# Róbotica - Taller 1
# Introducción a ROS

John Alejandro Duarte Carrasco - ja.duarte10@uniandes.edu.co - 2016140888
Jonathan Steven Roncancio Pinzon - js.roncancio@uniandes.edu.co - 201617312
Miguel Angel Mozo Reyes - ma.mozo@uniandes.edu.co - 201615973
Santiago Devia Valderrama - s.devia10@uniandes.edu.co - 201313766

February 14, 2019

## Punto 1

*Enunciado:* Cree un nodo de ROS que sea capaz de iniciar un juego en cualquier mapa que contenga un Pacman y que a través de lectura del teclado del computador sea capaz de controlar de manera directa a dicho Pacman. Es decir, que las acciones de Pacman deben coincidir con las teclas presionadas por el usuario (si no se presiona ninguna tecla Pacman debe quedarse quieto).

*Solución:*
Dados los requerimientos prácticos se desea que pacman obedezca las ordenes de un usuario mediante el uso de un teclado como control. La acción que debe seguir pacman en este caso se modela como una variable global con el nombre de "*accion*" y puede tener algún valor entre 0 y 4.

En primer lugar, el nodo se encarga de realizar la solicitud del servicio del mapa para comenzar el juego. En segundo lugar, con el fin de leer y analizar la acción que desea realizar el usuario, dentro del mismo nodo se creo un método llamado "$KeyAction$" encargado de decidir que acción tomar en función de la tecla que presiona el usuario, dependiendo de la tecla pulsada (w,a,s,d) define si se hace un movimiento hacia arriba, abajo, izquierda o derecha; siendo w arriba, s abajo, a izquierda y d derecha. Así mismo, se modifica la variable global accion por su valor correspondiente; tomando 0 como desplazamiento hacia arriba, 1 como desplazamiento hacia abajo, 2 como desplazamiento a la derecha y 3 como desplazamiento a la izquierda.

Adicionalmente, el nodo posee una función que detienen al pacman mientras el usuario no presione ninguna tecla, la función se llama "$stopMove$" y asigna a la variable accion el valor de 4 que al ser publicado en el tópico "*pacman_actions*" detiene el movimiento del pacman, o lo que es igual hace que su aceleración lineal en cualquiera de los dos ejes cardinales sea 0.

Por ultimo, se publica en el tópico "pacman_actions0" la variable accion ya modificada y el nodo queda a la espera de una nueva acción por realizar. A continuación se presenta el modelo de bloques que modela el nodo creado al igual que la gráfica generada por ROS sobre la comunicación entre los nodos:


Figure 1: Configuración de nodos del punto 1.

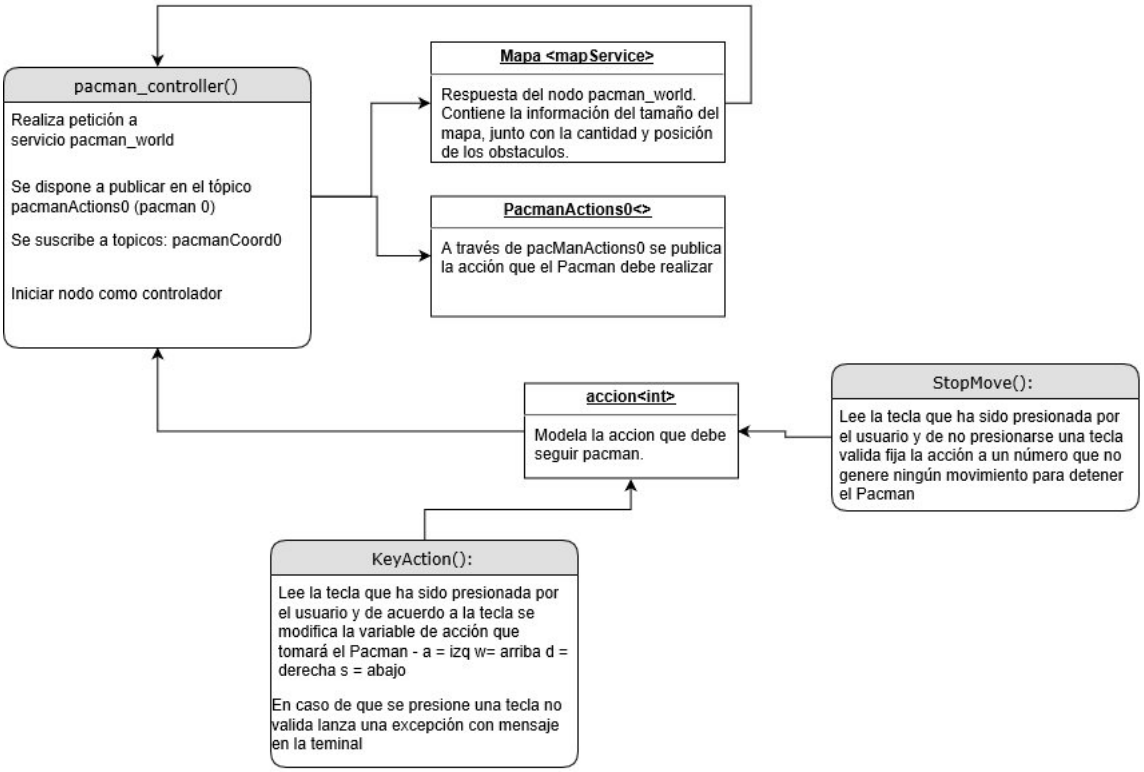


Figure 2: Diagrama de bloques del código p1MoverPacman.py

Para implementar la solución se utilizo una taza de muestreo de $\frac{1}{0.15}Hz$ o lo que es lo mismo 150ms, tiempo que corresponde a la taza de refresco de todos los nodos y topicos conectados a pacman. Además, se utilizaron librerías de python como rospy y pynput. La primera para poder publicar y suscribirse a los tópicos así como también realizar peticiones a los servicios, y la segunda para leer los eventos creados por el usuario al presionar una tecla.

