**Introducción**

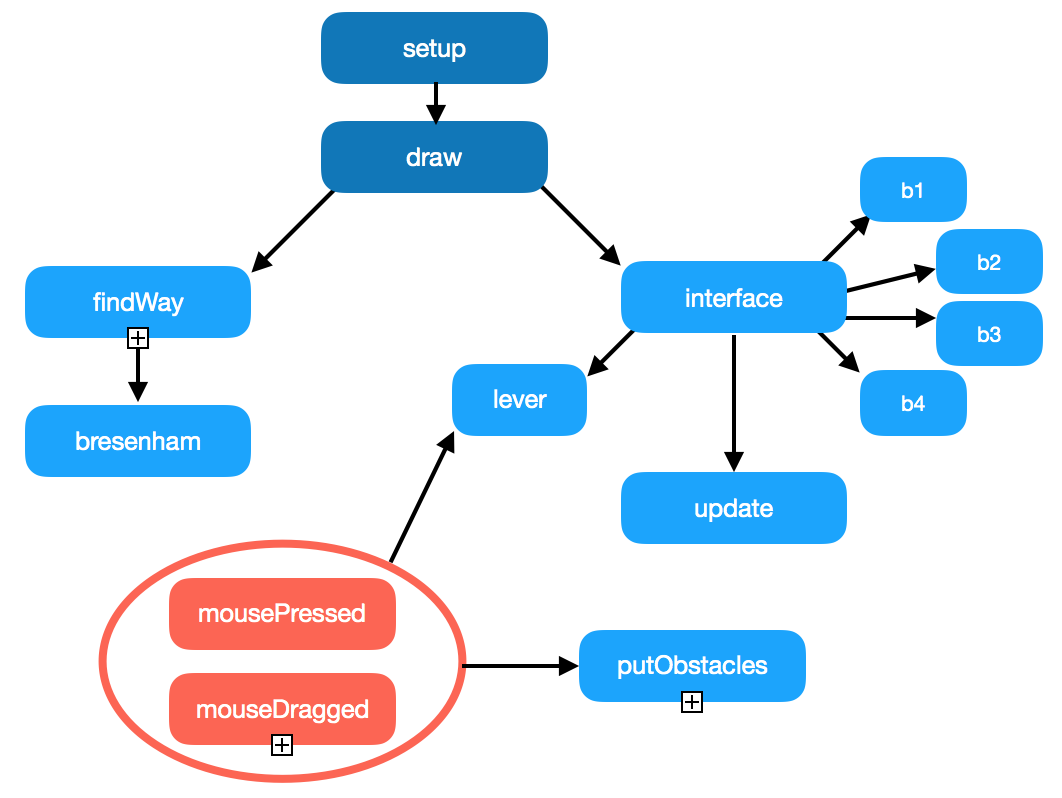
Implementación de un agente inteligente en un campo con obstáculos (matriz de nxn casillas), con un origen y un destino, teniendo la capacidad de desplazarse por si solo tomando decisiones en cada momento durante el recorrido, el agente inteligente deberá desplazarse haciendo uso del algoritmo de bresenham para el trazado de una linea. Cuando el agente encuentre un bloqueo en el desarrollo de su trayecto deberá tomar una decisión aleatoria para moverse a alguna de sus casillas vecinas, teniendo como ultimo recurso la ultima casilla que tomo ( que es de donde viene ). Se utilizaran banderas para marcar casillas no deseadas, pudiendo almacenar hasta 5 banderas cada casilla, donde 5 banderas implica que la casilla ya no es viable para realizar un trazado de bresenham desde esa posición y entonces se marca como un obstáculo más en el campo. La cantidad de obstáculos debe de ser variable, pudiendo poner desde 0% hasta un 100% de obstáculos en el campo ( siempre teniendo 2 casillas disponibles, que pertenecerán al origen y al destino del agente ).

**Desarrollo e implementación**

[](https://user-images.githubusercontent.com/28017456/45577444-99f2a080-b841-11e8-8d05-9aafa4410180.png)

La actividad fue realizada en Python con Processing Que básicamente es un lienzo en el que tiene que dibujar absolutamente todo, incluso botones. Para esta actividad se agrego la posibilidad de agregar entre 0 - 100 % de obstáculos disponibles en el campo, menos 2 que son el origen y el destino. Se hace uso del algoritmo de bresenham para calcular las próximas posibles posiciones del agente, recalculandose y agregando banderas cada vez que se interfiere con la ruta.

**Diagrama de Funciones**

[](https://user-images.githubusercontent.com/28017456/45577479-bf7faa00-b841-11e8-9b4b-0ed0db0474f3.png)

La inicialización formal de todos las variables se realiza dentro del setup

Todo el funcionamiento de processing se basa en el siguiente paso: el draw Que itera sin parar repitiendo una y otra vez todo lo que este dentro de el, de esta manera pueden hacerse animaciones y dibujos dentro del lienzo. Nada puede forzar ni alterar de ninguna manera una iteración del draw, asi que hace falta organización para poder lograr la visualización paso a paso.

