Implementación en JavaScript para el modelado y desarrollo de sistemas basados en agentes

Gastón H. Salazar-Silva y Abraham Rodríguez Galeotte

Instituto Politécnico Nacional, UPIITA. Av. IPN 2580, Col. La Laguna Ticoman, México, CDMX.

Abstract

Un agente autónomo es un modelo computacional que permite describir comportamientos muy complejos dentro de un entorno. Estos comportamientos se producen a partir de funciones relativamente simples que tienen como dominio una secuencia, o historia, de las percepciones del entorno. El modelado basado en agentes se aplica en inteligencia artificial, robótica, protocolos de comunicación, economía, sociología y aplicaciones web, entre mucho más. El presente trabajo muestra una librería que permite implementar un sistema de agentes autónomos, de una manera simple, dentro de un entorno gráfico que se ejecuta en un navegador web. Esta librería también puede operar en un sistema embebido. Cómo ejemplos se muestran una simulación de la navegación de múltiples robots en un entorno desconocido, así como la implementación de un robot mensajero.

1. Introduction

Un agente autónomo es un modelo computacional que permite describir comportamientos muy complejos dentro de un entorno. Estos comportamientos se producen a partir de funciones relativamente simples que tienen como dominio una secuencia, o historia, de las percepciones del entorno. El modelado basado en agentes se aplica en inteligencia artificial, robótica, protocolos de comunicación, economía, sociología y aplicaciones web, entre mucho más.

Literatura

El presente trabajo muestra una librería que permite implementar un sistema de agentes autónomos, de una manera simple, dentro de un entorno gráfico que se ejecuta en un navegador web. Esta librería también puede operar en un sistema embebido. Cómo ejemplos se muestran una simulación de la navegación de múltiples robots en un entorno desconocido, así como la implementación de un robot mensajero.

2. Metdodología

Un agente autónomo es sistema capaz de interactuar con su entorno, y por ello de disponer de una interfaz con su entorno. Ésta se puede componer de sensores y accionadores (efectores) para el caso de un sistema físico. Por otro lado, un agente debe de actuar de forma autónoma, es decir debe capaz de tomar de decisiones de forma independiente a partir de las perceciones que realiza sobre su entorno. Además, en un agente el comportamiento debe ser teleológico, es decir está orientado a alcanzar un objetivo. Un agente puede responder que no, porque contraviene a su objetivo.

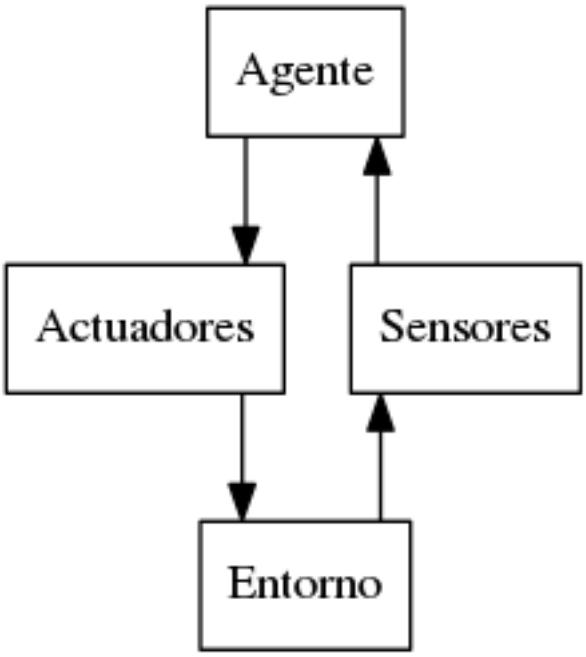

Figure 1: Agente

Un agente se puede describir como una función f,

$$f : p^* \mapsto a$$

donde p denota la percepción realizada por el agente en momento actual, p^* representa una secuencia de percepciones realizada a lo largo del tiempo, es decir una historia de las percepciones, y a representa una acción a desempeñar por el agente. Las variables a y p pueden ser escalares, vectores o incluso funciones.

