Práctica Recuperación – SI

Óscar De la torre López
oldelatorre@esei.uvigo.es
Estudiante del PCEO en Ingeniería informática y Administración de Empresas en la Universidad de Vigo.

Angel Lillo Amaro
alamaro@esei.uvigo.es
Estudiante de Ingeniería informática en la Universidad de Vigo.

RESUMEN

Se requiere para esta entrega final de la asignatura ,Sistemas Inteligentes, un player que juegue de manera autónoma e independiente y que sea capaz de realizar las máxima puntuación.

Para ello se maximizará las acciones del jugador buscando siempre la mejor jugada, es decir, la que más puntos otorgue y más casillas pinte de su color.

El objetivo según lo que se ha implementado es que nuestro player sea capaz de ganar a los diferentes players implementados por el resto de grupos de trabajo de la asignatura.

En esta entrega hay que añadir a las anteriormente hechas nuevos tipos de fichas especiales que se crearán intercambiando las fichas especiales ya hechas en las prácticas anteriores.

INTRODUCCIÓN

- El juego Candy Crush ESEI consta de 3 agentes: player1, player2 y judge.

- En este juego lo que se pretende es que los dos players compitan de manera autónoma decidiendo por sí mismos cuál es la jugada que mejor conviene en cada momento y que sea de manera óptima , ya que cualquier movimiento inútil o poco eficiente podría desembocar en una gran oportunidad para el rival para sumar puntos.

- No todo depende de nuestras jugadas o de las jugadas del rival si no que también contamos con el factor suerte, porque la caída de las fichas al rellenar el tablero, puede significar una gran suma de puntos si se dan ciertas combinaciones como por ejemplo explotar un comprador.

- El juego consta de tres niveles y para ganarlos se tienen que dar estas condiciones:

    - En el nivel 1, en menos de Mov1 movimentos:

        - No hay obstáculos.
        - Celdas sin dueño.
        - Llegar a 150 puntos.
        - En caso de que ambos players lleguen a 150 puntos, el que consiga mayor puntuación.
        - En caso de que ninguno llegue a 150 puntos, no habrá ganador.

    - En el nivel 2, en menos de Mov2 movimientos:

        - Hay obstáculos.
        - Celdas sin dueño.
        - Llegar a 200 puntos.
        - En caso de que ambos players lleguen a 200 puntos, el que consiga mayor puntuación.
        - En caso de que ninguno llegue a 200 puntos, no habrá ganador.

    - En el nivel 3, en menos de Mov3 movimientos:

        - Hay obstáculos.
        - Celdas con dueño.
        - Llegar a 250 puntos y domine totalmente el tablero.
        - En caso de no conseguirlo, ganará el que domine más celdas.
        - En caso de empate a celdas el que obtenga mayor puntuación.
        - Si hubiera empate también en esto ganaría el segundo jugador,

- Ganará el agente que gane más partidas (10 partidas). Una partida será ganada cuando se consiga ganar en un mayor número de niveles que el contrario; en caso de igualdad ganará aquel agente que consiga más puntos totales (suma de los puntos alcanzados en cada nivel).

SECCIONES

A- Juez

1. Creencias Iniciales

Las creencias que se han añadido para esta práctica son las siguientes:

- **comprobarPatronesEsp(Color,X1,Y1,X2,Y2,Tipo1,Tipo2,Pattern):** Creado para comprobar los patrones especiales, debido a las nuevas combinaciones de fichas que se permiten en esta práctica de recuperación.

2. Objetivos

No hay objetivos.

3. Planes

- **+!checkFinalWinner(W1,W2,W3) :** Recibe por parámetro los ganadores de cada nivel y determina el ganador de la partida. Si algún jugador recibido por parámetro ha ganado 2 de los 3 niveles se establece como ganador.

- **+!showPoints:** Muestra por pantalla la puntuación de ambos jugadores.

- **+!checkLevelWinner:** Comprueba que jugador ha ganado el nivel actual en función de cual ha obtenido mayor puntuación.

- **+!checkWinnerLevel3:** Comprueba que jugador ha ganado el nivel 3.

