Reference Sheet

Watershed

Watershed contains all the Tiles of the game map.

`WS.getAllTiles()`
Returns an array containing all the Tiles in the game map.

`WS.update()`
Updates all the land uses of the game map to have the current pollution values. Also computes and updates the total pollution in the watershed using current pollution values.

`WS.totalDecayPollution`
The total amount of pollution that reaches the river after distance decay.

Tile

A Tile is one square on the game map. It contains the tiles position on the game grid and its land use.

`t.getX()`
Returns the x coordinate of `t` on the canvas.

`t.getY()`
Returns the y coordinate of `t` on the canvas.

item`t.getDecayPollution()`
Returns the pollution that reaches the river after distance decay of `t`.

Landuse

Each tile on the game map has one of six landuses: Factory, Farm, House, Forest, Dirt or River.

`lu.getType()`
Returns the land use of `lu`. There are six possible values:

1. `LUType.FACTORY`
2. `LUType.FARM`
3. `LUType.HOUSE`
4. `LUType.FOREST`
5. `LUType.DIRT`
6. `LUType.RIVER`

Conditionals and Return

`if(b) { ... } else { ... }`
Conditionally executes the first block of code if `b` evaluates to `true`, and the second block of code if it evaluates to `false`.

`if( (a)&&(b) ) { ... } else { ... }`
Conditionally executes the first block of code if both `a` and `b` evaluates to `true`, and the second block of code if either evaluates to `false`.

`return b`
Returns `b` as the value of the current method.

Assignment and Arithmetic

`x = n`
Assigns variable `x` to the value `v`.

`n1 + n2`
Adds `n1` to `n2`.

`n1 - n2`
Subtracts `n1` from `n2`.

`n1 * n2`
Multiplies `n1` by `n2`.

`n1 / n2`
Divides `n1` by `n2` (with no remainder).

`n1 > n2`
Returns `true` if `n1` is strictly greater than `n2`. Otherwise returns `false`.

`n1 < n2`
Returns `true` if `n1` is strictly less than `n2`.Otherwise returns `false`.

Arrays

`new A[n]`
Creates an new array objects of type `A` of length `n`.

`A[] = { v1, ... vk };`
Creates a new array initialized with elements `v1` through `vk`.

`a.length`
The length of array `a`.

`a[i]`
The i^{th} element of array `a`.

`a[i] = v`
Stores `v` as the i^{th} element of array `a`.

`for(int i = 0; i < a.length; i++) { ... }`
Iterates through the elements of array `a`. In each iteration of the loop, the element can be accessed using index `i`.

Loops

`if(b) { ... } else { ... }`
Conditionally executes the first block of code if `b` evaluates to `true`, and the second block of code if it evaluates to `false`.

`if( (a)&&(b) ) { ... } else { ... }`
Conditionally executes the first block of code if both `a` and `b` evaluates to `true`, and the second block of code if either evaluates to `false`.

`return b`
Returns `b` as the value of the current method.

Files

`Drops`

The "main" file. Your code goes here.

`Constants`

Defines constants that determine the basic parameters of the game.

`Globals`

Declares global variables that store the game state.

`Draw`

Defines the main `draw` method, which is invoked every time the screen refreshes.

`Drop`

Defines the `Drop` data structure.

`Setup`

Defines the main `setup` method, which is invoked once when the application begins.

Advanced Graphics

`background(c)`

Sets the background color to `c`. There are many ways to describe colors:

- `n`: a single number between `0` (black) and `255` (white) describes a shade of grey.
- `(r,g,b)`: a triple of numbers each ranging from `0` to `255`, describe a color in terms of the relative amounts of red, green, and blue.

`fill(c)`

Sets the current fill color to `c`.

`ellipse(x,y,w,h)`

Draws an ellipse at coordinates `x` and `y` with witdth `w` and height `h`.

`rect(x,y,w,h)`

Draws a rectangle at coordinates `x` and `y` with witdth `w` and height `h`.

`mouseX`

The current x coordinate of the mouse pointer.

`mouseY`

The current y coordinate of the mouse pointer.

`mouseClicked()`

Invoked each time the mouse button is clicked.

`keyPressed()`

Invoked each time any key is pressed. The key pressed can be accessed from `key`.