*Análisis y resultados:* En primer lugar, fue posible iniciar un nuevo juego en cualquier mapa con un pacman utilizando un nuevo nodo que se agrega a la red. Así mismo, se observo que pacman obedece correctamente las decisiones tomadas por el usuario sin importar si era o no posible realizar el movimiento en la dirección dada. Además, después de crear el código fue notorio que mediante el correcto uso de ROS y sus herramientas, es sencillo transmitir información y acciones entre los objetos conectados a la red (nodos,servicios y tópicos).

# Punto 2

*Enunciado:* Cree un nodo de ROS que sea capaz de iniciar un juego en cualquier mapa y que muestre (puede ser en la terminal) todos los elementos del juego (obstáculos, Pacman, fantasmas, galletas y bonos). Es necesario que se vea cómo cambian los elementos del mundo mientras se mueven (no es necesario que Pacman haga movimientos).

*Solución:*

Para la implementación de este punto se creó un código en Python con el nombre *p2MostrarMapa.py*. Este código tenia el objetivo de iniciar un nodo ante ROS con el nombre *controlador_5_mapa*, y suscribirse a los tópicos *bonusCoord, pacmanCoord, cookiesCoord* y *ghostsCoord* del nodo *pacman_world* a manera de obtener la información necesaria para imprimir en consola el mapa de Pacman (ver figura 3).

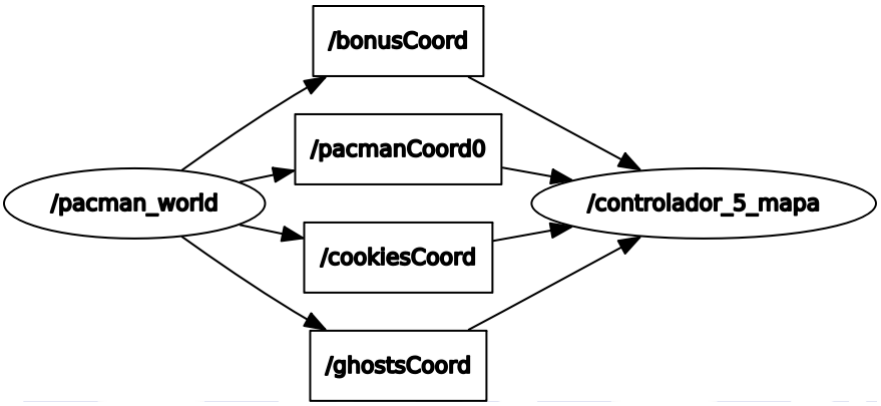


Figure 3: Configuración de nodos entre el nodo pacman_world y el nodo controlador_5_mapa

La manera en la que este código funciona se puede apreciar a manera de diagrama de bloques en la figura 4. Una vez el código se ejecuta, y esta complemente cargado, se inicia la función *mostrarMapa*, función principal que inicia el nodo ante ROS, se suscribe a los tópicos y envía la petición de tipo *mapService* a *pacman_world* con el nombre del jugador, para obtener la información del tamaño del mapa, la posición y cantidad de los obstáculos, almacenando dicha información en la variable *mapResponse*, iniciando al mismo tiempo el juego. En esta misma función, se tiene la linea de código que mantiene ejecutando el nodo a una tasa especificada, para este caso a una frecuencia de 10Hz, y le indica que hacer mientras este esté en funcionamiento. Para este caso, mientras el nodo *controlador_5_mapa* se mantenga activo, se actualizara la información del mapa llamando a la función *actualizarMapa* y imprimira la matriz del mapa *mapInfo* en consola.

Cada vez que se actualiza la información en un tópico, se lanza una función callback independiente para cada uno, enviando por parámetro la información de dicho tópico. Para este caso dichas funciones son *callbackCookies-Pos*, *callbackBonusPos*, *callbackGhostsPos*, *callbackPacmanPos1* y *callbackPacmanPos2*. Las primeras tres funciones toman la información recibida por parámetro y realizan cambios en tres listas diferentes, siendo estas *cookiesInfo*, *bonusInfo* y *ghostsInfo* respectivamente. Dichas listas son de tamaño 2, donde la primera posición es la cantidad de elementos (galletas, bonus y fantasmas) y la segunda posición es, nuevamente, una lista con diccionarios que indican la posición de dicho elemento. Aquí se hace la aclaración que dichas posiciones de cada elemento (x,y) están calculadas de acuerdo al formato de índices de una matriz, es decir, 0 en $x$ es la primera columna de izquierda a derecha y 0 en $y$ es la primera fila de arriba hacia abajo. Para el caso de las dos funciones restantes, asociadas a la posición de los Pacman, se tiene el mismo funcionamiento que la funciones anteriores, solo que ahora la información no se actualiza en una lista sino directamente en un diccionario, siendo estos *pacmanInfo1* y *pacmanInfo2*. La forma de estos diccionarios es {'x': posX, 'y': posY}, donde los valores posX y posY se calculan como se indicó anteriormente.

El código cuenta como constantes globales las variables *OBSTACULO: %*, *PACMAN: P*, *COOKIE: .*, *BONUS: o* ,*GHOST: "G"* y *BLANCO:* , variables tipo *String*, que representan un elemento particular del juego.

Como se dijo anteriormente, el código, en cada ejecución, mantiene llamando a la función *actualizarMapa*. Esta función, primero, reemplaza el contenido que este presente en la variable *mapInfo* por una matriz mxn de BLANCOS, siendo m y n los valores calculados de anchura y altura a partir de la información almacenada en *mapResponse*. Una vez aquí, se actualiza primero la posición de cada obstáculo en la matriz, seguido por la información de las galletas, bonus, fantasmas y por últimos pacmans, disponiendo en cada caso las constantes respectivas mencionadas anteriormente.

