State Machines – Example by Snake

Mission

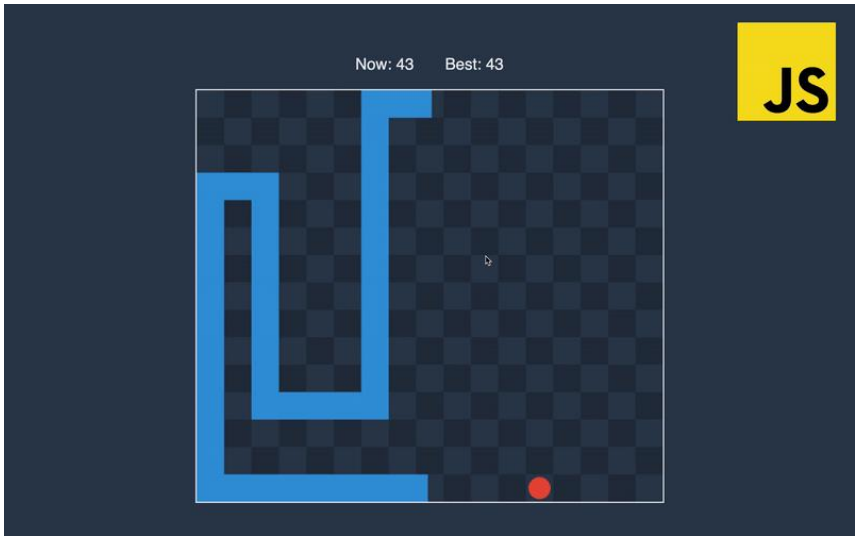
To make some simple snake game using two different implementations of state machines.

Background

- Theory: https://ocw.mit.edu/courses/6-01sc-introduction-to-electrical-engineering-and-computer-science-i-spring-2011/6d24bc51571a1a945a63ffa8343a5b55_MIT6_01SCS11_chap04.pdf
- Wiki: https://en.wikipedia.org/wiki/Finite-state_machine
- Fun intruduction: https://www.c-sharpcorner.com/article/understanding-state-design-pattern-by-implementing-finite-state/
- State Pattern: https://www.dofactory.com/net/state-design-pattern

Big Picture

The idea is you should implement the control part of the old fashion snake game.



See https://medium.com/@geekrodion/snake-game-with-javascript-part-3-2599f7c77f21

The control could be asdw: a=left, s=backwards, d=right and w=forward.
The directions is: up = North, down = South, left= West and right = East.

If the snake is moving West, then control 'w' is still moving West, 'a' turn left and move South, 'd' turn right and move North and here 's' is ignored therefore still moving West

Assignment 1- Create States and Transitions

You are to create a state diagram based on the background above of controls and directions, where you modelling the move directions as states and the controls as transitions.

Assignment 2 - State Pattern Machine

You are to create a Console Application project (optional a library) where to implement the state machine specified in assignment-1.

Create an interface e.g. 'IState' having this method:

```
Move NextMove(InputType input);
```

Where **InputType** is an enum with the values "left, right and forward" (no meaning of going back) and the **Move** is a record Move (int row, int col) telling the next position, relative to current position. That will say for a position (10,14) move (1,0) gives the new position (11,14) while move (0,-1) gives the new position (10,13).

Create a class SnakeStateMachine1 that implement the interface. You should use the State Pattern to implement the State Machine i.e. create an interface and a class for each State. Where the SnakeStateMachine1 have a current state (the interface) that refer to the object of the current state.

In your console application try out this new implementation of the state machine.

In the Main create the following:

1. A method the read a key (a,s,d,w) as the next input.
2. An instance of the StateMachine.
3. An instance of a SnakePlayground (see below)
4. Make a game loop (read next input, use the state-machine to get next movement, perform movement in the playground)

The SnakePlayground should be a class to hold the playground for the snake e.g. 20 times 20 squares (properly some parameters) and the head of the snake somewhere. (NB! In this first version, there is now body nor tail of the snake. Neither exists any 'food'). Moreover, should the class have a method to move the snake (-head) and throw an exception if the border have been reached (and later also check it's not move into itself).

And for some first-time view make a very simple printout of the playground including the snake-head.

Assignment 3 - Table Driven States Machine (a little harder using double arrays)

Create a class SnakeStateMachine2 that hold the state machine still implementing the IState interface. Where table should have two dimensions one for the states and one for the input (transitions), for each pair (input, state) you should have the action to take (continue, turn 90˚clockwise or counter clockwise) and the new state (moving North, East, West or South).

The class should have an instance field to hold the current state (direction) and a method with a parameter of the next input. Which find the next action (consider if this should be the return value) and set the new current state using the table.

(additional) You could now make a UnitTest to test your StateMachine are working properly.

to move the snake (-head) and throw an exception if the border have been reached (and later also check it's not move into itself).

Which one do you find the best in this case?

Assignment 4 – Make the state machine generic

For the implementation, refactor them to have two generic types one for inputs and one for states

Assignment 5 – Make two snakes

Update the reading keys to use jkli (j=left, k=backwards, l=right,i=forward) and have two snakes.

Assignment 6 – Add a body and tail to the snake

A snake should have a fixed length of body+tail of five unit.

Assignment 7 – Add food to the playground

Make food appear randomly at the playground (one at the time) the snake can eat.

And eventually grow one unit in length.

Assignment 8 – Add GUI to the snake game