*Robot, Micromouse, para la resolución autonóma de laberintos.*

*Presentado como requisito de la asignatura Microcontroladores.*

*Universidad del Quindío.*

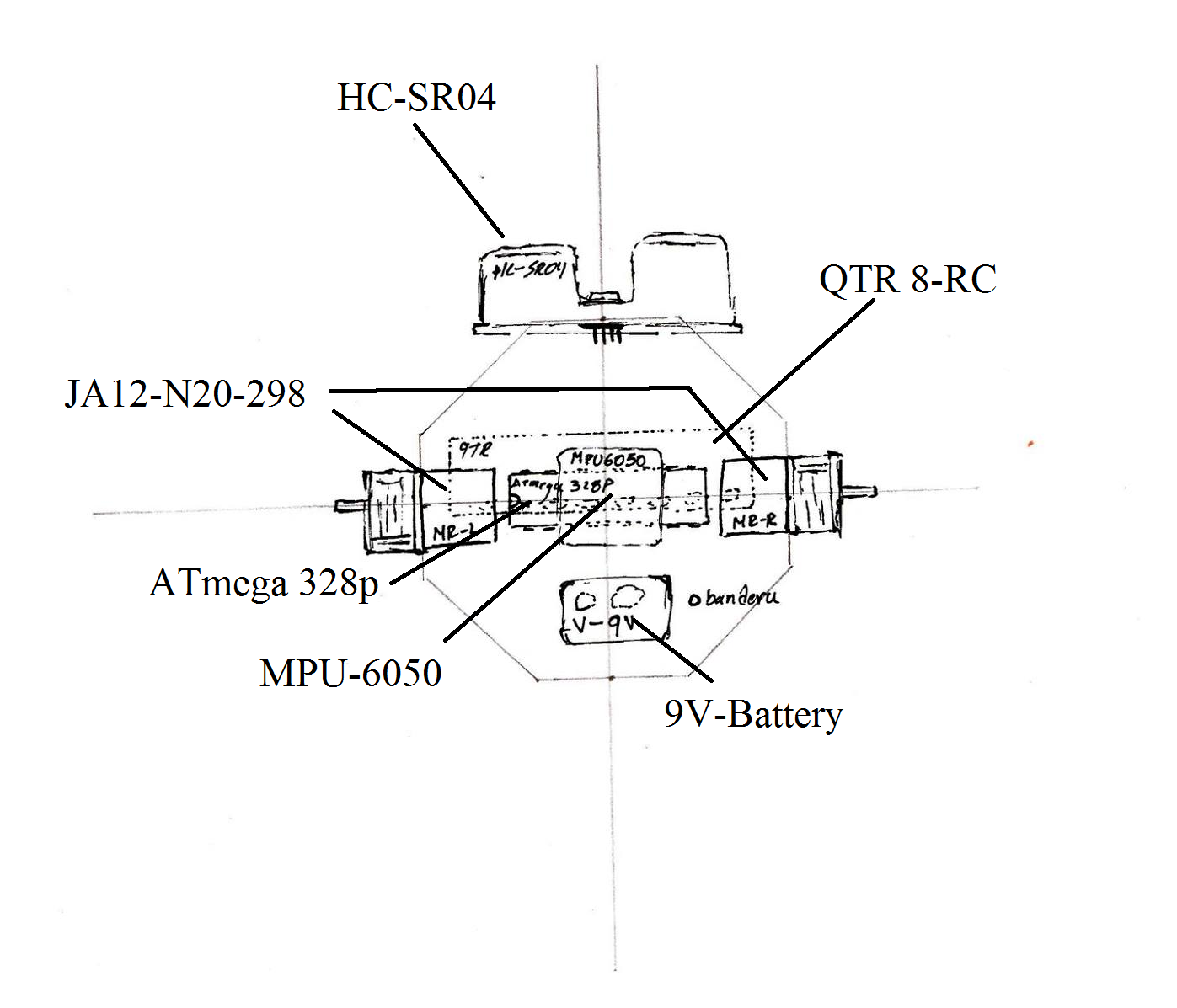
*Programa de Ingeniería electrónica.*

*Ronald Andrés Mora.*

*Nicolás Gutierrez Gonzales.*

*Santiago Restrepo Ruiz.*

*Armenia, mayo de 2016.*

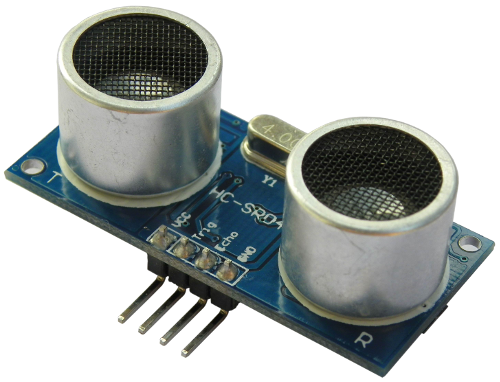


*Figura 1, Planos de diseño.*

# Justificación del diseño:

Con el fin de realizar un sistema más completo, se propone, para este proyecto un modelo con mayor número de componentes, lo que representa también un número mayor de funciones.

Se propone el uso de más sensores, para aprender a manejarlos, estos son:



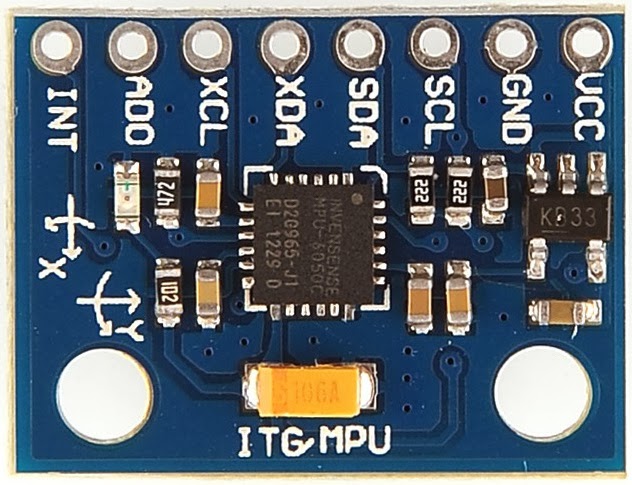
*Figura 2, sensor de ultrasonido HC-SR04.*