2.1 Un prototipo de agente en JavaScript

Para poder implementar los agentes, se deben poder desarrollar abstractamente los sensores, el planificador y los accionadores. En el caso de los sensores, deben ser capaces de detectar la información del entorno enque se encuentran y reaccionar a eventos en el mismo. Los accionadores deben ser capaz de afectar al entorno. Finalmente se debe poder representar el entorno para la planificación; para esto último se aprovecha la librería THREE.js, que no solo permite representar este entorno, sino también visualizarlo en 3D sobre un navegador *web*.

Para empezar, se define la función Agent(), que es el constructor de un agente abstracto. Un agente instanciado a partir de este constructor no tendrá realmente funcionalidad pero le permite tener una interfaz. Como se desea tener la capacidad de poder ser visualuizado en 3D, se utiliza como prototipo el constructor Object3D de la librería THREE.js.

```
function Agent(x=0, y=0){
    THREE.Object3D.call(this);
    this.position.x = x;
    this.position.y = y;
}

Agent.prototype = new THREE.Object3D();
```

Las tres primitivas esenciales de un agente son percibir (sense), planificar (plan) y actuar (act) sobre el entorno (environment) sobre el cual el agente esta operando. Como el constructor es para un objeto abstracto, estas primitivas se definen sin tener una implementación específica.

```
Agent.prototype.sense =
    function(environment) {};
Agent.prototype.plan =
    function(environment) {};
Agent.prototype.act =
    function(environment) {};
```

Estos métodos se implementan de manera abstracta porque existe un sinnúmero de formas de implementar estos métodos. Para una implementación específica, el comportamiento se establecerá redefiniendo estos tres métodos.

Como un agente opera sobre un entorno, se define también un constructor para los entornos. Este contructor se define como una instancia de Scene(), que es básicamente una colección de objetos de la clase Object3D(). Esto permite que los agentes puedan ser visualizados directamente en el navegador.

```
function Environment() {
    THREE.Scene.call(this);
```

```
}

Environment.prototype = new THREE.Scene();
```

Después, se implementan tres métodos para las instancias de Environment(). En esta primera implementación se consideró un modelo síncrono de operación. Por ello, estos métodos tienen la función de enviar el mensaje a los agentes para que realicen determinadas acciones. Con el método sense(), el entorno envía una solicitud a los agentes para que perciban su entorno. Lo mismo ocurre con los métodos plan() y act().

```
Environment.prototype.sense = function() {
    var c = this.children;
    for ( var i = 0; i < c.length; i++ )
        if (c[i].sense !== undefined)
            c[i].sense(this);
}

Environment.prototype.plan = function() {
    var c = this.children;
    for ( var i = 0; i < c.length; i++ )
        if (c[i].plan !== undefined)
            c[i].plan(this);
}

Environment.prototype.act = function() {
    var c = this.children;
    for ( var i = 0; i < c.length; i++ )
        if (c[i].act !== undefined)
            c[i].act(this);
}
```

Como el entorno es una instancia de Scene(), no todos los objetos que pueden exsitir dentro de éste son agentes. por lo tanto primero se debe determinar si el objeto contiene los métodos adecuados para ejecutarse. Esto permite que el entorno pueda contener objetos estáticos o de otro tipo, con los cuales los agentes pueden interactuar.

3. Resultados

A continuación se muestran ejemplos

3.1 Pelota rebotando

El Ejemplo 1 es una pelota que está rebotando entre dos paredes.

Primeramente se define el constructor.

```
function Pelota(r, x=0, y=0) {
    Agent.call(this,x,y);
    this.add(new THREE.Mesh(
        new THREE.SphereGeometry( r ),
        new THREE.MeshNormalMaterial()));
    this.step = 0.1;
    this.colision = 0;
    this.radius = r;
    this.sensor =
        new THREE.Raycaster(
            this.position,
            new THREE.Vector3(1,0,0));
}
```

El prototipo de una pelota es un agente.

```
Pelota.prototype = new Agent();
```

