

Introducción a Ms. Pac-Man % Ingeniería de Comportamientos Inteligentes %  
Juan Antonio Recio García (jareciog@fdi.ucm.es)

## Resumen

El objetivo de este tutorial es conocer el entorno de simulación de Ms. Pac-Man.

## Ms. Pacman

Ms. Pac-Man es un juego distribuido a partir de 1981 donde la señora Pac-Man tiene que conseguir el máximo número de puntos huyendo de los fantasmas Blinky, Pinky, Inky y Clyde.



Figure 1: Captura de MsPacMan

Las reglas son bastante sencillas. El objetivo del personaje es comer todos los puntos -pills- de la pantalla, momento en el que se pasa al siguiente nivel o pantalla. Sin embargo, cuatro fantasmas Blinky, Pinky, Inky y Clyde, recorren el laberinto para intentar capturar a Pac-Man. Estos fantasmas son, respectivamente, de colores rojo, rosa, cian y naranja. En el juego original fantasmas no son iguales, así mientras Blinky es muy rápido, y tiene la habilidad de encontrar a Pac-Man en el escenario, Inky es muy lento y muchas veces evitará el encuentro con Pac-Man.

Hay un “pasillo” a los costados del laberinto que permiten a Pac-Man o sus enemigos transportarse al costado opuesto (sale por la derecha y reingresa por la izquierda, o viceversa). Cuatro puntos más grandes de lo normal situados cerca de las esquinas del laberinto nombrados en inglés «Power Pills» (que en español lo han traducido en diversas formas como píldoras mágicas o de poder, bolas de energía o simplemente punto de poder), proporcionan a Pac-Man, durante un tiempo limitado, la habilidad de comerse él a los monstruos (todos ellos se vuelven azules mientras Pac-Man tiene esa habilidad), tras lo cual todo vuelve a ser como al principio.

Después de haber sido comidos por Pac-Man, los fantasmas se regeneran en «casa» (una caja situada en el centro del laberinto).

## Simulador de MsPacMan

En las prácticas utilizaremos una adaptación del simulador creado para la competición **Ms. Pac-Man vs Ghosts** que se celebra anualmente en la Conference on Computational Intelligence and Games.

Este simulador tiene las siguientes características:

- Ejecución con o sin interfaz gráfica.
- Ejecución con límite de tiempo por tick (40ms).
- Posibilidad de crear comportamiento tanto global como individual para los fantasmas.
- Al contrario que en el juego original tanto MsPacMan como los fantasmas se mueven a la misma velocidad.
- Visibilidad parcial.
- Repetición de partidas.
- Generación de logs.
- Generación de estadísticas.
- Cuatro niveles distintos.

Se deberá utilizar el simulador publicado en el CV de la asignatura.

## Reglas del juego en el simulador

- Cada pill comida suma 10 puntos.

- El primer fantasma comido vale 200 puntos. Dicha puntuación se dobla para el siguiente fantasma. De forma que los consiguientes fantasmas comidos valen 400, 800 y 1600 puntos.
- Dado un tablero con  $n$  pills, la máxima puntuación es:  $10n + 4x(200 + 400 + 800 + 1600)$ .

## Clase Executor

Es la clase encargada de ejecutar la simulación del juego. Contiene métodos para ejecutar el juego con las siguientes características:

- Juego normal con límite de tiempo total.
- Juego con límite de tiempo para la toma de decisión (40ms).
- Juego con límite de tiempo optimizado. No espera los 40ms si el controlador ha terminado antes.
- Juego con grabación.
- Repetición de juego grabado.
- Experimento con varias repeticiones seguidas.



Figure 2: Métodos públicos de la clase Executor

Dependiendo del método ejecutado, se devolverá o bien la puntuación obtenida o una serie de estadísticas sobre puntuaciones. Las estadísticas se guardan en el objeto Stats, que ofrece información adicional como la media, máximo, mínimo,

etc. Para la evaluación de las prácticas se realizarán varias simulaciones de partidas y se utilizará la puntuación media obtenida.

