Critters with the WBEngine

CBPrice 19-12-19 (Palindrome)

Aims

To explore what animations can be created with a new API, by a discovery-based approach.

Creating assets	
Cell cell1 = new Cell(canvas,"redCell");	The string "redCell" is the image filename as usual.
Bug bug1 = new Bug(canvas,"name", direction);	Prepares a bug to be added, and defines its direction - (use the keywords, up, down, right, left.) Done here so not to create a different "add" statement

Adding assets	
addB(roadTile,X,Y);	These "addBs" (add Blocking) are additional adds for this application. They conform to the "standard " SWC adds.
addB(bug, X,Y);	
addB(cell,X,Y);	
removeB(bug);	Removes the bug from the system
removeB(cell);	Removes the cell from the system

Movement	
bug1.moveB(N);	Bug1 moves N tiles in its current direction. In a while loop situation, N = 1
bug1.rotateThenMove(clockwise/antiClockwise, N);	Either clock or anticlock rotation (90-degrees, followed by an N-tile move. In a while loop situation N=1. This method is needed to get the bug to escape from a tile where it has detected a cell.
bug1.turnAroundB();	About face – rotates 180 degrees
bug1.rotateB(clockwise/antiClockwise);	Probably not needed

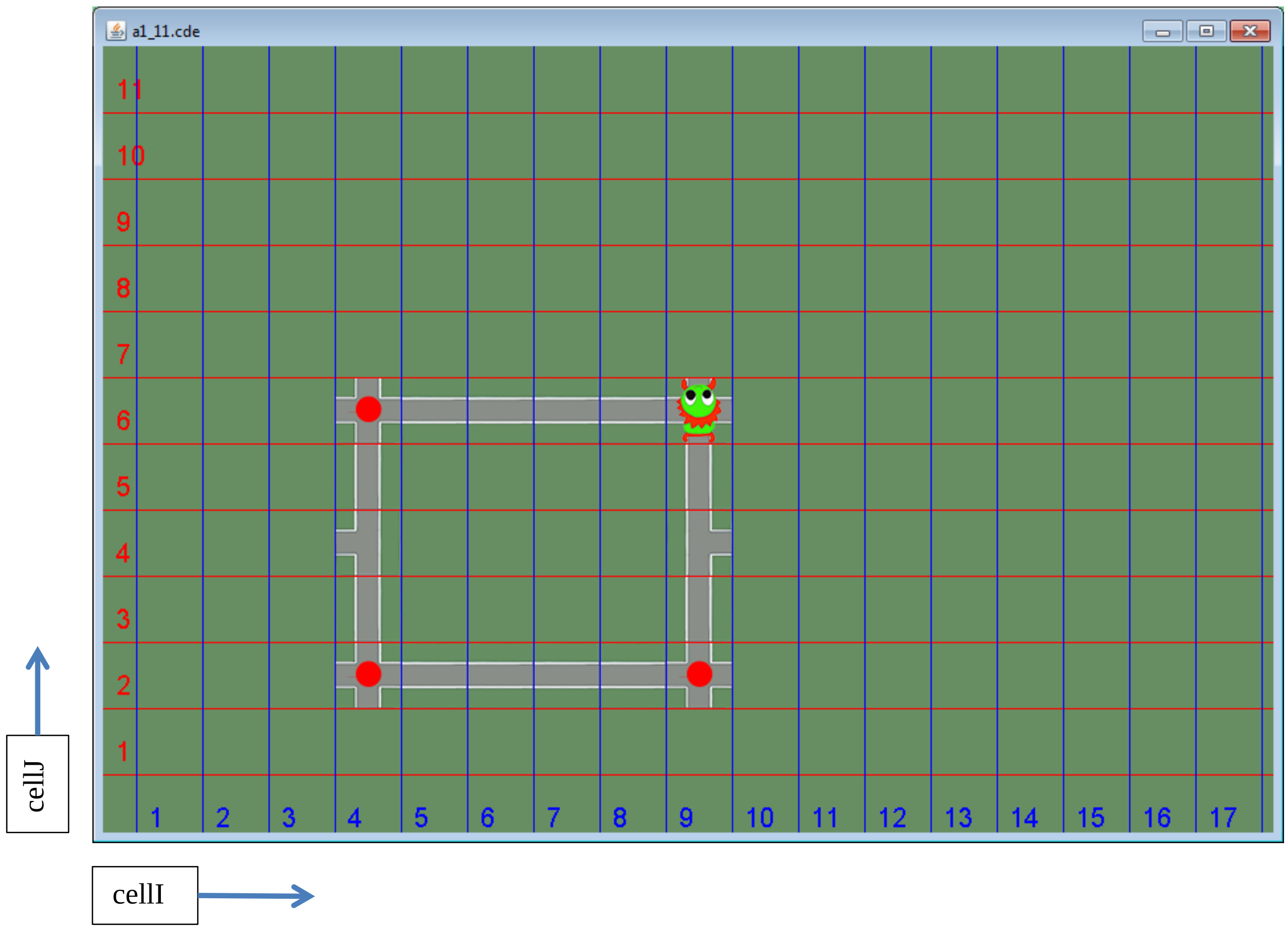
Getting underlying cell and bug information	
String cellName = bug1.getCell();	This sees if there is any cell underneath the bug's (i,j) lattice position. The cellName is the original filename
String cellName = bug1.getCellAhead();	Returns the name of any cell at the next tile to be visited by the bug
Bug bg = bug1.getBug();	Returns the bug at the cell bug1 is on, else null if there is no bug there
Bug bg = bug1.getBugAhead();	Returns the bug at the next tile to be visited by the bug

i = bug1.getCellI();
j = bug1.getCellJ();
	Returns the i and j location of the bug