Se redefinen los métodos sense() y act() del agente Pelota. Una pelota no planifica, así que no se redefine plan().

```
Pelota.prototype.sense =
    function(environment) {
        this.sensor.set(
            this.position,
            new THREE.Vector3( 1,0,0) );
        var obstaculo1 =
            this.sensor.intersectObjects(
                environment.children,
                true);
        this.sensor.set(
            this.position,
            new THREE.Vector3(-1,0,0));
        var obstaculo2 =
            this.sensor.intersectObjects(
                environment.children,
                true);
        if ((obstaculo1.length > 0 &&
            (obstaculo1[0].distance <= this.radius)) ||
            (obstaculo2.length > 0 &&
            (obstaculo2[0].distance <= this.radius)))
            this.colision = 1;
```

```
        else
            this.colision = 0;
};

Pelota.prototype.act = function(environment) {
    if (this.colision === 1)
        this.step = -this.step;
    this.position.x += this.step;
};
```

Las paredes en las que rebota la pelota no son agentes, por lo tanto pueden ser simplemente del tipo Object3D.

```
function Pared(size, x=0, y=0) {
    THREE.Object3D.call(this,x,y);
    this.add(
        new THREE.Mesh(
            new THREE.BoxGeometry(size,size,size),
            new THREE.MeshNormalMaterial()));
    this.size = size;
    this.position.x = x;
    this.position.y = y;
}

Pared.prototype = new THREE.Object3D();
```

El programa principal queda muy simple.

```
function setup() {
    entorno = new Environment();
    camara = new THREE.PerspectiveCamera();
    camara.position.z = 30;

    entorno.add( new Pared(1,7,0) );
    entorno.add( new Pared(1,-7,0) );
    entorno.add( new Pared(1,7,1) );
    entorno.add( new Pared(1,-7,1) );
    entorno.add( new Pared(1,7,-1) );
    entorno.add( new Pared(1,-7,-1) );
    entorno.add( new Pelota(0.5) );
    entorno.add( camara );

    renderer = new THREE.WebGLRenderer();
    renderer.setSize( window.innerHeight*.95, window.innerHeight*.95 );
    document.body.appendChild( renderer.domElement );
}
```

```
function loop(){
   requestAnimationFrame( loop );

   entorno.sense();
   entorno.plan();
   entorno.act();
   renderer.render( entorno, camara );
}

var entorno, camara, renderer;

setup();
loop();
```

El archivo HTML se enlista a continuación.

```
<!doctype html>
<html>
  <head>
    <title>Ejemplo: Agentes</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <style>
      body { text-align: center; }
    </style>
  </head>
  <body>
    <script src="http://mrdoob.github.com/three.js/build/three.min.js">
    </script>
    <script src="Agents.js">
    </script>
    <script src="ejemplo.1.js">
    </script>
  </body>
</html>
```

3.2 Un sistema de agentes autónomos

El Ejemplo 2 define un entorno con varios robots operando independientemente.

```
function Wall(size, x, y) {
    THREE.Mesh.call(this,
            new THREE.BoxGeometry( size, size, size ),
                    new THREE.MeshNormalMaterial());
    this.size = size;
    this.position.x = x;
```

```
    this.position.y = y;
}

Wall.prototype =  new THREE.Mesh();

Environment.prototype.setMap = function(map) {
    var _offset = Math.floor(map.length/2);

    for ( var i = 0; i < map.length ; i++)
    for ( var j = 0; j < map.length ; j++) {
        if (map[i][j] === "x")
        this.add( new Wall( 1, j -_offset, -(i-_offset) ) );
        else if (map[i][j] === "r")
        this.add( new Robot(0.5,  j -_offset, -(i-_offset) ) );
    }
}

function setup() {
    var mapa = new Array();
    mapa[0]  = "xxxxxxxxxxxxxxxxxxxxxxxx";
    mapa[1]  = "xr               r      x";
    mapa[2]  = "x                       x";
    mapa[3]  = "x                       x";
    mapa[4]  = "x                       x";
    mapa[5]  = "x                       x";
    mapa[6]  = "x                       x";
    mapa[7]  = "x                       x";
    mapa[8]  = "xxxx  xxxxxxxxxxxxxxxxx";
    mapa[9]  = "x                       x";
    mapa[10] = "x     r                 x";
    mapa[11] = "x                       x";
    mapa[12] = "xxxxxxxxxxxxxxxxx  xxxxx";
    mapa[13] = "x                       x";
    mapa[14] = "x                       x";
    mapa[15] = "x                       x";
    mapa[16] = "x                       x";
    mapa[17] = "x                       x";
    mapa[18] = "x                       x";
    mapa[19] = "xxxxxxxxx    xxxxxxxxxxx";
    mapa[20] = "x                       x";
    mapa[21] = "x                       x";
    mapa[22] = "x                       x";
    mapa[23] = "x                       x";
    mapa[24] = "xxxxxxxxxxxxxxxxxxxxxxxx";

    environment = new Environment();
```

```
    environment.setMap(mapa);

    camera = new THREE.PerspectiveCamera();
    camera.position.z = 30;

    renderer = new THREE.WebGLRenderer();
    renderer.setSize( window.innerHeight*.95, window.innerHeight*.95 );
    document.body.appendChild( renderer.domElement );

    environment.add( camera );
}

function loop(){
    requestAnimationFrame( loop );

    environment.sense();
    environment.plan();
    environment.act();

    renderer.render( environment, camera );
}

var environment, camera, renderer;

setup();
loop();

<!doctype html>
<html>
  <head>
    <title>Ejemplo: Robots como agentes</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <style>
      body { text-align: center; }
    </style>
  </head>
  <body>
    <script src="http://mrdoob.github.com/three.js/build/three.min.js">
    </script>
    <script src="Agents.js">
    </script>
    <script src="robots.js">
    </script>
    <script src="ejemplo.2.js">
    </script>
```

```
  </body>
</html>
```