### Creación del Executor

El ejecutor se crea con el objeto `Executor.Builder` que permite configurar las distintas opciones:

- `setPacmanPO` y `setGhostPO` establecen la visibilidad parcial tanto para `MsPacMan` como para los fantasmas. `setSightLimit` establece el límite de visibilidad y `setPOType` el tipo (línea recta, radial, ...).
- `setGhostsMessage` y `setMessenger` habilitan los mensajes entre fantasmas.
- `setVisual` activa la interfaz gráfica.
- `setScaleFactor` establece el factor de escala de la interfaz gráfica.
- `setTickLimit` establece el tiempo total de juego
- `setTimeLimit` establece el tiempo disponible por cada controlador para devolver el siguiente movimiento.

Ejemplo:

```
Executor executor = new Executor.Builder()
    .setTickLimit(4000)
    .setGhostPO(false)
    .setPacmanPO(false)
    .setGhostsMessage(false)
    .setVisual(true)
    .setScaleFactor(3.0)
    .build();
```

### Controladores

El comportamiento de Ms. Pac-Man se define implementando la clase `pacman.controllers.PacmanController` que obliga a implementar el método `public MOVE getMove(Game game, long timeDue)` llamado en cada ciclo de la simulación. Recibe información sobre el juego y el tiempo límite para devolver el movimiento (`MOVE`). Los movimientos posibles son: `UP`, `RIGHT`, `DOWN`, `LEFT` y `NEUTRAL`.

Por ejemplo, una Ms. Pac-Man con movimientos aleatorios se implementaría así:

```
public final class MsPacManRandom extends PacmanController {
    private Random rnd = new Random();
    private MOVE[] allMoves = MOVE.values();

    @Override
    public MOVE getMove(Game game, long timeDue) {
```

```

        return allMoves[rnd.nextInt(allMoves.length)];
    }
}

```

Por su parte la implementación del comportamiento de los fantasmas se realiza extendiendo la clase abstracta `pacman.controllers.GhostController`. Esta es la interfaz a implementar si queremos utilizar un único controlador para todos los fantasmas y devuelve un mapa donde se define el movimiento para cada fantasma (`EnumMap<GHOST, MOVE>`).

En caso de quere implementar un comportamiento independiente para cada fantasma utilizaremos `pacman.controllers.MASController` que se configura con distintos `IndiviudalGhostControllers`. Por ahora no utilizaremos estas clases.

El código de un controlador de fantasmas con movimientos aleatorios sería el siguiente:

```

public final class GhostsRandom extends GhostController {
    private EnumMap<GHOST, MOVE> moves = new EnumMap<GHOST, MOVE>(GHOST.class);
    private MOVE[] allMoves = MOVE.values();
    private Random rnd = new Random();

    @Override
    public EnumMap<GHOST, MOVE> getMove(Game game, long timeDue) {
        moves.clear();
        for (GHOST ghostType : GHOST.values()) {
            if (game.doesGhostRequireAction(ghostType)) {
                moves.put(ghostType, allMoves[rnd.nextInt(allMoves.length)]);
            }
        }
        return moves;
    }
}

```

Cada controlador se ejecuta en una hebra distinta. Dependiendo del modo de simulación la hebra será detenida pasado el tiempo límite (fijado por defecto en 40 ms).

## Reglas de entrega

Como norma general todas las entregas deberán realizarse siguiendo el mismo esquema:

- Se utilizará el paquete `es.ucm.fdi.ici.c2021.practicaX.grupoYY` que contendrá todo el código de los controladores tanto de Ms. Pac-Man como de los fantasmas.

- Por defecto el controlador de Ms. Pac-Man se denominara `MsPacMan` y el de los fantasmas `Ghosts`. En caso de entregar varios controladores se indicará el nombre de cada uno.