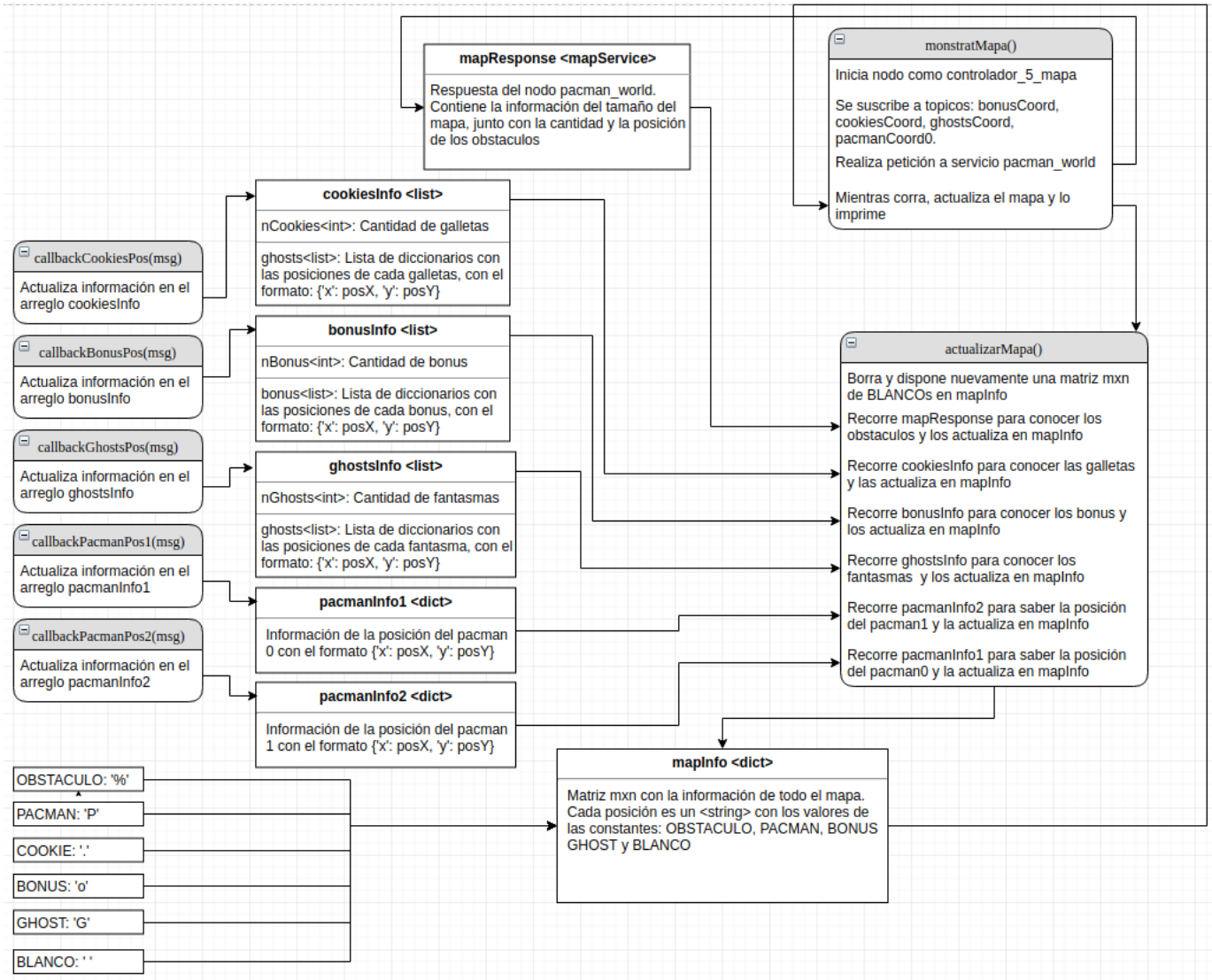


Figure 4: Diagrama de bloques del código p2mostrarMapa.py

Una vez la matriz es actualizada, se imprime, fila por fila, en consola. En la figura 5 se puede apreciar el mapa *mediumClassic* para un momento dado de tiempo. A manera de comparación se dispuso la figura 6 para poder evidenciar la relación entre el mapa dado por el nodo de *pacman_world* y el código implementado.

```
['%', '%', '%', '%', '%', '%', '%', '%', '%', '%', '%', '%', '%', '%', '%', '%', '%', '%', '%', '%', '%']
['%', 'o', '.', '.', '.', '.', '%', '.', '.', '.', '.', '.', '.', '.', '.', '%', '.', '.', '.', '.', '%']
['%', '.', '%', '%', '.', '.', '%', '.', '%', '%', '%', '%', '%', '%', '%', '.', '%', '.', '%', '%', '%']
['%', '.', '%', '.', '.', '.', '.', '.', '.', '.', '.', '.', '.', '.', '.', '.', '.', '.', '%', '.', '%']
['%', '.', '%', '.', '%', '%', '.', '%', '%', ' ', ' ', ' ', '%', '%', '.', '%', '%', '.', '%', '.', '%']
['%', '.', '.', '.', '.', '.', '.', '%', ' ', ' ', ' ', ' ', ' ', '%', '.', '.', '.', 'G', '.', '.', '%']
['%', '.', '%', '.', '%', '%', '.', '%', '%', '%', '%', '%', '%', '%', '.', '%', '%', '.', '%', '.', '%']
['%', '.', '%', '.', '.', '.', '.', '.', '.', '.', '.', '.', '.', '.', '.', '.', '.', '.', '%', '.', '%']
['%', '.', '%', '%', '.', '%', '.', '%', '%', '%', '%', '%', '%', '%', '.', '%', '.', '%', '%', '.', '%']
['%', '.', '.', 'G', '.', '%', '.', '.', '.', 'P', '.', '.', '.', '.', '.', '%', '.', '.', 'o', '%', '%']
['%', '%', '%', '%', '%', '%', '%', '%', '%', '%', '%', '%', '%', '%', '%', '%', '%', '%', '%', '%', '%']
```

Figure 5: Captura en un momento del tiempo del mapa mostrado por el nodo controlador_5_mapa.