**Un sensor de proximidad de ultrasonido**, para esto se usará un módulo propio de *Arduino®*: Los ultrasonidos son señales acústicas cuyas frecuencias están por encima del rango de frecuencias sensibles al oí­do humano. Los sensores de ultrasonidos son capaces de medir la distancia a la que están respecto a un objeto por medio de un sistema de medición de ecos.

Estos sensores está formados por un transductor que emite un pulso corto de energí­a ultrasónica, cuando el pulso es reflejado por un objeto, el sensor captura el eco producido por medio de un receptor, y mediante un sistema de tratamiento de la señal, calcula la distancia a la que está de dicho objeto.

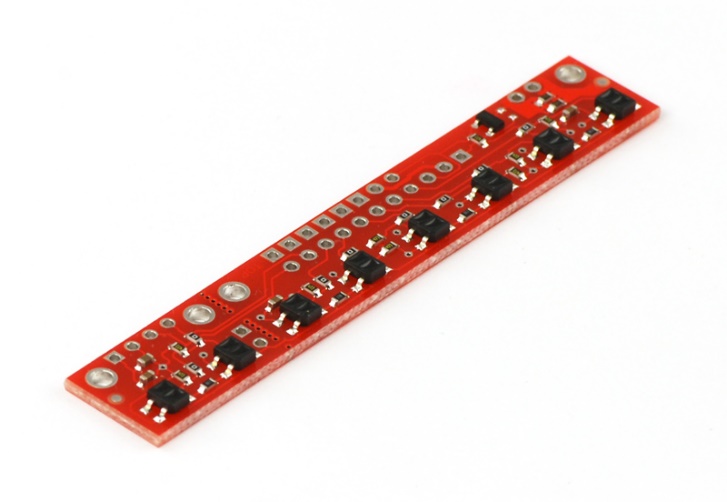
Debido a sus caracterí­sticas, se pueden encontrar sensores de ultrasonidos en aplicaciones como medición de nivel (en tanques que contienen diferentes productos en forma lí­quida), control de colisiones en sistemas de aparcamiento, control de posición en campos como robótica, industria del plástico, control de llenado de tanques.