- **+!generateSpecialSteak:** Se añade el owner de la posición X,Y dada por parámetro para que la nueva creencia tenga el mismo valor para owner.

- **ownerNumber(Owner,OwnerNumber):** almacena el valor del Owner que debe tener las fichas del propio jugador.

- **ownerName(Owner,OwnerName) :** almacena el nombre del Owner que debe tener las fichas del propio jugador.

B- Player

- **Propietario:** almacena el valor del Owner que debe tener las fichas del propio jugador.

- **propietarioRival:** almacena el valor del Owner que debe tener las fichas del jugador rival.

La idea principal de nuestra estrategia es que conquiste el mayor número de celdas posibles para el nivel 3, y una variante de la misma para el level 1 y 2.

La estrategia se basa en buscar todas las figuras del tablero, mientras las fichas sean del jugador que le toca mover. Para cada figura encontrada, se llama a un plan encargado de contar el número de celdas o puntos que se obtendrían con la hipotética explosión de dicha figura.

El plan **Contar1**, recibe todas las coordenadas de la figura, la ficha a mover y la dirección del movimiento. Éste a su vez llama a **sumadorParcial**, que, para cada posición de la figura, calcula el número de celdas que se conquistarían si esta posición explotase. Se prioriza el robo de celdas al rival, para ello si se conquista una celda del rival son 2 puntos, si no pertenece a nadie 1 punto y si pertenece al player 0.05 puntos.

El plan **Contar2**, recibe todas las coordenadas de la figura, la ficha a mover y la dirección del movimiento. Éste a su vez llama a **sumadorParcial2**, que, para cada posición de la figura, calcula el número de puntos que se obtendrían.

El plan **Contar1** y **Contar2** almacenan el movimiento si el contador actual es mayor que el anterior, es decir, si la figura actual conquistaría más celdas que la anterior, entonces, al final de la ejecución tendríamos almacenado de la creencia siguiente el mejor movimiento posible.

1. Creencias Iniciales

- **dir(dir,num):** Aquí se guarda la relación que hay entre los valores dir y num. El valor dir viene dado por los strings “up”, “down”, “left”, “right” y los valores de num están comprendidos entre 0 y 3.

- **propietario(N):** Guarda el número que representa al propio jugador en la recreación del juego.

- **propietarioRival(N):** Guarda el número que representa al jugador rival en la recreación del juego.

- **contador1(N):** Guarda el valor de la jugada que se está a analizar.

- **contador2(N):** Guarda el valor de la mejor jugada analizada.

- **bandera(N):** Guarda valor 0 o 1 e indica si se encontrado una jugada con figura o no.

- **direccion(N):** Guarda un valor entre 0 y 3 que indica la dirección del próximo movimiento a realizar.

- **siguiente(moverDesdeEnDireccion(pos(X,Y),Dir)):** Guarda la estructura a mandar al juez del siguiente movimiento a realizar.

2. Objetivos

No hay objetivos.

3. Planes

+!comprobarQuintuple: Realiza la comprobación de la posibilidad de realizar líneas de 5 fichas en vertical u horizontal con la posición de la ficha a mover con el valor owner que le corresponde al jugador que está jugando. Para esto realizamos los siguientes pasos:

1. Realizamos 2 **.findall** (1 **.findall** por cada una de las orientaciones de la figura) donde guardamos en las listas **ListaQuintupleH** y **ListaQuintupleV** la estructura **celda(X,Y)** que sería la posición donde acabaría la ficha que queremos mover.

2. Realizamos una comprobación del tamaño de las listas. Para cada una de las listas cuyo tamaño sea mayor que 0 se realiza un bucle for para todos los miembros de la lista, donde realizamos un test de las posiciones fijas de la figura y luego comprobamos con un if si la posición de la ficha a mover está arriba o abajo, en el caso de las horizontales, o si está a la derecha o izquierda, en el caso de las verticales. En cada uno de estos if se comprueba los valores de la ficha localizada y el valor de dirección correspondiente para colocarla en el sitio correspondiente. Se lanza **!contar1** con los valores correspondientes (se explica más adelante).