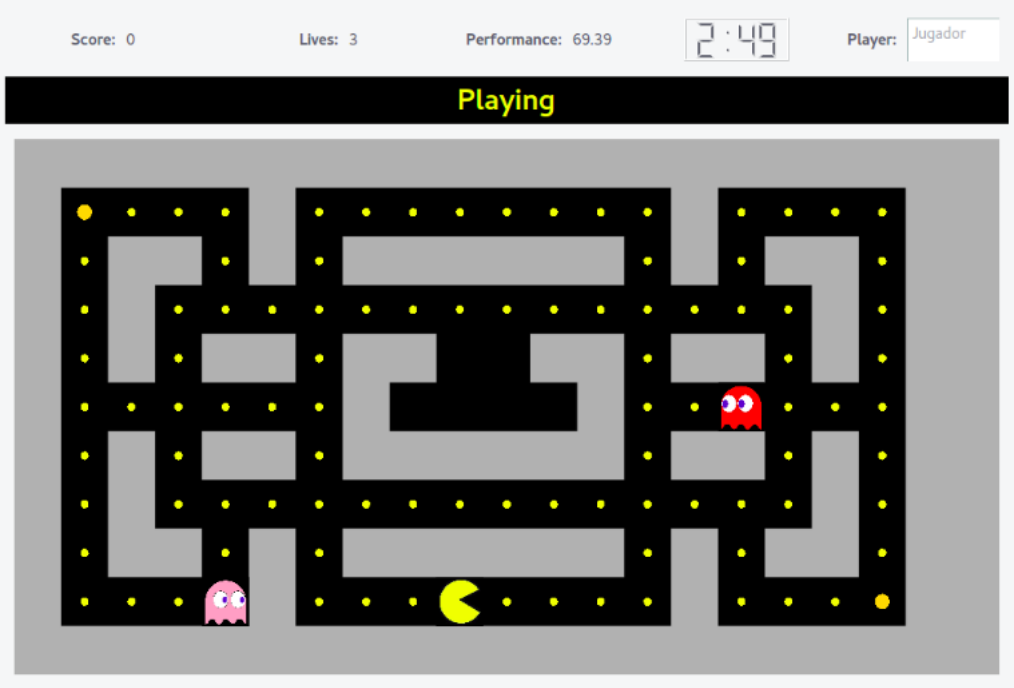


Figure 6: Captura en un momento del tiempo del mapa mostrado por el nodo pacman_world

A manera de análisis, se determina que la implementación para este código fue correcta. Se puede apreciar como para un tiempo arbitrario dado, se tiene una actualización de la información constante y correcta, permitiendo al usuario poder identificar con gran facilidad los componentes del mapa en una forma muy sencilla, como lo son caracteres de texto.

# Punto 3

*Enunciado:* Cree un nodo de ROS que sea capaz de iniciar un juego en cualquier mapa que contenga un Pacman y que a través de lectura del teclado del computador sea capaz de controlar a dicho Pacman siguiendo la mecánica del juego original. Es decir: Pacman se moverá en la última dirección válida escogida por el usuario. Si Pacman se encuentra en movimiento y el usuario escoge una acción que no es válida a en ese momento (por ejemplo hacia arriba en un pasillo horizontal), deberá seguir con su trayectoria y ejecutar dicha al llegar a la siguiente intersección donde dicho movimiento sea válido. Además de los requerimientos generales de entrega descritos en las instrucciones cada grupo deberá hacer un vídeo mostrando la situación descrita. Asegúrese de mostrar en el vídeo la escena de Pacman moviéndose en el laberinto y las acciones realizadas por el usuario en el teclado de manera sincronizada.

*Solución:*

Para realizar la solución de este punto se desarrolló un Script en Python en el archivo punto3.py. Este *script* permite la inicialización de un Nodo ROS el cual se relacionaría con el Servicio de Mapa de Pacman y publicaría las acciones que debe realizar Pacman en el tópico *pacmanActions0*. Estas relaciones se pueden ver en figura 7

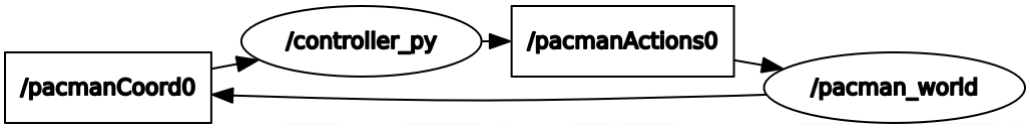


Figure 7: Configuración nodos punto 3

El funcionamiento de este se basa en el mismo principio de funcionamiento del primer punto. Una vez el nodo inicia, y se une a ROS como el nodo controller_py, se dispone a publicar en el tópico pacmanActions0 y se suscribe al tópico pacmanCoords0. Una vez aquí, haciendo uso de la libreria pynput, se programa el código de manera que cada vez que se presione una tecla se lanza la función ActionKey(key). Cabe aclarar que tambien se hace uso de la librería threading, usada en este caso para crear un hilo aparte al hilo principal de ejecución, que permita el listening de las teclas.

Así pues, la función ActionKey(key) compara primero si la tecla presionada es igual a última tecla presionada (Variable OldKey), actualizando la variable OldKey en caso de que estas sean diferentes. Una vez aquí el código

determina, a partir de condicionales, cual es la tecla presionada. Una vez aquí, el programa determina si la acción que se va a tomar es valida, llamando a la función MovimientoValido(pos,posA,dirección), que pide como parámetros las posición del pacman sobre el eje en el que se quiere mover, la posición del pacman sobre el eje en el que no se moverá y la dirección hacia la cual se quiere mover. En caso de que la acción sea valida, se guarda en la variable global acción.