Las principales ventajas de estos sensores se basan en que no necesitan contacto fí­sico para poder detectar objetos, tienen una buena relación de calidad-precio y en comparación con otras tecnologí­as, los dispositivos basados en ultrasonidos son compactos y livianos. El HC-SR04 está formado por un emisor y un receptor de ultrasonidos que opera a una frecuencia de 40kHz. El sensor se alimenta a 5V, por lo que se puede alimentar directamente desde Arduino, y puede llegar a detectar objetos hasta 5 metros con una resolución de 1cm.



*Figura 3, Giroscopio MPU-6050.*

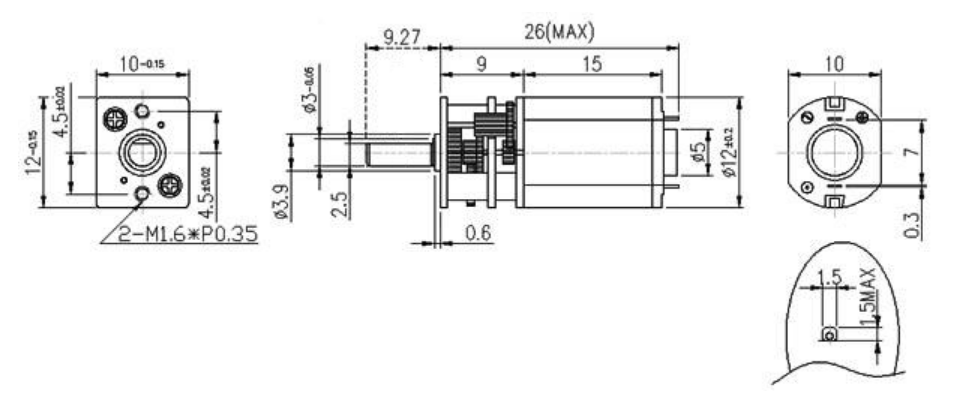
**Giroscopio**, para esto se usará un módulo de posicionamiento geoestacionario angular de *Arduino®*: Este módulo está basado en el sensor MPU6050 y contiene todo lo necesario para dar a los proyectos a realizar la capacidad de medir el movimiento, combina un giroscopio de 3 ejes y un acelerómetro de 3 ejes en un mismo chip, siendo así­ el primer sensor de movimiento que integra estas caracterí­sticas a un bajo precio en el mercado. Integra un DMP (Procesador digital de movimiento) capaz de realizar complejos algoritmos de captura de movimiento de 9 ejes.  Se comunica a través de una interfaz I2C y posee una librerí­a muy difundida para su uso inmediato. Este sensor puede entregar 6 grados de libertad e incorpora un regulador de tensión a 3.3V y resistencias pull-up para su uso directo por I2C. Usando la librerí­a i2cdevlib se facilita el uso para implementar proyectos donde se requiera controlar la inclinación, giro y altitud como podrí­a ser en aeromodelismo. Puede ser usado mediante microcontroladores AVR, ARM y microchip. Su conexión es sencilla, a través de interfaz I2C master, permitiendo así­ controlar sensores externos adicionales como magnetómetros o barómetros, entre otros, sin intervención del procesador principal (economizar recursos). Para un captura precisa de movimiento rápido y lento, posee un rango de escala programable de 250/500/1000/2000 grados/seg para el giroscopio y de 2g/4g/8g/16g para el acelerómetro. Este sensor es usado generalmente para el control de juegos, realidad aumentada, estabilización de imagen electrónica, estabilización de imagen óptica, interfaz de gestos de usuario, circulación de personal peatonal, movimientos gestuales.