Dentro del draw se llaman únicamente dos funciones: findWay e interface

FindWay hace uso de la función bressenham que llena una lista global con pares de puntos ordenados en la linea trazada, cada vez que es necesario. Para poder apreciar el movimiento del agente inteligente se hace uso de un iterador externo a la función que avanza un solo paso a la vez, cada vez que esta es llamada, esto es cada vez que draw realiza una iteración, de esta manera y con un Delay, se crea una muy buena animación del comportamiento del agente.

def findWay():

global avax,avay,lastBox,play,iterBresen, doBresen, grid, played

options=[]

if avax==trex and avay==trey:

endWin()

return

if doBresen:

bresenham(avax,avay,trex,trey)

print "actual: "+str(avax)+" "+str(avay)

if len(bresenList)>1 and grid[bresenList[iterBresen][0]][bresenList[iterBresen][1]]<5:

lastBox.push((avax,avay))

avax=bresenList[iterBresen][0]

avay=bresenList[iterBresen][1]

if(avax==trex and avay==trey):

endWin()

return

iterBresen+=1

doBresen=False

return

else: grid[avax][avay]+=1

doBresen=True

cont=0

for x in range(-1,2):

for y in range(-1,2):

if avax+x>=0 and avax+x<=sz-1 and avay+y>=0 and avay+y<=sz-1:

if lastBox.size()>0 and grid[avax+x][avay+y]<5 and (avax+x,avay+y)!=lastBox.peek() and not (x==0 and y==0) :

cont+=1

options.append((x,y))

elif lastBox.isEmpty() and grid[avax+x][avay+y]<5 and not (x==0 and y==0):

cont+=1

options.append((x,y))

if cont==0 and lastBox.size>0:

elem=lastBox.peek()

if grid[elem[0]][elem[1]]<5:

grid[avax][avay]+=1

avax=elem[0]

avay=elem[1]

lastBox.pop()

return

endLose()

return

elif cont==0 and lastBox.size==0:

endLose()

return

aux = random.choice(options)

lastBox.push((avax,avay))

avax+= aux[0]

avay+= aux[1]

if(avax==trex and avay==trey):

endWin()

return

Ademas del findWay, draw también llama a interface, que esta se encarga de redibujar todo en cada paso, llamando a b1,b2,b3 y b4, que son los botones y textos disponibles para la interacción del usuario. Interface hace llamado también a update, que es la encargada de redibujar todos los elementos de la matriz.

Esta actividad está basada en una matriz de nxn que se declara al inicio de la ejecución y todo alrededor de esta esta armado a “escala”, permitiendo así poder aumentar o disminuir el tamaño de la matriz sin ningún problema y manteniendo la funcionalidad.

La matriz puede contener alguno de los siguientes valores:

* -2: Tesoro al que debe llegar el agente ( Destino )
* -1: Agente inteligente o Avatar ( Origen )
* 0: Camino de libre circulación del agente
* 1 - 4: Banderas colocadas a casillas no deseadas ( Disminuye la transparencia del obstáculo )
* > 5: Obstáculo, el agente no es capaz de tomar la posición de la casilla donde se encuentre algún obstáculo

De esta manera, dentro de las funciones donde se toman decisiones sobre el recorrido, únicamente es necesario alterar el valor de la matriz y la función update se encargara de redibujar todo en base a las reglas anteriores

def putObstacles():

global avax,avay,trex,trey

v = [[(j,i) for i in range(0,sz)] for j in range(0,sz)]

clearGrid()

total=sz\*sz

objectsP=int(total\*percent)/100

if avax==trex and avay==trey:

avax=0

avay=0

trex=sz-1

trey=sz-1

if objectsP>total-2:

objectsP = total-2

v[avax].remove((avax,avay))

v[trex].remove((trex,trey))

print objectsP

while objectsP>0:

aux=random.choice(v)

while len(aux)==0:

aux=random.choice(v)

pos=random.choice(aux)

if grid[pos[0]][pos[1]]!=5 and not(pos[0]==avax and pos[0]==trex) and not(pos[1]==avay and pos[1]==trey):

grid[pos[0]][pos[1]]=5

objectsP-=1

aux.remove(pos)

elif grid[pos[0]][pos[1]]!=5 and avax==trex and pos[0]==avax and pos[1]!=avay and pos[1]!=trey:

grid[pos[0]][pos[1]]=5

objectsP-=1

aux.remove(pos)

elif grid[pos[0]][pos[1]]!=5 and avay==trey and pos[1]==avay and pos[0]!=avax and pos[0]!=trex:

grid[pos[0]][pos[1]]=5

objectsP-=1

aux.remove(pos)

Por último se utilizaron 2 funciones que operan de manera independiente al draw: mousePressed y mouseDragged que actúan en función a indicaciones recibidas por medio del mouse, que estas a su vez manejan las funciones putObstacles y lever, que van directamente relacionadas la una a la otra.

Lever es la barra disponible para que el usuario pueda modificar el porcentaje de obstáculos con solo deslizar el mouse, esta llama a putObstacles cada vez que sus valores son alterados y se dibujan elementos de manera aleatoria sobre el campo.