En cada iteración, que se da a una tasa de 150Hz, se publica la acción actual en el tópico del pacman 0.

Para mostrar el funcionamiento a grandes rasgos del *script* y los nodos que intervienen en la realización del punto 3, se desarrolló el diagrama mostrado en la figura 8, la cual muestra el funcionamiento macro del algoritmo a través de un diagrama de bloques. Así mismo, se puede evidenciar en el link `https://bit.ly/2IbkvZP` un video del funcionamiento de este punto.

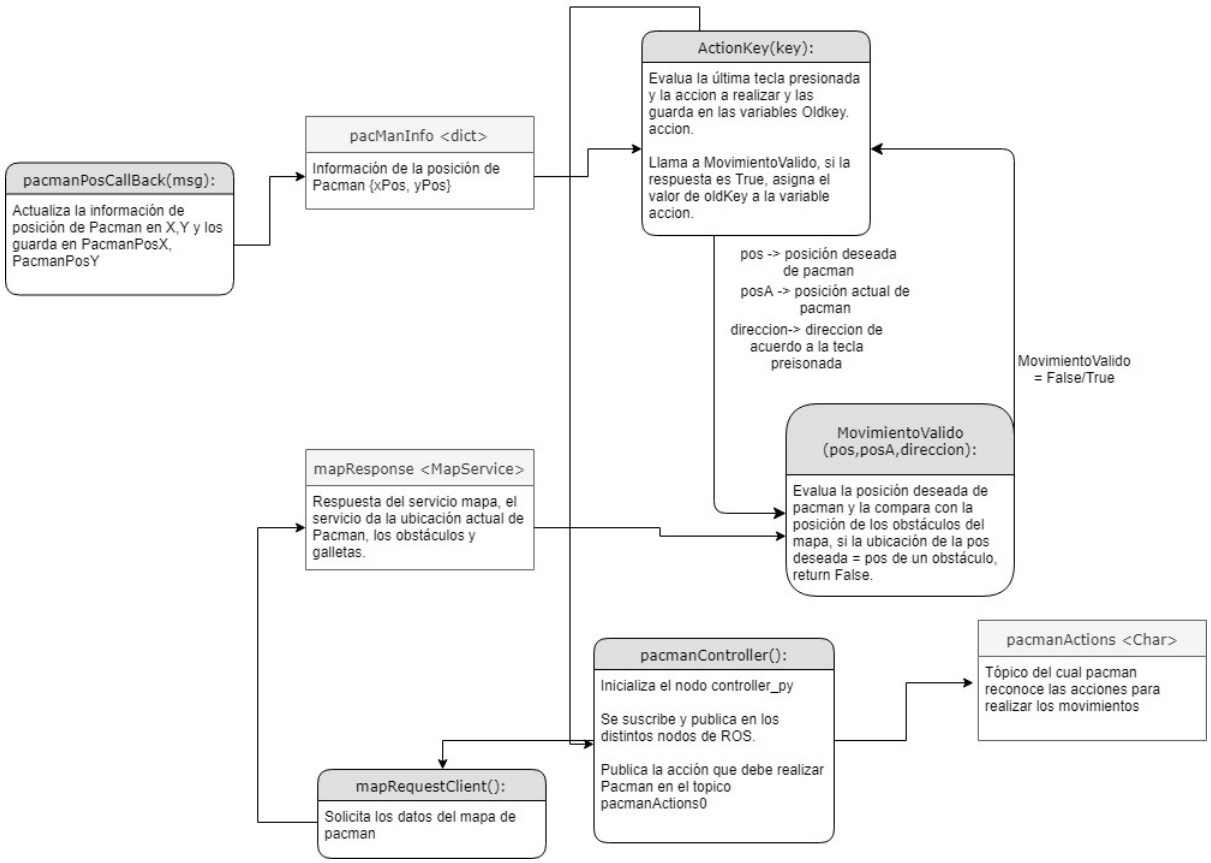


Figure 8: Diagrama de bloques del código p3MoverPacman.py

# Punto 4

*Enunciado:* Cree un nodo de ROS que sea capaz de iniciar un juego en el mapa mediumCorners y controlar a Pacman en dicho mapa siguiendo el algoritmo de la pared (regla de la mano derecha). En este método, el agente que resuelve el laberinto se mueve a través de los pasillos del mismo y cuando llega a un punto donde tiene más de una opción de decisión siempre toma la decisión de ir "lo más a la derecha posible". Para mayor información puede ver el vídeo de demostración en Sicua+. Además de los requerimientos generales de entrega descritos en las instrucciones cada grupo deberá hacer un vídeo mostrando la situación descrita.

*Solución:* Con el fin de solucionar el problemas planteado, se creo un código en python con el nombre *p4Mover PacmanPared.py*. Al ejecutarse el código crea 5 variables locales que cumplirán la función de modelar el mundo y las acciones de pacman. Dichas variables son:

- Mapa: variable que modela al mapa del juego y es inicializa al pedir el servicio "*mapService*". Esta variable contiene la ubicación de todos los obstáculos en el mapa tomando como punto (0,0) el centro del mismo.
- accion: Esta variable global corresponde a la dirección actual en la que se debe mover pacman y es la variable que se publica al tópico "$pacman_Actions0$".

- nObs: Numero de obstáculos en el mapa
- PacmanPosX: posición en x actual del pacman referida al mapa
- PacmanPosY: posición en y actual del pacman referida al mapa

Seguidamente, se ejecuta la función "*pacman_controller*" donde se inicia el nodo "controller_py"; se suscribe dicho nodo al tópico "pacmanCoord0" con el fin de conocer la posición de pacman en todo momento e inicializar las variables "PacmanPosX" y "PacmanPosY".Luego, se crea la conexión entre el tópico "pacmanActions0" y el nuevo nodo, con esta relación se define una variable "pub" que sera la encargada de publicar las acciones al tópico. A continuación, se hace una petición al servicio "mapService" para que este inicie el juego y de paso inicializar la variable mapa para conocer la distribución de obstáculos presentes en el. Para más claridad acerca de la distribución de nodos, se puede evidencia la figura 9.