*Figura 4, arreglo de 8 sensores infrarrojos QTR-8RC.*

**Sensor infrarrojo,** este es un modelo comercial común de sensores. El arreglo de sensores de reflectancia QTR-8RC está pensado como un sensor de línea, pero puede ser utilizado como una proximidad de propósito general o un sensor de reflectancia. El módulo es un vehículo conveniente para ocho pares IR emisor y receptor (fototransistor) uniformemente espaciados a intervalos de 0,375 "(9,525 mm).

Para este modelo en específico, se usará este sensor con el fin de controlar el rumbo, siguiendo una referencia en el suelo, esto es, una línea negra.



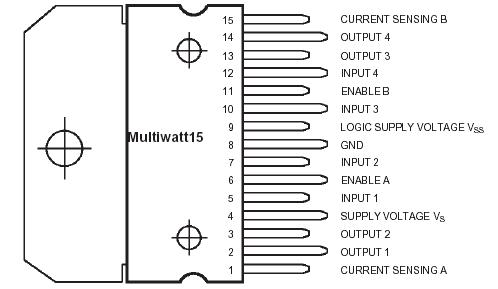
*Figura 5, motorreductor 6V, 30RPM 1.5Kg.cm.*

**Motorreductor,** Este pequeño motor, gracias a sus dimensiones, proporciona una gran facilidad para trabajarlos en el área de la robótica, como en la implementación de carros seguidores, sistemas de pequeñas poleas, entre otros.

Características:

*Tabla 1, características de los motorreductores.*

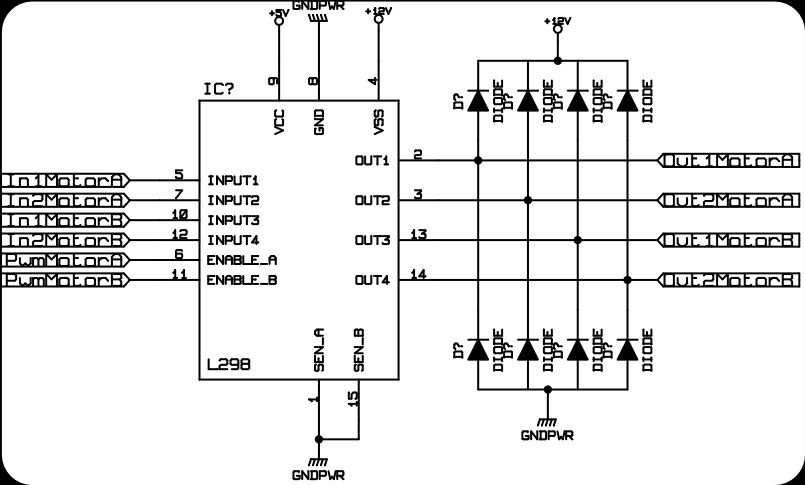
|  |  |
| --- | --- |
| Modelo | JA12-N20-298 |
| Voltaje de operación | 3-9V |
| Velocidad sin carga a 6V | 30 RPM |
| Velocidad con carga a 6V | 25 RPM |
| Torque Nominal a 6V | 1.5Kg.cm |
| Stall torque (Torque Motor Bloqueado) a 6V | 10Kg.cm |
| Consumo Corrriente sin Carga a 6V | 20 mA |
| Consumo Corriente Nominal a 6V | 40 mA |
| Consumo Corriente Motor Bloqueado a 6v | 1 A |



