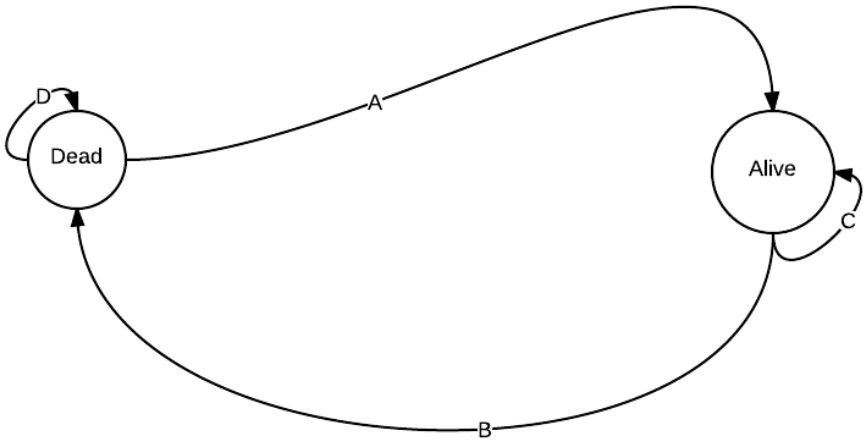
Game of Life

Problem Statement

The Game of life is cellular automaton with zero players, where every output is dependent on previous generations called “seed’. The whole universe of game is divided in grids with a cell can only be possibly in two states : Dead or Alive . The finite state machine for the conditions are as follows.



A. Any dead cell with exactly three live neighbours comes to life.
B. Any live cell with fewer than two live neighbours dies, as if by loneliness.
B. Any live cell with more than three live neighbours dies, as if by overcrowding.
C. Any live cell with two or three live neighbours lives, unchanged, to the next generation.
D. Any dead cell which does not fulfills A , remains dead.

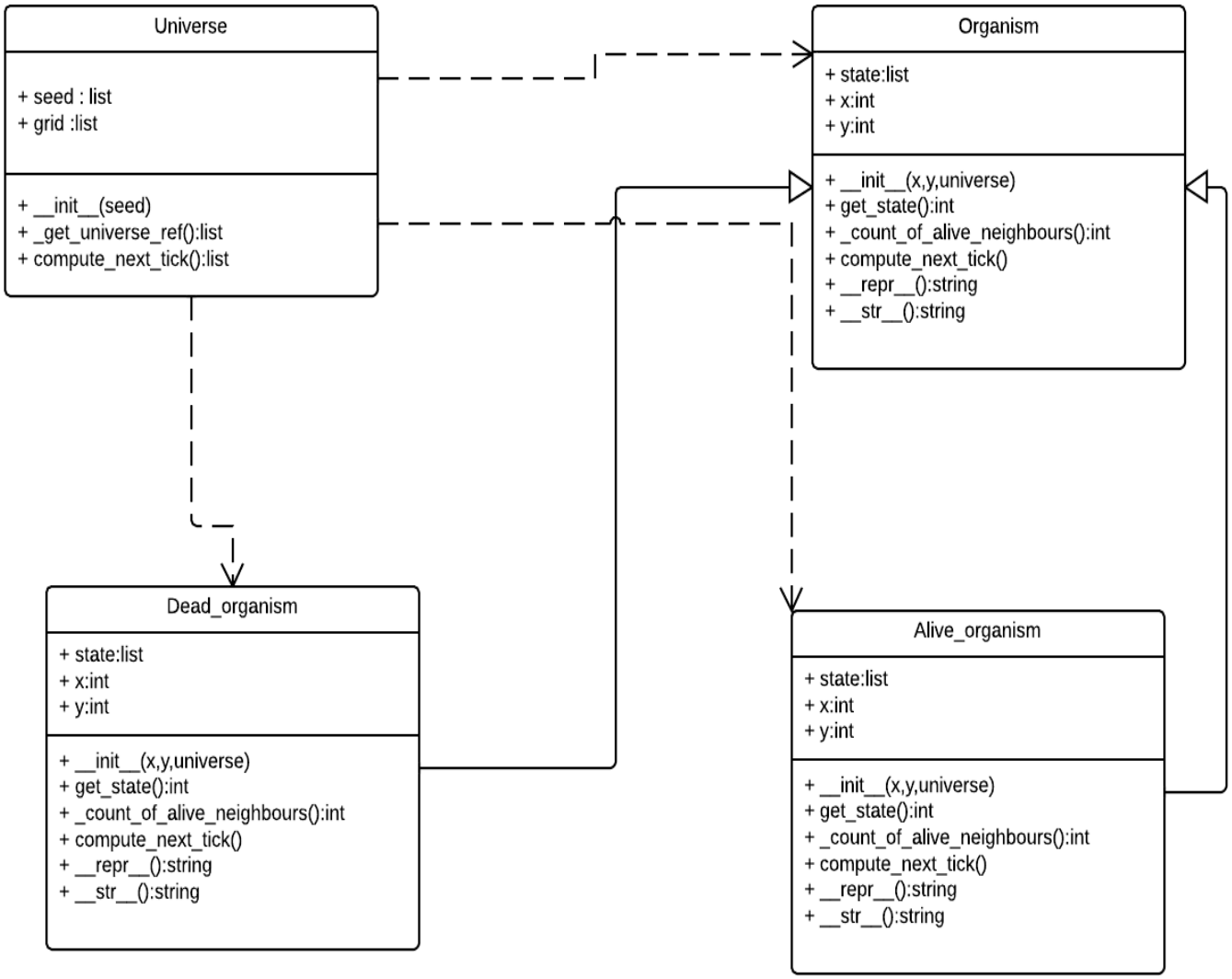
Solution

Design

The pattern used in this specific solution is “State design pattern”, salient features of the solution:

1. Object oriented state machine
2. Extensible state transitions, so this code could be, with very little change, used to implement any other cellular automaton.
3. Easy to understand

Code defines “Universe” class as a external interface and it handles all the operations .



Tests:

Code comes included with following unit tests

1. Test case to check if we are able to correctly change state alive-->dead
2. Test case to check if we can get alive neighbours correctly for Single dimension Universe -XX
3. Test case to check if we can get alive neighbours correctly for Double dimension Universe XXX , XXX
4. Test case to check if we can get alive neighbours correctly for Double dimension Universe XXX , -X-
5. Test case to check if we can get alive neighbours correctly for Three dimensional Universe -X- , -X-, -X-
6. Test output with a input X X , X X --> BLOCK Pattern
7. Test output with a input X X - , X - X , - X - --> BOAT PATTERN
8. Test output with a input - X X X, X X X - --> TOAD PATTERN

to execute tests , run following commands

python tests.py