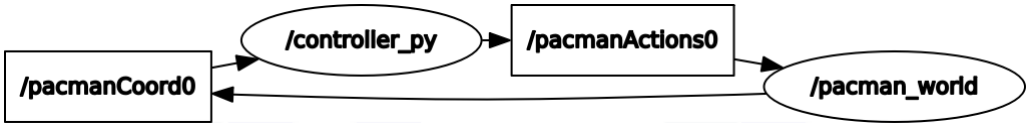


Figure 9: Diagrama de nodos del punto 4.

Una vez se realizaron las relaciones con tópicos y servicios, y se inicializaron las variables, se llama a la función "determinarAccion". Dicha función, determina que opción debe tomar pacman con el fin de moverse lo mas a la derecha posible teniendo en cuenta todas las posibilidades(que el movimiento a la derecha sea valido, que no lo sea pero pueda seguir en la misma dirección de movimiento, que pueda moverse a la izquierda o que como ultimo recurso, tenga que devolverse).

Con el fin de simplificar las tareas que debía cumplir la función, se crearon sub-funciones para dividir las responsabilidades macro en pequeñas responsabilidades mas fáciles de cumplir. La primera de estas funciones es "MovimientoValido", esta función recibe una dirección como parámetro y verifica que sea posible que pacman se mueva en esa dirección; el algoritmo utiliza las posiciones de los obstáculos y el numero de los mismos para realizar la verificación.

El segundo método utilizado es "buscarDerecha", como es lógico la derecha de pacman cambia dependiendo de hacia que dirección se este moviendo. Por ejemplo, si se mueve hacia arriba su derecha sera la derecha (dirección este), mientras que si se mueve hacia abajo su derecha sera la izquierda(dirección oeste). El método "buscarDerecha" se encarga de esto, recibe como parámetro la acción que esta ejecutando actualmente pacman y retorna la acción que corresponde a "moverse a la derecha" dependiendo de su dirección de movimiento.

La ultima función utilizada se llama "devolverse", como su nombre lo indica busca hacer que pacman se devuelva; toma la variable global acción, verifica su valor y asigna a acción su opuesto. Es decir, si pacman va hacia la derecha asigna a acción 1 que corresponde a ir para atrás; o sí pacman va hacia la derecha asigna a acción 3 que corresponde a ir hacia la izquierda.

Finalmente, "determinarAccion" decide que accion debe tomarse y se publica en el tópico "pacmanActions0".El funcionamiento del código se puede apreciar mas clara y resumidamente en la figura 10.

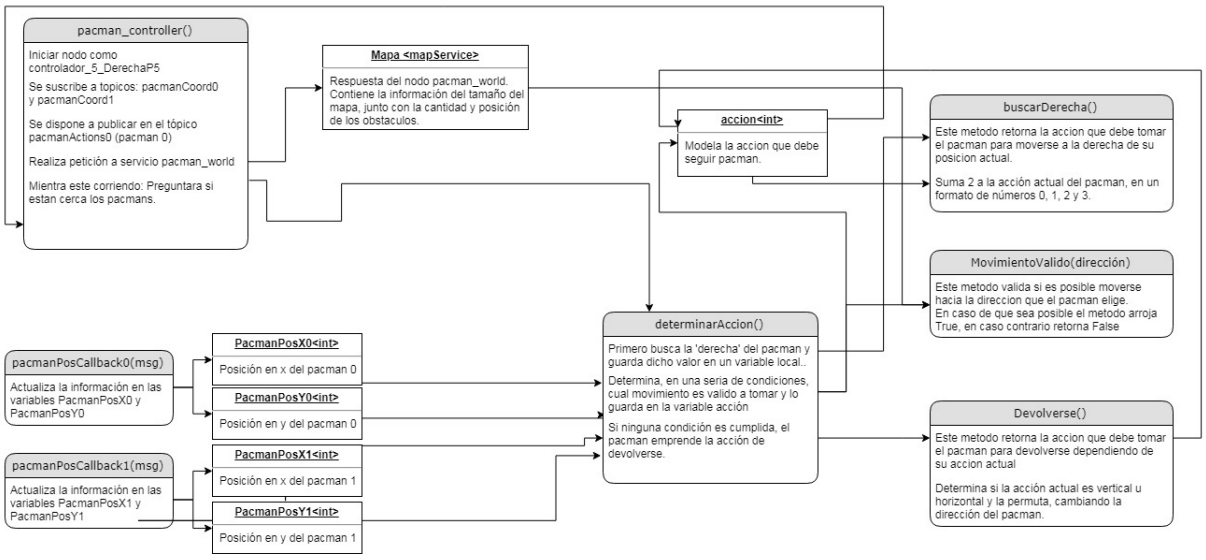


Figure 10: Diagrama de bloques del código p4MoverPacmanPared.py

Esta solución trabaja con una taza de refresco de $\frac{1}{0.15}Hz$ o 150ms, que corresponde a la taza de refresco de todos los elementos conectados a pacman. Por otra parte, se utilizó la librería rospy de python para poder publicar y suscribirse en los tópicos, así como también hacer peticiones a los servicios del juego.