### Un primer ejemplo

- Implementa los controladores anteriores en las clases `es.ucm.fdi.ici.c2021.practica0.grupoYY.MsPacMan` y `es.ucm.fdi.ici.c2021.practica0.grupoYY.GhostsRandom`.
- Crea un método `main` en una clase aparte donde se configure la clase `Executor` y a continuación se ejecute el juego:

```
public class ExecutorTest {

    public static void main(String[] args) {
        Executor executor = new Executor.Builder()
            .setTickLimit(4000)
            .setVisual(true)
            .setScaleFactor(3.0)
            .build();

        PacmanController pacMan = new PacManRandom();
        GhostController ghosts = new GhostsRandom();

        System.out.println(
            executor.runGame(pacMan, ghosts, 50)
        );
    }
}
```

En vez de utilizar el controlador que hemos programado puedes controlar a Ms. Pac-Man a través del teclado. Para ello utiliza el controlador:

```
PacmanController pacMan = new HumanController(new KeyBoardInput());
```

### La clase `Game`

La clase `Game` contiene toda la información -quizás demasiada y mal documentada- sobre el estado del juego además de distintas funciones auxiliares.

El juego tiene un método principal `update` encargado de mover las distintas entidades del juego. De esta forma se actualizan las posiciones, pills y power pills restantes.

También incluye métodos para obtener las posibles direcciones de movimiento o calcular distancias y rutas entre dos puntos. Las distancias pueden calcularse con las métricas `PATH` (distancia real) o las aproximaciones `Euclidea`, `Manhatan`

(ver Constants.DM). Para calcular las rutas puede utilizarse A\*, que ya viene precalculado para ahorrar tiempo de cálculo.

### Segundo ejercicio

Investiga a fondo la funcionalidad ofrecida por la clase `Game`.

Crea el controlador `GhostsAggressive` que utiliza el método `game.getApproximateNextMoveTowardsTarget` para dirigirse siempre hacia Ms PacMan, que encontrarás con `getPacmanCurrentNodeIndex`.

Crea el controlador `MsPacManRunAway` que utiliza el método `game.getApproximateNextMoveTowardsTarget` para huir del fantasma más cercano. Para obtener las posiciones de los fantasmas utiliza `getGhostCurrentNodeIndex` y calcula las distancias con cualquiera de las métricas disponibles.

### Tercer ejercicio

Crea las clases `MyFirstMsPacMan` y `MyFirstGhosts` implementando los siguientes comportamientos:

#### MsPacMan

El comportamiento de Ms. Pac-Man consiste en evitar fantasmas que están demasiado cerca o perseguirles si son comestibles. En otro caso moverse a la pill o power pill más cercana.

```
function GETMOVE()
    limit ← 20
    nearestGhost ← GETNEARESTCHASINGGHOST(limit)
    if nearestGhost ≠ NULL then
        return NEXTMOVEAWAYFROM(nearestGhost)
    end if
    nearestGhost ← GETNEARESTEDIBLEGHOST(limit)
    if nearestGhost ≠ NULL then
        return NEXTMOVETOWARDS(nearestGhost)
    end if
    nearestPill ← GETNEARESTPILL()
    return NEXTMOVETOWARDS(nearestPill)
end function
```

Figure 3: Pseudocódigo del comportamiento de MsPacMan para práctica 0

## Ghosts

El comportamiento de los fantasmas también es bastante sencillo. Huirá de Ms. Pac-Man si ésta puede comerle o está cerca de una power pill. En caso contrario se moverá hacia ella o aleatoriamente con probabilidad 90%-10% respectivamente. Fija también un límite para estimar que MsPacMan está cerca de una power pill (p.e. 15).

```
function GETMOVE()
  if GAME.DOESREQUIREACTION() = False then return
  NULL end if
  pacman ← GETPACMANINDEX()
  if ISEDBLE() OR PACMANCLOSETOPPILL() then
    return NEXTMOVEAWAYFROM(pacman)
  end if
  if NEXTFLOAT < 0.9 then
    return NEXTMOVETOWARDS(pacman)
  else
    return NEXTRANDOMMOVE()
  end if
end function
```

Figure 4: Pseudocódigo del comportamiento de los fantasmas para práctica 0