Conway's Game of Life in Second Life - Technical Details

Joel Dearden

Centre for Advanced Spatial Analysis, University College London

Overview

This document is a collection of rough notes about how CASA's version of the Game of Life in Second Life works.

The notes were made as part of the development process and details were inevitably changed during implementation and bug fixing so there may be some differences between this document and the real code.

Agent Rules

From Wikipedia (http://en.wikipedia.org/wiki/Conway's_Game_of_Life):

The universe of the Game of Life is an infinite two-dimensional orthogonal grid of square cells, each of which is in one of two possible states, live or dead. Every cell interacts with its eight neighbours, which are the cells that are directly horizontally, vertically, or diagonally adjacent. At each step in time, the following transitions occur:

1. *Any live cell with fewer than two live neighbours dies, as if by loneliness.*
2. *Any live cell with more than three live neighbours dies, as if by overcrowding.*
3. *Any live cell with two or three live neighbours lives, unchanged, to the next generation.*
4. *Any dead cell with exactly three live neighbours comes to life.*

The initial pattern constitutes the 'seed' of the system. The first generation is created by applying the above rules simultaneously to every cell in the seed — births and deaths happen simultaneously, and the discrete moment at which this happens is sometimes called a tick. (In other words, each generation is a pure function of the one before.) The rules continue to be applied repeatedly to create further generations.

Block Creation and State Display

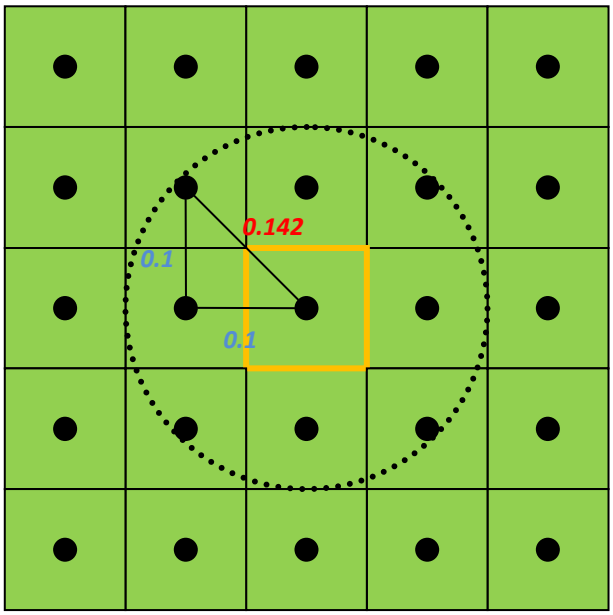
We manually create all the grid blocks initially and change their opacity (via script) to 1 for ALIVE and 0 for DEAD. This is faster than creating, deleting or moving them.

Calculation

Each agent is responsible for calculating its state. We don't use any centralised control except to synchronise the blocks.

Agent Vision

Each agent needs to be able to see the state of the eight agents adjacent to it. We implement this using the llSensor command in LSL. The diagram below illustrates the sensor range required.



Assuming each block is 0.1m square, the sensor range needs to be:

$0.1^2 + 0.1^2 = c^2$

c = 0.142 (rounded up)

Synchronising Block Agents

We use a controller to synchronise the blocks in the grid. Obviously they all need to think and act at the same rate otherwise the grid will not behave as intended.

The sequence for the game of life is:

1. *Block start think – Sync point A*
2. *Block stop think – Sync point B*
3. *Block act*
4. *Repeat*

Sync point A is required to prevent any blocks from thinking too soon, i.e. before other blocks have acted, and so sending the whole grid out of sync.

Sync point B is required to prevent any blocks from acting too soon, i.e. before others have finished thinking and again sending the whole grid out of sync.

How do we synchronise 100 block scripts quickly to perform an action at the same time?

All blocks register with the controller initially so it has a saved list of block keys.

All blocks also save the controller key.

For SyncA:

Blocks check llKey2Name(controllerKey) to see if it is called "think"

The controller only sets its name to "think" once all blocks have set their name to "<alive/dead state>:wt" which indicates that they are waiting to think.

When blocks see controller name as "think" they do their thinking and then when finished set their name to "<state>:wa" to indicate that they are waiting to act.

For SyncB:

Blocks check llKey2Name(controllerKey) to see if it is called "act"

The controller only sets its name to "act" once all blocks have set their name to "<state>:wa" which indicates that they are waiting to act.

When blocks see controller name as "act" they do their acting and then when finished set their name to "<state>:wt" to indicate that they are waiting to think.

Repeat...