Snakes and Ladders

This is a popular game in which players traverse a series of numbers 1 thru 100. They encounter ladders and snakes along the way which help or impede their progress. Look up the following Wikipedia page to understand the rules of this game.

https://en.wikipedia.org/wiki/Snakes_and_ladders

In this program, we will have two players: one the user and the other the computer.

Explore the game:

If you want to play with my final program to get a feel for this game, click the link given at the end of the article. Try not to peek at the scripts yet, since we want to design them ourselves below.

1. Click the "Green flag". Read instructions and click to continue.
2. You will play with the computer. The program randomly picks the first one to go.
3. Click the die when it is your turn and watch the chips move. Do this again and again until one of the players wins.

Click here to download all program files.

Snap and CS Concepts Used

When we design this program, we will make use of the following Snap and CS concepts. Learn these concepts if you don't know them before proceeding further.

- Algorithms
    - Abstraction
    - Using algorithms
    - Designing new algorithms
- Animation using costumes
- Arithmetic
    - Expressions
    - Basic operators (+, -, *, /)
    - Advanced operators: mod, floor, ceiling, etc.
- Concurrency

    - Synchronization using broadcasting
- Conditional statements:
    - Conditions: YES/NO questions
    - 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)
- Motion
    - Motion - absolute
    - Motion - relative
- Procedures
    - User defined (custom)
    - Simple
    - With inputs
    - With return value
- Program output
    - Text
- Random numbers
- Recursion
- Sequence
- Sounds - playing sounds
- Stopping scripts
- User input
    - Text
    - Input validation
    - Mouse events

- Variables
    - Simple
    - Properties (built-in)
    - Local/global scope
- XY Geometry

Version 1 High Level Design

As usual, we will build our program step by step. It is clear that we will need the following sprites:

- The board (we can simply borrow an image from the Internet)
- Chips (we can draw these ourselves)
- The cat for miscellaneous work

The game consists of moving the chips on the board and there are two types of movements:

- Direct motion between 2 cells (e.g. for snakes and ladders)
- Row-wise motion that occurs when the die is cast, i.e. a chip moves from the current position to another number (max by 6 positions)

Once we get these movements working, the rest of the game would be easy.

Let us begin with a single chip and get these movements working.

Feature Idea # 1: Set things up

Get sprites for the board and a single chip.

Design:

You can use my starter file `snakes-ladders-starter.sb2` or do this yourself.

Download an image of the snakes and ladder board from the Internet. For the chip, we will draw a filled circle. We will resize the circle such that it fits within a single cell of the board.

Next, we will get the following measurements of the board which we will need for our movements. We can get these by trial and error (by moving the chip around the board).

- X and Y positions of #1 on the board (using which we can calculate X, Y of any other cell) and save them in global variables `startX` and `startY` (in my program they are -93, -149).
- Width of each square cell (in my program it is 33.35).

Feature Idea # 2: Move between two cells

Create an algorithm to directly move from current cell to any other cell on the board.

Design:

We can view the board as a matrix of 10 rows and 10 columns counting from the bottom-left corner. Row 1 is numbers 1 thru 10, row 2 is numbers 20 thru 11, row 2 is numbers 21 thru 30, and so on. Column 1 is 1 thru 100 (1, 20, 21, 40, …, 100), column 2 is 2 thru 99 (2, 19, 22, 39, …, 99), and so on.

If we know the row and column numbers of a cell, we can calculate (using simple Cartesian geometry) its X and Y positions using the following equations:

```
X = startX + width x (column – 1)
Y = startY + width x (row – 1)
```

You can verify these formulae by trying different values of row and column and see if "go to X, y" makes the chip go the correct location on the board.

But, how to find the row and column of any given cell (numbered from 1 thru 100) on this board?

If we think about it carefully, the row number seems to be a bit straightforward: it increases by 1 for every 10 cells. In other words, it is related to the *integer division* of *cell value* and 10. After a bit of trial and error, we come up with the following relation:

```
row = (integer division of (cell – 1) and 10) + 1
```

Let us try this out for a few cell values as below:

```
cell = 5, row = 1
cell = 20, row = 2
cell = 89, row = 9
```

Next, let's work out the column number.

It appears that column number depends on the direction in which the 'cell' value increases. When 'cell' value increases from left to right (such as in row 1 or 3), the relation is as follows:

```
column = (Remainder of (cell - 1) / 10 ) + 1
```

You can verify this formula by trying a few values of 'cell':

```
cell = 8, column = 8
cell = 50, column = 10
cell = 43, column = 3
```

When 'cell' count increases from right to left (such as in row 2 or 4), the relation is as follows:

```
column = 11 - [(Remainder of (cell - 1) / 10 ) + 1]
```

You can verify this formula by trying a few values of 'cell':

```
cell = 17, column = 4
cell = 31, column = 10
cell = 54, column = 7
```

So far so good!

Now, the final challenge is to figure out the direction in which 'cell' increases! This appears easy because the direction of increase is from *left to right* in all "odd" rows and *right to left* in all "even" rows.