*Análisis y resultados:* Como se puede observar en el link `https://bit.ly/2TNqBRF` el algoritmo ejecuto el método de la mano derecha para tomar cada una de sus decisiones, moviéndose en cada momento lo mas a la derecha que le fuera posible. Gran parte del funcionamiento del algoritmo es gracias a las facilidades que ofrece ROS como sistema operativo, el tener una estructura bien definida que permite a múltiples nodos hacer uso del mismo tópico ofrece la posibilidad de realizar múltiples tareas simultáneamente utilizando la información suministrada por la propia red de nodos, sin tener que calcular o sensar valores dentro del nodo. Así mismo, tener mensajes tan claros tanto para servicios como para tópicos facilita la comunicación e implementar nuevos nodos que utilicen las herramientas ya existentes (tópicos y servicios).

A pesar de que en este caso el algoritmo funciona, se tiene que tener en cuenta que el método de la mano derecha no funciona para todos los laberintos, sino que se deben cumplir ciertas condiciones para que funcione. Esto significa que este algoritmo no se puede utilizar como inteligencia artificial para pacman, dado que es muy limitado.

# Punto 5

*Enunciado:* Cree al menos dos nodos de ROS que sean capaces de iniciar un juego en un mapa diseñado por su grupo (diferente a todos los que están en el proyecto) que sea mínimo del mismo tamaño del mapa mediumCorners pero que haga que el algoritmo de la pared implementado en el punto anterior falle. Este mapa debe contar además con dos Pacman y una sola galleta. Uno de los Pacman será controlado por uno de los nodos lanzados e implementará inicialmente el algoritmo de la pared. El otro Pacman lo controlará un nodo que recibe órdenes del usuario a través de las teclas del computador. La dinámica del juego consistirá en que el Pacman controlado por el usuario debe ir a rescatar al Pacman controlado por el algoritmo de la pared (ya que éste no puede llegar a la galleta por sí mismo). En el momento en que el Pacman controlado por el humano encuentre al autónomo (por ejemplo esté lo suficientemente cerca), éste último cambiará de comportamiento y seguirá las mismas órdenes del primero. El humano controlará a los dos Pacman hasta la galleta para finalizar el juego. Además de los requerimientos generales de entrega descritos en las instrucciones cada grupo deberá hacer un vídeo mostrando la situación descrita.

*Solución:*

Para este punto se crearon dos códigos en python diferentes. Cada uno muy similar a los implementados en puntos anteriores. El primer código (p5PacmanFlechas.py) generaba el movimiento para el pacman 0 a partir de la interacción con el teclado (awsd). El segundo código (p5PacmanManoDerecha.py) generaba los movimientos para el pacman 1, en este caso moviéndose en primera instancia por la regla de la mano derecha. Aquí, era diseñado un mapa (ver figura 12) que imposibilitará al pacman 1 de alcanzar la única galleta del mapa y evitar, por tanto, que el juego terminase.

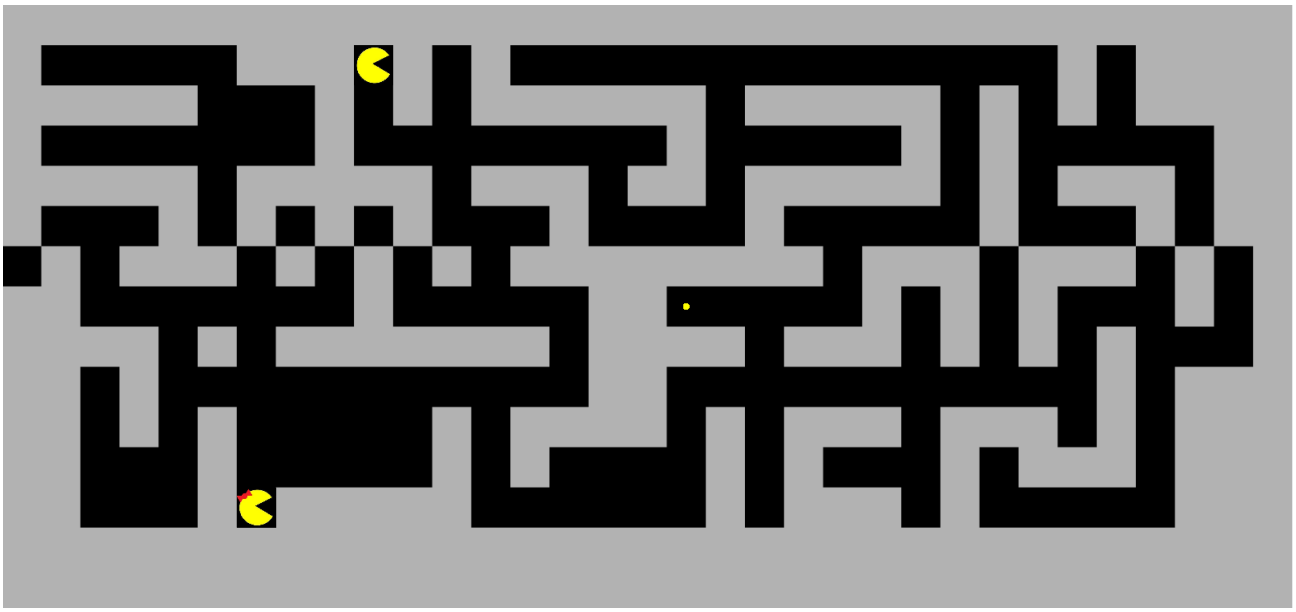

Figure 11: Mapa generado para el punto 5, de manera que el pacman 1, guiado por el algoritmo de la pared, nunca alcanzara la galleta.