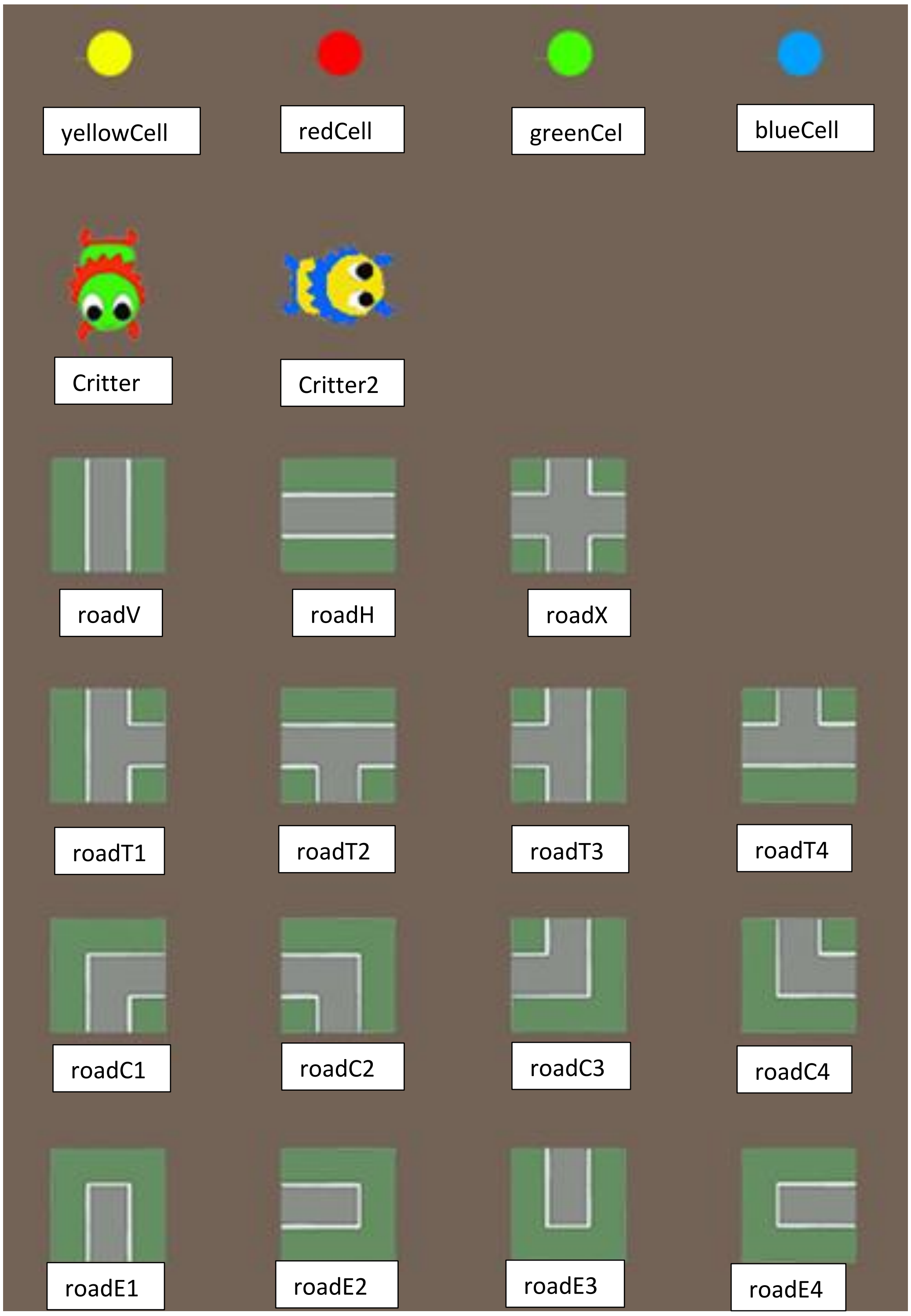
i = cell1.getCellI();
j = cell1.getCellJ();
	Returns the i and j location of the cell (not really useful)

Debugging – Outputting information	
consoleOut("Some text "+ variableName);	Outputs to the engine console. Scrolls
bug1.tellsB("Some text "+ variableName);	Outputs to the canvas, for a short time.
bug1.tells("Some text "+ variableName);	Outputs to the canvas, permanent, over-writes.

Setting the Bug's behaviour	
bug1.setExecTime(float);	Sets time for each action. Default is 1.0F secs.


a1_11.cde
11
10
9
8
7
6
5
4
3
2
1
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
cellJ
cellI

Assets



Example Code

a1_assets.cde	Code which will replicate the above figure (without labels)
a1_11.cde	How to get the critter moving around a square.
a1_6.cde	Code to show how to make a random turn at a “yellowCell”
a1_9c.cde	How to handle collisions: bug-bug and bug-cell.

Coding

Here’s the basic code layout

```
// ==========================================
// Program title, author and date
// ==========================================

// Declarations
Bug bug1, bug2;
Cell cell1, cell2;
int i;

public void once() {
    showGridB();
    setScene(“TurtleBackground”);
    // Create instances of any assets
    bug1 = new Bug(canvas,”Critter”,up);  // Initial direction of the bug. Can be up, down, left, right
    cell1 = new Cell(canvas,”yellowCell”);
    // Add tiles and assets to the map
    addB(roadX,3,3);
    …

    // Run in a continuous loop (set “someNumber” quite small when developing/debugging.
    i = 0;
    while(i < someNumber) {

        // Do tests for tile occupants first

        // Make the bugs move 1 tile
        bug1.moveB(1);
        bug2.moveB(1);

        // update loop counter
        i++;
    } // end while(…)
} // end once()
```