So, here is the complete algorithm:

```
Algorithm Move direct to another cell:
Given:
cell (a number 1 thru 100)
Steps:
```

```
row = (integer division of (cell - 1) and 10) + 1
if remainder of row/2 is 0 (which means the row is even):
      column = 11 - [(Remainder of (cell - 1) / 10 ) + 1]
Else:
      column = (Remainder of (cell - 1) / 10 ) + 1
End if
X = startX + width x (column - 1)
Y = startY + width x (row - 1)
Glide to X,Y
```

Feature Idea # 3: Move by distance

Move the chip from its current cell by the given distance. For example, if the current cell is 21 and distance is 5 the chip should move to 21+5=26.

Design:

As described above, this particular movement is needed when a player throws the die. The chip would move by whatever number the die returns. If the current cell is C1 and the die returns D (a number between1 and 6), the chip would move from C1 to C1+D.

Prima facie, this movement may appear to be just a special case of what we implemented in Feature idea #2 above. But it is not! It will not always be a straight move between C1 and C1+D. Instead, the chip will trace the gradual increase in cell value. For example, if C1=8 and D=6, the chip would first move to 10, then to 11, and finally to 14.

Of course, we can certainly use the "move direct" algorithm for each of these sub-steps.

First, let us think about how we can work out the sub-steps.

Every row ends with a multiple of 10. If the destination cell C2 exceeds the current row's end cell, we are going to have multiple sub-steps.

Step 1: Find out current row's end-cell value E (which would be a multiple of 10).
Step 2: If C2 > E create sub-steps, else make a direct move to C2.

Creating sub-steps would mean:

- Sub-step 1: C1 to E, decrement D by E-C1
- Sub-step 2: E to E+1, decrement D by 1

- Sub-step 3: E+1 to C2

Here is the final algorithm:

```
Algorithm move by distance:
Given:
C1 = current cell of the chip
D = distance of the movement (1 thru 6)
Steps:
C2 = C1 + D
If C1 is a multiple of 10
      E = C1
Else
      E = ((integer division of C1 and 10) + 1) x 10
End if
If C2 > E
      Move direct C1 to E, decrement D by E-C1
      If D > 0
            Call "Move direct" to E+1, decrement D by 1
      End if
      If D > 0
            Call "Move direct" to C2
      End if
Else
      Call "Move direct" C2
End if
```

While we are at it, we will create a private variable `current` for the chip sprite: it will always indicate its current position on the board.

Finally, we will need to include a special case: at the very start of the game, the chips are outside the board, with `current = 0`. At this point, we can simply call the "move direct" script.

Feature Idea # 4: Custom blocks for movements

Create custom blocks for the movement scripts.

Design:

This is a straightforward change. Create 2 new custom blocks: one for each of these movement scripts. A custom block is generally better than a broadcast script because it is guaranteed to invoke only your own script. For example, if the chip sprite sends the broadcast message "move by distance", it will run all scripts in your program that are

triggered by that message, whereas if it calls the custom block "move by distance", it will only invoke its own custom block.

Save as Program Version 1

Congratulations! You have completed all the features of Version 1. Compare your program with my program at the link below.

File: snakes-ladders-1.xml

How to run this program:

1. Click the "Green flag".
2. Press 1 to enter a cell number (1 thru 100). The chip will move there.
3. Press 2 to throw the die and make the chip move. You can do this again and again.

Version 2 High Level Design

In this version, we will include a "die" sprite that the user can throw by clicking on it. We also add the most interesting feature of the game: snakes and ladders that affect your journey on the board. Finally, we will also deal with the situation when a chip reaches the number 100.

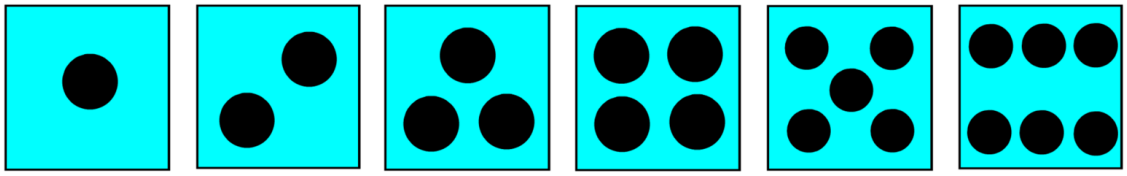
Feature Idea # 5: Include a die

Include a "die" sprite that the user can "throw" by clicking on it.

Design:

We will create a die sprite by having a square with 6 costumes: each showing a different face of the die. Clicking it will cause it to randomly show one of its 6 faces. That face will be associated with the 'die' variable.

The following images show the 6 faces (costumes) of the die.



To create the impression that the die has been thrown, we can switch costumes a number of times before settling on one of them. The 'costume #' property will tell us its associated value (i.e. the number of dots).

Feature Idea # 6: Include snakes and ladders.

Allow the chip to climb ladders and be swallowed by snakes.