Una vez se inicia el juego en el mapa creado (mapaPunto5.lay), se lanzan los códigos en terminales diferentes,

creando cada uno un nodo independiente. Para el caso del nodo *controlador_5_MoverP5*, este solo publica en el tópico *pacmanActions0*, sin suscribirse a ningún tópico. Para el caso del nodo *controlador_5_DerechaP5*, este publica en el tópico *pacmanAction1* y se suscribe a los tópicos *pacmanCoord0* y *pacmanCoord1*. El diagrama entero de nodo se puede evidenciar a continuación:

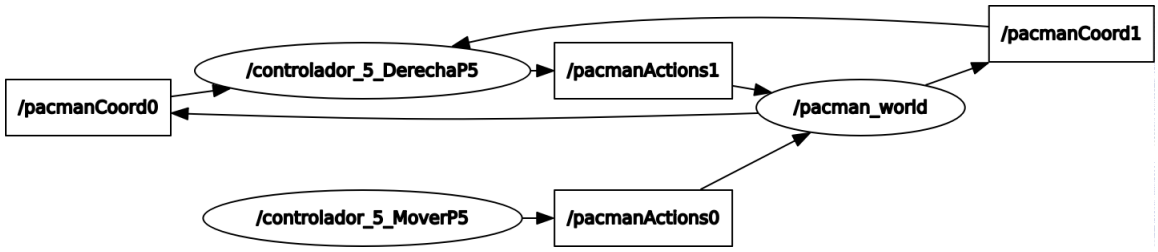


Figure 12: Configuración de nodos en el punto 5. Se evidencia la presencia del nodo ros-pacman, junto con los nodos que controlan los dos pacmans.

Los códigos fueron diseñado de manera de que al acerca el pacman 0 al pacman 1 a una distancia menor de 1, tanto en x como en y, el pacman 1 dejara de implementar el algoritmo de la mano derecha, y por el contrario empezara a determinar las acciones que permitieran seguir al pacman 0. El código p5MoverFlechas.py es el mismo que el código p1MoverPacman.py (ver figura 2), con la única diferencia que al soltar la tecla, método on_release(), se quitaba la linea que dejaba al pacman sin movimiento (acción = 4 era borrada). En la figura 13 se puede evidenciar el diagrama de bloques del código p5PacmanManoDerecha.py. Para el primer código, se tiene una tasa de 10Hz y para el segundo código es inicialmente (1/0.15) Hz.

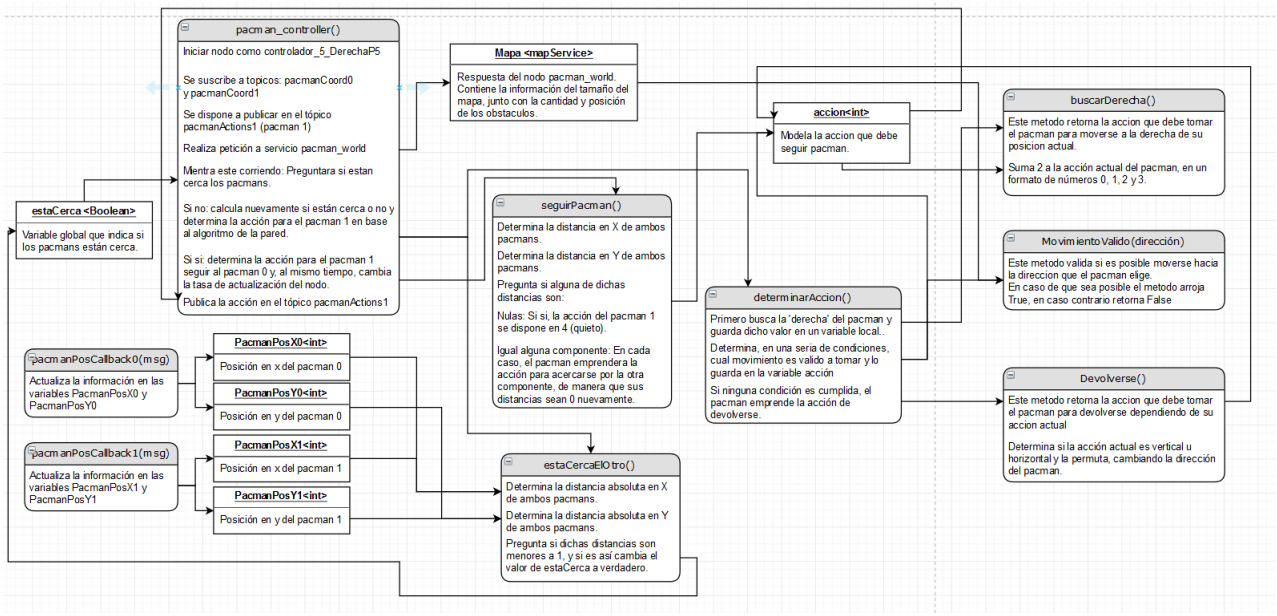


Figure 13: Diagrama de bloques del código p5PacmanManoDerecha.py

Este último código tiene el mismo funcionamiento que el implementado en el punto 4. La única diferencia radica en que en cada ciclo del nodo, se pregunta y determina si ambos pacmans están cerca o no. Si no están cerca las acciones seguirán igual, implementadas por el algoritmo de la pared. Pero si por el contrario, si están cerca, se lanza una función que determina las acciones para seguir al pacman 0. Dichas acciones se determinan con la siguiente lógica: Primero se determina la distancia relativa en $X$ y $Y$ de los pacmans. A partir de esta se determina si dichas distancias son nulas ($\Delta X = 0$ y $\Delta Y = 0$), en cuyo caso se deja al pacman 1 inmóvil (acción = 4). En caso de que alguna distancia difiera, se determina que componente ($X$ o $Y$) es nula, y se programa al código de manera que el pacman se acerque por la otra componente. Por ejemplo, si se tiene $\Delta X = 0$, y se tiene que $\Delta Y = Y_0 - Y_1 > 0$ se emprende la acción de acercase subiendo en posición. Esta misma lógica para las 4 posibles combinaciones. Cabe aclarar en este punto, que la tasa de actualización del nodo es cambiada, a 10Hz, debido a que se quiere que el pacman 1 este lo más 'atento' a la posición del pacman 0, y no permite la posibilidad de perderse en el movimiento

El juego finaliza cuando el pacman 0, movilizado por las acciones del jugador, alcanza la única galleta del mapa.