Slither.io-bot Documentation

Release 0.9.0

FliiFe

January 06, 2017

Contents

Contents:

CHAPTER 1

1. Game variables

Note: All the variable names (currently) mean something. For example, `gsc` means general scale, `playing` is literal, etc...

Warning: All the variable below are global, and should be accessed using `window.<variable name>`

1.1 1.1 Scale variables

1.1.1 1.1.1 `gsc` - Number

The general scale variable. It is used to zoom in and out of the game, and to scale the game according to the player's size. It defaults at 0.9.

1.1.2 1.1.2 `csc` - Number

This is the scale variable used by the canvas, to convert screen coordinates to canvas ones. It doesn't have any default value.

1.2 1.2 State variables

1.2.1 1.2.1 `playing` - Boolean

This variable tells wether the game has started or not.

1.3 1.3 Game elements, content variables

The title may not be clear, but these are variables containing a list of snakes, etc..., with game unit.

Game unit One of the units used by the bot. Please refer to *2.2 Different units*.

1.3.1 1.3.1 `snake` - Object

This is the object corresponding to the player's snake. It contains a lot of useful informations (angle, coordinates, length, width, etc...).

Note: Its value is `undefined` if the game did not start yet.

1.3.1.1 `snake.xx` and `snake.yy` - Number

They are the coordinates of the snake, in Game unit.

1.3.1.2 `snake.ang` - Number

This is the angle, *in radians*, at which the snake is going. It goes from 0 to 2π.

1.3.1.3 `snake.nk` - String

The nickname you choose on login screen. In case you forget...

1.3.1.4 `snake.alive_amt` - Number

Reflects loading of snake when the game begins. Range from 0 to 1. Always 1 in game play.

1.3.1.5 `snake.sc` - Number

This is the snake scale, determining how big the snake is drawn.

1.3.2 1.3.2 `snakes` - Array of objects

This is an array containing nearby snakes, including yours. This means you cannot access all snakes to base a strategy upon. The content of each object is almost the same as in the `snake` object. It may contain null or undefined objects.

Note: `snakes[index].pts` contains every components of the snake.

1.3.3 1.3.3 `foods` - Array of objects

An array containing the foods objects. each object has multiple keys:

1.3.3.1 `foods[index].xx` and `foods[index].yy` - Number

In game unit, position of each food coordinate.

index Arbitrary or generated number to access the (n+1)th element of an array. For example, an index of 5 would access the sixth element of an array.

1.3.3.2 `foods[index].sz` - Number

Size of the food. Each food has a different size. This value can be used to sort foods according to their size.

1.3.4 1.3.4 `preys` - Array of objects

Contains every objects corresponding to a prey. Every object contains the coordinate of the prey under the keys `xx` and `yy`.

preys Moving foods, that are ways better and make you grow much more.

1.4 1.4 Visual variables

Variables containing visual things.

1.4.1 1.4.1 `mc` - canvas

This is the game canvas. It is not a modified DOM object, it is a classic canvas.

Note: You can get the canvas' context with `mc.getContext('2d');`

1.4.2 1.4.2 `ww` and `hh` - Number

They are the window size. They are dynamic read-only values.

1.4.3 1.4.3 `xm` and `ym` - Number

They stand for 'x mouse' and 'y mouse' respectively. They are used to control the snake.

Note: It should be in mouse coordinates. Please refer to *2.2 Different units*

1.4.4 1.4.3 `window.sector_size` - Number

This is the size of your snake vision (scope) in the game map. Used with `bot.sectorBox`.

1.5 1.5 Game functions

1.5.1 1.5.1 `document.onkeydown`

This is a listener to keyboard press. Upon running the bot.user.js script, this funciton is saved as `userInterface.original_keydown` and then overridden to add more functionalities.

1.5.2 1.5.2 `window.onmousedown`

This is a listener to mouse press. Upon running the bot.user.js script, this funciton is saved as `userInterface.original_onmousedown` and then overridden to add more functionalities.

1.5.3 1.5.3 `window.onmousemove`

This is a listener to mouse cursor movement. Upon running the bot.user.js script, this funciton is saved as `userInterface.original_onmousemove` and then overridden to add more functionalities.

1.5.4 1.5.4 `window.setAcceleration(x)`

This is the function used to accelerate your snake. When called with argument '1' snake will accelerate.

CHAPTER 2

2. Different units, and coordinates systems

Currently, the bot uses multiple coordinates system, and multiple units for each system.

2.1 2.1 Coordinate systems

The two systems used rely on an orthogonal set. `x` represents the horizontal axis, and `y` the vertical axis. Let's consider this configuration (Conf. 1) :

2.1.1 2.1.1 Cartesian coordinates

These are the most used. As in traditional maths, they are represented by to numbers, each representing the position on an axis. They are represented as `(x, y)`. `x` for the horizontal axis, and `y` for the vertical one.

In Conf 1, we have the following coordinates :

- `A(2, 3)`
- `B(-3, 1)`

2.1.2 2.1.2 Polar coordinates

Less used (and also less popular), these are used to avoid things inside certain angles. They are represented as a number and an angle. They look like this : (r, θ), where r is the distance from the origin, and θ is the angle, in radians, at which the point is, starting at the horizontal axis.

Important notes :

- In the game, the angles are reversed, they go clockwise. *i.e* positive angles are under the `x` axis, and negative ones are above.
- We only use angles up to π radians (180). That means we use negative angles to access the other side of a circle.