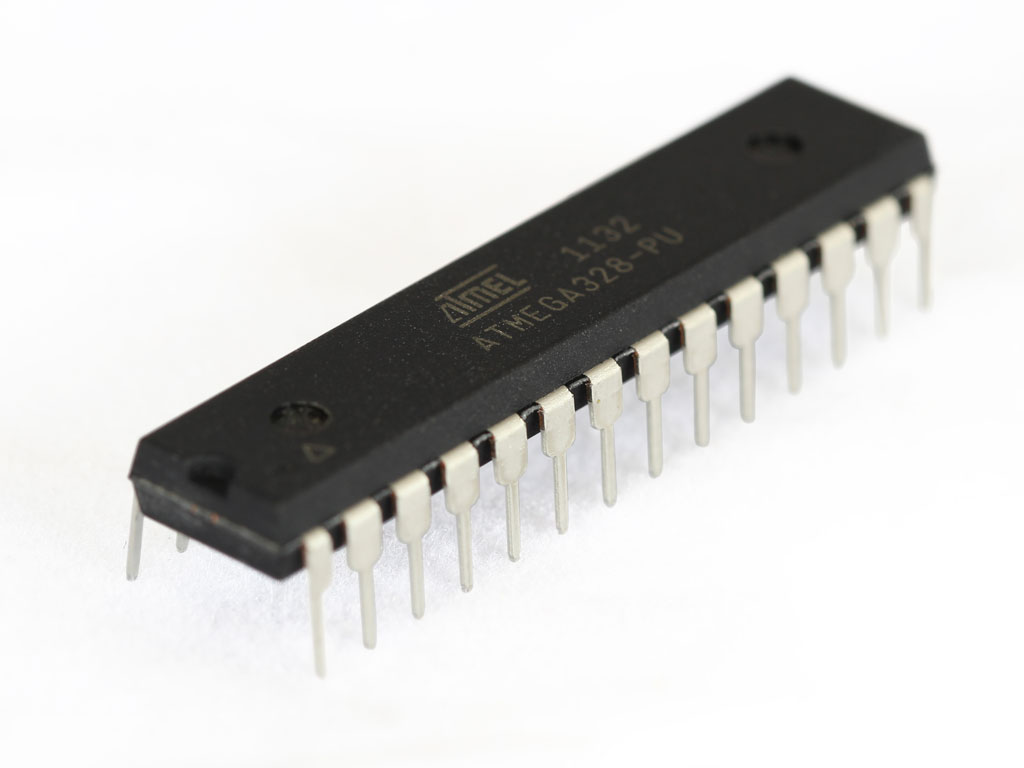
*Figura 6, Puente h, l298.*

**Puente H**, la figura 6 muestra el driver L298N, este es un dispositivo que permite controlar el sentido de funcionamiento de motores a una corriente de salida por canal de hasta 2A. Este módulo cuenta con un disipador de calor acorde a las caracterí­sticas de este driver, sus caracterí­sticas de diseño le permiten un rendimiento anti-interferencia excepcional, puede llegar a trabajar hasta con un nivel de tensión de entrada de 46V, aunque por cuestiones de seguridad se recomiendo usar niveles de tensión algo debajo de este valor lí­mite. Puede llegar a manejar un motor paso a paso de dos fases y cuatro fases, o dos motores de corriente continua. Este módulo incluye un regulador de voltaje 78M05 para obtener la energí­a de alimentación del dispositivo, sin embargo, cuando se superan los 12V, se sugiere usar una fuente de poder externa de 5V como fuente de alimentación digital y deshabilitar el jumper de 12V. Este módulo tiene gran capacidad de filtrado de ruido, cuenta con un diodo de protección ante corriente inversa, haciendo que su funcionamiento sea más estable y fiable.

La conexión del puente H, es la siguiente:



*Figura 7, conexión de l298.*

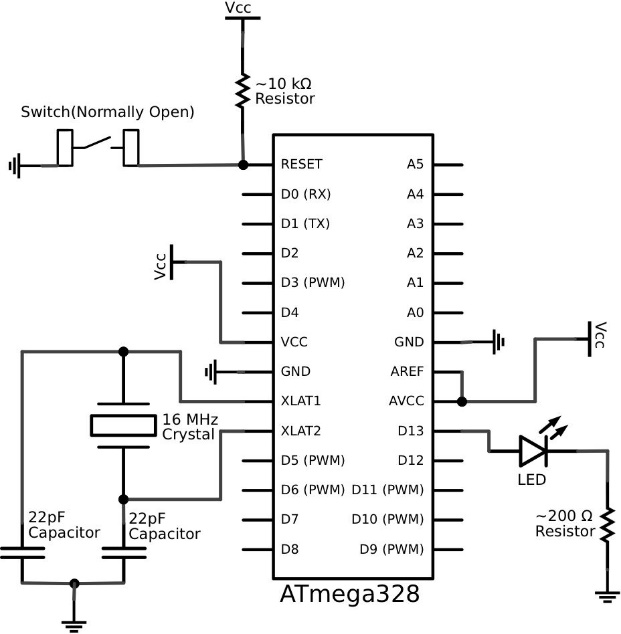


*Figura 8, microcontrolador ATmega 328p.*

**Microcontrolador,** ATmega 328p, este es un Microcontrolador RISC AVR de la compañí­a ATMEL, posee una resolución de 8 bits, con memoria Flash de 32KB con la capacidad de lectura y escritura, a través de comunicación ISP. Cuenta con una memoria EEPROM de 1024B, una SRAM de 2KB, con 23 pines de propósito general, usadas como entradas/salidas, cuenta con 32 registros de trabajo de propósito general, 3 temporizadores flexibles/comparadores, se pueden generar interrupciones internas y externas. Se programa a través de comunicación serie USART, con una interfaz serial de 2 hilos orientado a bytes. Cuenta con un puerto serie SPI, un conversor analógico a digital de 10 bits de 6 canales. Funciona a través de un oscilador interno el cual se encarga de proporcionar la frecuencias de reloj para que funcione correctamente.

Mediante la ejecución de las instrucciones de gran alcance en un solo ciclo de reloj, el dispositivo alcanza rendimientos se acercan a 1 MIPS por MHz, por lo que permite equilibrar el consumo de energí­a y la velocidad de procesamiento.

La conexión se realiza así:



*Figura 9, conexión ATmega 328p, para su utilización.*

# Planteamiento de la estructura del algoritmo.

Es usual que para este tipo de robots, se usen algoritmos basados en un algoritmo de rellenado por difusión, conocido como *Flood Fill*. Este determina el área formada por elementos contiguos en una matriz multidimensional. Se usa en la herramienta Bote de pintura de programas de dibujo para determinar qué partes de un mapa de bits se van a modificar. Para este caso las modificaciones son numéricas y representan la distancia de una posición determinada al centro de un laberinto de 6x6.

Para el inicializar el algoritmo se debe tener un registro inicial de los valores de las casillas, esto se hace con esta función (escrita en lenguaje c):

*Código 1, iniciación de cuadricula.*

// Inicializa la cuadrícula

for (int i = 0; i <= size/2-even; i++) {

for (int j = 0; j <= size/2-even; j++) {

gArray[i][j] = size-j-i-1-even;

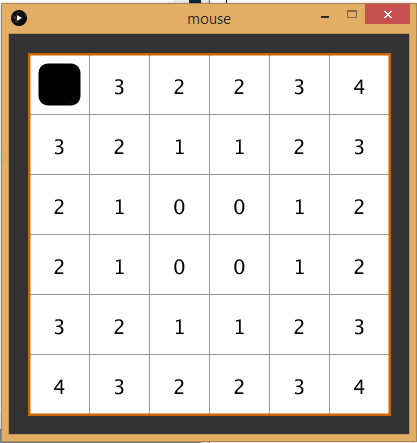
gArray[size-i-1][j] = size-j-i-1-even;

gArray[i][size-j-1] = size-j-i-1-even;

gArray[size-i-1][size-j-1] = size-j-i-1-even;

}

}

La que da un resultado cómo muestra la figura 10.

*Figura 10, interfaz de la Cuadricula inicial en Processing®.*

Ahora se debe otorgar una función que permita modificar las paredes del laberinto en función de los datos que envíe el robot, la función que modifica el estado de las paredes es la siguiente:

Código 2, actualización de paredes.

// Actualiza los muros de una celda

public void setWalls(int x, int y, int walls){

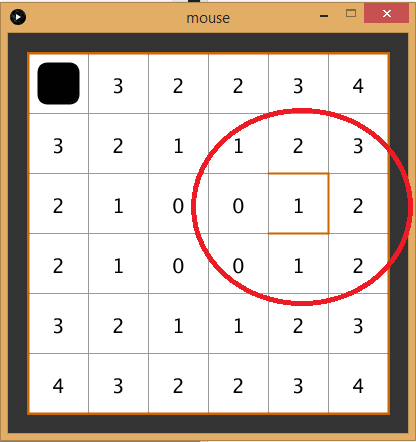
if( (x<gSize) && (y<gSize) ){

gWalls[x][y] = walls & 0xF;

}

}

Se obtiene un resultado así, cuando se le asigna, por ejemplo, la trama que corresponde a un punto muerto, esto son, tres paredes.