Design:

This is obviously the next exciting part of the game. We will now make the snakes and ladders on the board come alive. On our board, there are 8 ladders and 7 snakes. As you can see, each of them goes from one cell to another. So, we could save their positions in two separate lists. Each item (snake or ladder) will occupy a pair of numbers in the list: one indicating its 'start' and the other its 'end' position. For example, snake (95,56) indicates that it starts in cell 95 and ends in 56.

Thus, the 'snakes' list will contain 14 items: 2 for each snake. And the 'ladders' list will contain 16 items: 2 for each ladder. We will load these lists when the Green flag is clicked.

Next, we need to implement the logic. After making a move, the chip will check if there is snake or ladder that begins in its current cell by searching these lists. If so, it will acquire the 'end' location (of the snake or ladder) and moves there.

This search script will set the global variable 'cellContent' to 'snake', 'ladder' or 'none'. In addition, it will set variable 'to' to the end point of the given object.

Below, we see the algorithm to search for a snake. We will a similar algorithm for ladders.

```
Algorithm Search for snakes
Given:
```

```
cell = current position of the chip
Steps:
For every pair of items in the list 'snakes':
      If first item matches cell
            cellContent = "snake"
            to = second item
            stop search
      End if
End for
```

Feature Idea # 7: Cell with 100

Deal with what happens when the chip reaches the cell 100.

Design:

There are two ways in which a chip can reach the cell 100.

A. Become the winner.

If the chip ends up in this cell, it is the winner. It can then send a message about winning.

B. Bounce back.

If the chip reaches this cell but hasn't completed its move, it will bounce back. For example, let's say the chip is at 98 and the die returns 6. According to game rules, the chip will move 2 steps to 100 and then 4 steps backwards to 96.

In this case, we basically have make 2 direct moves:

- Move direct to 100
- Move direct to 100 – (die – (100 – current)) which is = 200 – (die + current)

Here is an algorithm to implement this feature.

```
Algorithm bounce at 100
Given:
current = current position of the chip
die = a number 1 thru 6
Steps:
If current + die > 100
      to = 200 - (die + current)
      Move direct to 100
      Move direct to to
End if
```

Save as Program Version 2

Congratulations! You have completed all the features of Version 2. Compare your program with my program at the link below.

File: snakes-ladders-2.xml

How to run this program:

1. Click the "Green flag".
2. Throw the die by clicking the die sprite and watch the chip move. Do this again and again.

Next version High Level Design

In this version, we will include a second player and add other niceties such as sounds and welcome and help screens.

Feature Idea # 8: Second player.

Include a second player.

Design:

A variable called 'Turn' will indicate whose turn it is. User will click the die for each player alternately. The chip whose turn it is, will perform a move.

Feature Idea # 9: Add sounds and help/support screens

1. Include appropriate sounds when the chip encounters a snake or a ladder.

Design:

It would be nice to give audio feedback to the player(s). Encountering a snake should elicit a sad sound, and a ladder a happy sound.

2. Include a welcome/help screen.

Since this is a fairly well-known and easy game, we can just have one screen (sprite) that combines help and welcome. This welcome/help screen would be a separate sprite which would get out of the way upon clicking anywhere.

Save as Program Version 3

Congratulations! You have completed several important features of the game. Compare your program with my program at the link below.

File: snakes-ladders-3.xml

How to run this program:

1. Click the "Green flag". Read instructions and click to continue.
2. The two players will take turns. The program randomly picks the first one to go.
3. Throw the die by clicking the die sprite and watch the chips move. Do this again and again until one of the players wins.

Final version High Level Design

In this version, we will make computer the second player. So, a human player will get to play against the computer. We will reserve the 'blue' chip for the computer.

Feature Idea # 9: Play with computer

Automate the 'blue' chip's play.

Design:

This change is really quite straightforward. Right now, the die can be thrown only by clicking it. We now need to have a way to "simulate" clicking. Rest of the work is already in place. To simulate clicking, we just need to put the die's script under a broadcast message.

The following script will ensure that manual clicking continues to work for human players.

```
When this sprite clicked:
```

```
If it is not blue's turn
     Call 'throw the die' script
End if
```

The blue chip will continuously monitor the 'turn' variable; when its turn comes it will invoke the simulated die script by sending the required broadcast message.

We will also make a couple of minor changes:

- Each chip will monitor if it wins (i.e. reaches cell 100) and announces it.
- The 'referee' sprite will stop the game once a winner is declared.
- The 'die' sprite will disregard any clicks during computer's turn since the click is supposed to be 'simulated'.

Save the final version

Congratulations! You have completed all features of the game. Compare your program with my program at the link below.

File: snakes-ladders-final.xml
Snap website: https://snap.berkeley.edu/snap/snap.html#present:Username=abjoshi&ProjectName=snakes-ladders-final

How to run this program:

1. Click the "Green flag". Read instructions and click to continue.
2. You will play with the computer. The program randomly picks the first one to go.
3. Click the die when it is your turn and watch the chips move. Do this again and again until one of the players wins.

Author: Abhay B. Joshi (abjoshi@yahoo.com)
Last updated: 14 September 2021