The coordinate system looks like this :

In Conf 1, we have the following coordinates :

- `A(3.60555, 56.30993°)`
- `B(3.16228, 161.56505°)`

The above coordinates are approximative.

In the bot, every polar coordinate is converted from cartesian coordinates. Also, we don't use the r value, only the angle is important. That helps reducing computing expensiveness. Please refer to the conversion section below.

2.2 2.2 Different units

There are three main units used for the game. Some of them subdivide into others, but overall, we have three units.

2.2.1 2.2.1 Screen unit

This is the unit used by the game for mouse input. Basically, they are cartesian coordinates, relative to the browther window. The origin can either be at the top left corner, starting at $(0, 0)$, or the center of the window.

Note: If the origin is the center, then the unit is called **Mouse unit**.

2.2.2 2.2.2 Canvas unit

This is the unit used by the canvas to draw every elements. As before, cartesian coordinates, origin is the top left corner or the canvas center, but the canvas element doesn't start at screen unit $(0, 0)$. Instead, there is an offset relative to window size. Also, the canvas size has not much to do with window size.

On initialization, an array containing the ratio between the canvas size and the window size is created. This ratio is stored inside `canvasRatio[]`, which is defined as follows

```
var canvasRatio = [window.mc.width / window.ww, window.mc.height /
    window.hh
]
```

As you can see, the ratio is the width/height of the canvas (the canvas element is `window.mc`) divided by the width/height of the screen (`window.ww` or `window.hh`).

2.2.3 2.2.3 Game/Map unit

This is the unit used by the game, and which is present in every objects. It's origin is the top left corner, or, as the previous units, the snake's head position (center of the screen).

2.3 2.3 Conversion

Every part will have a title in the form `Starting unit -> Result unit`

2.3.1 2.3.1 Screen unit -> Canvas unit

The function used for this conversion is this one:

```
// Convert screen coordinates to canvas coordinates.
screenToCanvas: function(point) {
   var canvasX = window.csc * (point.x * canvas.canvasRatio[0]) - parseInt(window.mc.style.left);
   var canvasY = window.csc * (point.y * canvas.canvasRatio[1]) - parseInt(window.mc.style.top);
   return {
       x: canvasX,
       y: canvasY
   };
}
```

It takes a object as argument, which contain a value `x` and a value `y`, given in screen unit. Both are positive integers. The object can be defined as follows:

```
var point = {
   x: 100,
   y: 200
}
```

The conversion process may seem complicated, but it is quite simple. Firstly, we multiply the point coordinates and the ratio defined at initialization. (Cf. *2.2.2 Canvas unit*):

```
point.x * canvas.canvasRatio[0]
```

Then we multiply the result by the canvas scale variable window.csc:

```
window.csc * (point.x * canvas.canvasRatio[0])
```

But, as stated in *2.2.2 Canvas unit*, there is an offset between the canvas and the screen. We solve this issue by taking it away from the result. However, this offset is defined as a String, thus we need to convert it to an integer. The final result looks like:

```
window.csc * (point.x * canvas.canvasRatio[0]) - parseInt(window.mc.style.left);
```

2.3.2 2.3.2 Mouse unit -> Screen unit

This is the function used for conversion:

```
// Convert snake-relative coordinates to absolute screen coordinates.
mouseToScreen: function(point) {
    var screenX = point.x + (window.ww / 2);
    var screenY = point.y + (window.hh / 2);
    return {
        x: screenX,
        y: screenY
    };
}
```

It takes a object as argument, which contain a value `x` and a value `y`, given in mouse unit. As for *2.3.1 Screen unit -> Canvas unit*, the object can be defined as follows:

```
var point = {
   x: -100,
   y: 150
}
```

Note: `x` and `y` can be negative, if they are on the left part of the screen, because the origin is the center.

The conversion algorithm is simple, take the mouse coordinates, and add the coordinates of the screen's center.

2.3.3 2.3.3 Game unit -> Mouse unit

The map unit -> mouse unit is made using this function:

```
// Convert map coordinates to mouse coordinates.
mapToMouse: function(point) {
    var mouseX = (point.x - window.snake.xx) * window.gsc;
    var mouseY = (point.y - window.snake.yy) * window.gsc;
    return {
        x: mouseX,
        y: mouseY
    };
},
```

Both game unit and mouse unit share the same norm, thus don't need any coordinates scale. Only the general scale variable window.gsc is needed to apply zoom.

The game unit is just mouse unit shifted. The origin is the top left corner (of the map), so we just need to take the player's coordinates away from the given coordinates. Then we multiply by the general scale.

2.3.4 2.3.4 Game unit -> Canvas unit

This function is just a shorthand, using all of the above coordinates conversion, respectively `Game unit -> Mouse unit`, `Mouse unit -> Screen unit`, and `Screen unit -> Canvas unit`

```
// Map cordinates to Canvas cordinate shortcut
mapToCanvas: function(point) {
    var c = canvas.mapToMouse(point);
    c = canvas.mouseToScreen(c);
    c = canvas.screenToCanvas(c);
    return c;
}
```

CHAPTER 3

3. Bot variables

3.1 3.1 Preferences

3.2 3.2 Visual variables

3.2.1 3.2.1 `bot.sectorBox` - Rectangle

A rectangle reflecting the 'vision' of the bot.

3.3 3.3 Bot functions

CHAPTER 4

4. Bot mechanics

4.1 4.1 “Classic” version

4.2 4.2 A* Path finding algorithm

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

G

I

P