*Figura 11, asignación de paredes.*

La presencia de paredes, tiende a cambiar la distancia mínima entre un punto y el centro, la presencia de paredes por lo tanto determina el orden lógico mediante el cual se modifican los valores en las casillas. Modificar los valores de las casillas se hace con ésta función:

Código 3, asignación de valores a las cuadriculas.

// Actualiza el valor de una celda

public void setValue(int x, int y, int value){

if( (x<gSize) && (y<gSize) )

gArray[x][y] = value;

}

Frecuentemente será necesario, para el algoritmo conocer conocer el valor de una celda, esto se realiza mediante la función del código 4:

Código 4, obtención de valores de una celda.

// Obtiene el valor de una celda

public int getValue(int x, int y){

if( (x<gSize) && (y<gSize) )

return gArray[x][y];

else

return 0;

}

Para el ejemplo de la figura 11, se modifican los valores de las casillas:



*Figura 12, actualización de casillas.*

Debe notarse ahora, que existe un elemento que representa la ubicación del robot en el laberinto, este es el recuadro negro en la esquina superior izquierda de la figura 12. El código que actualiza ésta posición se debe ejecutar cada que se envíe una trama desde el robot hacia el computador. Este código es:

*Código 5, actualización de la posición del robot.*

*// Actualiza la posición del mouse*

*public void setMouse(int x, int y){*

*if( (x<gSize) && (y<gSize) ){*

*mPosx = x;*

*mPosy = y;*

*}*

*}*

La interfaz que constantemente muestra los valores, paredes y ubicación del Robot, se debe actualizar constantemente, la función que hace esto es la que sigue:

Código 6, actualización de interfaz.

public void display(){

// Dibuja la cuadrícula

for (int x = 0; x < gSize; x++) {

for (int y = 0; y < gSize; y++) {

// Color de relleno para las figuras

fill(255);

stroke(153);

strokeWeight(1);

// Dibuja un rectángulo en la posición

// x, y de un ancho de 30 x 30

rect( (x\*widthSquare)+20, (y\*widthSquare)+20, widthSquare, widthSquare);

// Color del texto

fill(1);

// Coloca los números en la cuadrícula

text(gArray[x][y], (x\*widthSquare)+50, (y\*widthSquare)+50);

}

}

// Color de línea

stroke(204, 102, 0);

// Ancho paredes

strokeWeight(2);

// Dibuja las paredes

for (int x = 0; x < gSize; x++) {

for (int y = 0; y < gSize; y++) {

// Dibuja las paredes

if( (gWalls[x][y]&NORTH) == NORTH)

line((x\*widthSquare)+20, (y\*widthSquare)+20, (x\*widthSquare)+widthSquare+20, (y\*widthSquare)+20);

if( (gWalls[x][y]&EAST) == EAST)

line((x\*widthSquare)+widthSquare+20, (y\*widthSquare)+20, (x\*widthSquare)+widthSquare+20, (y\*widthSquare)+widthSquare+20);

if( (gWalls[x][y]&SOUTH) == SOUTH)

line((x\*widthSquare)+20, (y\*widthSquare)+widthSquare+20, (x\*widthSquare)+widthSquare+20, (y\*widthSquare)+widthSquare+20);

if( (gWalls[x][y]&WEST) == WEST)

line((x\*widthSquare)+20, (y\*widthSquare)+20, (x\*widthSquare)+20, (y\*widthSquare)+widthSquare+20);

}

}

stroke(1);

// Dibuja el mouse

rect( (mPosx\*widthSquare)+30, (mPosy\*widthSquare)+30, widthSquare-20, widthSquare-20, 10);

}