Definición de `Robot()`

Primeramente definir un sensor de colisiones.

```
function Sensor(position, direction) {
    THREE.Raycaster.call(this, position, direction);
    this.colision = false;
}
Sensor.prototype = new THREE.Raycaster();

function Robot(size,x,y) {
    Agent.call(this, x, y);

    this.sensor = new Sensor();
    this.actuator = new THREE.Mesh(
    new THREE.BoxGeometry( size, size, size ),
        new THREE.MeshBasicMaterial({color: '#aa0000'}));
    this.actuator.commands = [];
    this.add(this.actuator);
}

Robot.prototype = new Agent();

Robot.prototype.sense = function(environment) {
    this.sensor.set( this.position,
              new THREE.Vector3( Math.cos(this.rotation.z),
                     Math.sin(this.rotation.z),
                     0 ));
    var obstaculo = this.sensor.intersectObjects(environment.children,
                                                 true);

    if ((obstaculo.length > 0 &&
     (obstaculo[0].distance <= .5)))
        this.sensor.colision = true;
    else
        this.sensor.colision = false;
};

Robot.prototype.plan = function(environment) {
    this.actuator.commands = [];

    if (this.sensor.colision == true)
```

```
    this.actuator.commands.push('rotateCCW');
    else
    this.actuator.commands.push('goStraight');
};

Robot.prototype.act = function(environment) {
    var command = this.actuator.commands.pop();

    if (command === undefined)
    console.log('Undefined command');
    else if (command in this.operations)
    this.operations[command](this);
    else
    console.log('Unknown command');
};
```

Las operaciones posibles con este robot son

- goStraight(),
- rotateCW(), y
- rotateCCW().

```
Robot.prototype.operations = {};

Robot.prototype.operations.goStraight = function(robot, distance) {
    if (distance === undefined)
    distance = .05;
    robot.position.x += distance*Math.cos(robot.rotation.z);
    robot.position.y += distance*Math.sin(robot.rotation.z);
};

Robot.prototype.operations.rotateCW = function(robot, angle) {
    if (angle === undefined)
    angle = -Math.PI/2;
    robot.rotation.z += angle;
};

Robot.prototype.operations.rotateCCW = function(robot, angle) {
    if (angle === undefined)
    angle = Math.PI/2;
    robot.rotation.z += angle;
};
```

Índice de referencias

Niazi, M., & Hussain, A. (2011). Agent-based computing from multi-agent systems to agent-based models: A visual survey. Scientometrics, 89(2), 479-499.

Richmond, P., & Romano, D. (2008). Agent Based GPU, a Real-time 3D Simulation and Interactive Visualisation Framework for Massive Agent Based Modelling on the GPU. Proceedings International Workshop on Super Visualisation. Kos, Grecia, June 7, 2008.