Assignment 4, Part 1, Specification

SFWR ENG 2AA4

April 11, 2019

This specification was created for John Conway's Game of Life. The user interface consists of an ascii text representation of a grid for the game which is modifiable by the user.

Cell Types Module

Module

CellType

Uses

N/A

Syntax

Exported Constants

N/A

Exported Types

$StateT$ = Dead, Alive
$CellT$ = tuple of (s : StateT)

Exported Access Programs

None

Semantics

State Variables

None

State Invariant

None

Grid2D Module

Template Module

Grid2D

Uses

CellType

Syntax

Exported Types

Grid2D = ?

Exported Access Programs

Routine name	**In**	**Out**	**Exceptions**
Grid2D	fileName:string		invalid_argument
viewGrid	fileName:string		invalid_argument
evolveGrid			
viewGenerations	$\mathbb{N}$		invalid_argument

Semantics

Environment Variables

initialState: File containing the graphical representation of the initial state of the Grid.
nextState: File containing the graphical representation of the state after any iteration of the program.

State Variables

grid: seq of CellT

State Invariant

None

Assumptions

- The Grid2D constructor is called before any other access routine is called on that instance. Once a Grid2D has been created, the constructor will not be called on it again.
- The initial state of the game is initialized through a text file and the user can manually change each point on the grid. To stay true to the view portion of the Model View Controller design pattern, the representation of the grid in the text file is a graphical one where a period "." represents a dead cell and a capital O "O" represents a a cell that is alive on the grid. The number or rows and columns can be changed as well.
- Cells which are at the border of the grid do not have the number of their neighbours counted, that is, they are ignored.

Access Routine Semantics

Grid2D($fileName$):

- transition: Reads the state of the grid from the environment variable initialState and use the graphical representation of the initial state of the grid to create a grid consisting of CellTypes which are either dead or alive.

  The test file initialState consists of an n x m grid where n and m can be modified. The graphical representation of a dead cell is a period ".", and the graphical representation of a cell which is alive is a capital O "O".
- exception: invalid_argument when there is a problem opening the initialState text file to read.

viewGrid($fileName$):

- transition: Writes to the environment variable nextState the graphical representation of the grid of CellTypes . Translates the grid consisting of CellTypes into its graphical representation.

  The text file nextState has the same format as initialState.
- exception: invalid_argument when there is a problem opening the nextState text file to write.

evolveGrid():

- transition:
  $\forall$i, j: $\mathbb{N}$ | i $\in$ [0..$grid.size()$], j $\in$ [0..$grid[i].size()$]: findNextState(grid[i][j])

  grid = temporaryStorage

- exception: None

viewGenerations(int n):

- transition #procedural specification

```
for i in range(n)
        viewGrid()
        evolveGrid()
```

- exceptions: None

Local Functions

countNeighbours(int row, int col):

- output: #procedural specification

```
sum = 0
for i in range(-1..2)
     for j in range(-1..2)
          if(row+ i > 1 and row + i < grid.size() and col + j > -1 and col + j < grid[0].size())
                sum += grid[row + i][col + j]
sum -= grid[row][col]
return sum
```

- exceptions: None

findNextState(int i, int j):

- transition:

Grid[i][j].s	**countNeighbours(i, j)**	**temporaryStorage[i][j]**
Dead	3	CellT.s = Alive
Alive	$\leq$ 3	CellT.s = Alive
Alive	$>$ 3	CellT.s = Dead
Alive	$<$ 2	CellT.s = Dead
Dead	! 3	CellT.s = Dead

Local Types

temporaryStorage: seq of CellT

Critique of design

I tried to stay as true to the model view controller design pattern when creating my design of the program as possible. Instead of creating a module which lets the user determine which cells should be alive and which cells should be dead through a program call, I decided to let the user modify the cells using by changing the text file themselves. I also decided to make the format of the text file containing the intial and final state of the grid to both bt a graphical representation of the grid as close to the one on https://bitstorm.org/gameoflife/ as I could make it using ascii characters. I kept the amount of access programs as small as possible and made sure that each access program is essential. That being said, viewGenerations is not essential since it can be done by a series of evolveGrid and viewGrid, but I included as a convenience to the user. In turn this also increases the principle of separation of concerns as the user does not have to worry about evolving the grid and viewing the grid at each stage of the game. The Grid2D ADT is very general since it is the only module I needed to create he game. I felt that another module to write to the file is not necessary but it would increase separation of concerns so I believe I would do that if I were to redesign my program. I also believe that the access routines in Grid2D are not as minimal as I could make them, but are minimal in the sense they do only one job (with the exception of viewGenerations which evolves the grid and calls viewGrid).