time to let the computer try to solve the remaining puzzle.

Introduction:

The aim of the game is to free all animals from the prison (the 4x4 grid).

To free these animals from the prison make them exit from right (east) by clicking one at a time. Each clicked animal lines up in a "freedom queue" below the grid.

The 4x4 prison contains 16 animals:
4 animal types: crab, dinosaur, dog, penguin
4 animal colors: red, blue, yellow, green

These are randomly placed all facing east.

Rules:

1. An animal can exit the grid only if there is no one in front of it in its own row.
2. An animal can exit the grid only if its color or type matches with the last animal in the queue below the grid.

Solution approach:

We will use the following ideas in our solution:

- To draw the 4x4 grid we will simply use the "Matrix" program from the book "Practice CS Concepts with Scratch".
- We will have 16 sprites: 4 animal types each with 4 colors.
    - Each sprite will have a unique id, from 1 thru 16.
    - Each sprite will have private variables telling us its type and color.
- To represent the 4x4 grid, we will use a list L1 with 16 items.
- To represent the free queue, we will use a list L2 whose length would depend on how many are free.
- Manual play:
    - When an animal is clicked, depending where it is (L1 or L2), appropriate action will be taken. Rules of the game will be checked before it is moved.
- Computer play:

- We need to devise an algorithm that would implement the "exhaustive search" method.

Data structures:

The 4 animal types are designated as a, b, c, d. The 4 colors are designated as 1, 2, 3, 4.

1. Private variables:
    a. ID for each animal: 2 letter: e.g. a1 means animal "a" with color "1", d3 means animal "d" with color "3", etc. A private variable (myID) will hold this value for each animal.
    b. Each animal will have an id 1 thru 16, which never changes.
    c. Each animal will save its location in the grid in private variables myrow and mycol.

2. L1 - list of 16 will indicate cells in the grid. Left to right and bottom to top. If a cell is empty, it contains 0. Else it contains the animal id.

3. L2 - list that will show animals freed from the grid. Its length will vary as animals enter or leave it. L2 will hold animal ids.

Stack for local variables and return values:

Refer to the blog article "Script local variables" that describes how to use a "stack" list to implement local variables and return values. We will use this feature in our program at certain places.

Other global variables:

Row – used during the initial animal placement in the grid
Column – ditto
Count – ditto
Xorigin – northwest corner of the grid (used to calculate positions of other cells)
Yorigin – ditto
cellSize – size of each cell in the grid
Status
Elapsed – used during "auto" mode
Number of moves – used
Width – used for animal positioning in the "free queue"

Algorithms:

Setting up:

Green flag or “New game” to set up a new game.

The following things need to happen during setup:

1. Every sprite sets its own pre-game state: variables, visibility, etc.
2. A 4x4 grid of squares is drawn.
3. The 16 animal ids 1 thru 16 are saved in list L1 in a random order. We use Knuth's shuffle algorithm.
4. Each animal shows itself in the grid according to its location in L1.
5. Prompt user to play.

Clearly all above actions can be initiated through broadcast messages.

Placing animals at the start:

The list L1 will be primed during setup with numbers 1 to 16 in random order. Then, a broadcast "draw face" will be sent for each animal to listen and move itself to its location.

Algorithm for an animal to place itself in the grid:

```
Given: L1 contains 1 to 16 randomly, indicating who is supposed to go
where.

Procedure:
The sprite coordinating the animal placement will go through a repeat
loop and cycle a counter variable from 1 thru 16.

Animal sprites, upon receiving the broadcast message will do the
following:
If current position in L1 indicates my id
     Calculate my x and y coordinates
     Position using x, y
     Save my position (in myrow and mycol)
End if
```

During game-playing when user clicks on an animal:

Determine if it's in the grid or free.

- Use the "contains" operator to check if it's in the L2 list. If yes, the animal is free, else in the grid

(A) If it's in the grid: rules below will be checked and the animal will be added to L2 and moved to the queue at screen bottom.

- Verify there is no one ahead in the same row. (If there is someone, do nothing.)

```
Verify column=4 OR
Verify [row, column+1] is empty.
     Location in L1 = (row-1)*4 + column+1
     Verify item at this location is zero.
```

- If L2 is not empty, verify color matches with color or type of the last item in L2.

```
If L2 is empty, go ahead with the move
Get color and type of the last item in L2 (see below)
```

```
Compare with the clicked animal, if either type or color is
equal, go ahead with the move.
```

(B) If it's in the free row and is the last one in L2, it will be moved back to its position in the grid.

- `Calculate [x, y] position in grid using saved myrow and mycol.`

Getting color and type of the last animal in L2:

The animal that needs this info will send a broadcast.
Only the animal whose id matches the last one in L2 will respond:

- `It will copy its color-type (myID) in the last element of "stack".`

Algorithm to shuffle 1 to 16 in L1:

```
(adapted Knuth's algorithm)
I = 0
Repeat 16 times:
     R = I + Random(0,15-I)
     swap(R+1, I+1)
     I = I + 1
End repeat
```

Utility: Move from L1 to L2

```
Add animal id to L2
Replace L1 location to 0
Resize and move the animal sprite just behind the last animal in the
free queue.
```

Utility: Move from L2 to L1

```
Delete the last item in L2
Replace target location in L1 by id
Calculate [x,y] using myrow, mycol.
Position and reset size.
```

Algorithm for solving by computer:

This is the interesting and challenging part of the program.

We basically use the "exhaustive search" idea (some people call it the "Brute force" approach) to systematically try out all possible moves until a solution is found.

Since the computer can't click, we use "click simulation" which means the script that runs when a sprite is clicked is now invoked through a broadcast message. All animal scripts are thus modified. This does not affect the "manual play" behavior.

Approach:
We scan L1 from beginning and look for a "candidate" animal, i.e. an animal that can be moved to L2. Then we simulate a click on it. If no animal can be found, we undo the last move, i.e. we simulate a click on the last free animal. Then we once again scan L1 for a "candidate" animal, but from the place where the "undo" animal returned.

Every time the search for a "candidate" animal succeeds, we reset the search location to the beginning of L1.

We essentially repeat this process until either L1 becomes empty (which means success), or L2 becomes empty (which means there is no solution).

Here is the algorithm:

```
Location N = 1
Repeat until L2 has 16 animals:
```

Search for candidate animal:

```
Scan L1 from N for an animal that can be moved (which has no animal in
front)
   - Simulate click
   - Continue this scan until click succeeds (meaning all rules for
     the move are satisfied) OR until no candidate animal is found in
     L1
```

After search:

```
If success:
      Reset N = 1
Else
      If L2 is empty
            Declare "No solution to this puzzle"
            Stop
      Else
            Get grid location of last animal in L2.
                  Use "Get grid location" algorithm below
            N = location of moved animal + 1
            Move last animal in L2 back to grid.
            Go to "search for candidate animal" above
      Endif
Endif
End repeat
```

Getting grid location of the last animal in L2:

The animal that needs this info will send a broadcast.
Only the animal whose id matches the last one in L2 will respond:

- It will calculate its grid location using its myrow and mycol variables, and copy it in the last element of "stack".

Solutions:

Manual play only: animal-grid-1.sb2
https://scratch.mit.edu/projects/322429109/

Final (including "computer play"): animal-grid-final.sb2
https://scratch.mit.edu/projects/322461995/

Author: Abhay B. Joshi (abjoshi@yahoo.com)
Last modified: 28 July 2019