3. Se lanza el plan **!comprobarT**.

+!comprobarT: Realiza la comprobación de la posibilidad de realizar figuras T con la posición de la ficha a mover con el valor owner que le corresponde a el jugador que está jugando. Para esto realizamos los siguientes pasos:

1. Realizamos 4 **.findall** (1 **.findall** por cada una de las orientaciones de la figura) donde guardamos en las listas **ListaTAbajo**, **ListaTArriba**, **ListaTIzquierda** y **ListaTDerecha** la estructura **celda(X,Y)** que sería la posición donde acabaría la ficha que queremos mover.

2. Realizamos una comprobación del tamaño de las listas. Para cada una de las listas cuyo tamaño sea mayor que 0 se realiza un bucle for para todos los miembros de la lista, donde realizamos un test de las posiciones de la figura (como cada sentido de la T las tenemos en listas independientes sabemos la posición de cada ficha). Se comprueba el valor de dirección correspondiente para colocar la ficha a mover en el sitio correspondiente. Se lanza **!contar1** con los valores correspondientes (se explica más adelante).

3. Se lanza el plan **!comprobar4**.

+**!comprobar4**: Realiza la comprobación de la posibilidad de realizar líneas de 4 fichas en vertical u horizontal con la posición de la ficha a mover con el valor owner que le corresponde a el jugador que está jugando. Para esto realizamos los siguientes pasos:

1. Realizamos 4 .f**indall** (2 .**findall** para cada una de las orientaciones ya que se puede formar por cualquiera de las 2 fichas centrales) donde guardamos en las listas **Lista4HDerecha** y **Lista4HIzquierda**, en el caso de las horizontales, y en las listas **Lista4YArriba** y **Lista4YAbajo**, en el caso de las verticales, la estructura **celda(X,Y)** que sería la posición donde acabaría la ficha que queremos mover.

2. Realizamos una comprobación del tamaño de las listas. Para cada una de las listas cuyo tamaño sea mayor que 0 se realiza un bucle for para todos los miembros de la lista, donde realizamos un test de las posiciones fijas de la figura y luego comprobamos con un if si la posición de la ficha a mover está arriba o abajo, en el caso de las horizontales, o si está a la derecha o izquierda, en el caso de las verticales. En cada uno de estos if se comprueba los valores de la ficha localizada y el valor de dirección correspondiente para colocarla en el sitio correspondiente. Se lanza **!contar1** con los valores correspondientes (se explica más adelante).

3. Se lanza el plan **!comprobarCuadrado**.

+**!comprobarCuadrado:** Realiza la comprobación de la posibilidad de realizar cuadrados en el tablero con la posición de la ficha a mover con el valor owner que le corresponde a el jugador que está jugando. Para esto realizamos los siguientes pasos:

1. Realizamos 4 **.findall** (1 **.findall** por cada una de las esquinas del cuadrado) donde guardamos en las listas **ListaCuadradoUpI, ListaCuadradoUpD, ListaCuadradoDI y ListaCuadradoDD** la estructura **celda(X,Y)** que sería la posición donde acabaría la ficha que queremos mover.

2. Realizamos una comprobación del tamaño de las listas. Para cada una de las listas cuyo tamaño sea mayor que 0 se realiza un bucle for para todos los miembros de la lista, donde realizamos un test de las posiciones fijas de la figura y luego comprobamos con un if la posición de la ficha a mover ya que en cada esquina del cuadrado se puede completar desde dos direcciones diferentes.
   En cada uno de estos if se comprueba los valores de la ficha localizada y el valor de dirección correspondiente para colocarla en el sitio correspondiente. Se lanza **!contar1** con los valores correspondientes (se explica más adelante).

3. Se lanza el plan **!comprobarTriple.**

+!comprobarTriple: Realiza la comprobación de la posibilidad de realizar líneas de 3 fichas en el tablero con la posición de la ficha a mover con el valor owner que le corresponde a el jugador que está jugando. Para esto realizamos los siguientes pasos:

1. Realizamos 6 **.findall** (3 **.findall** por cada una de las orientaciones de la figura) donde guardamos en las listas **Lista3HD, Lista3HMedio, Lista3HI,** en el caso de las horizontales, **Lista3VAbajo, Lista3VM y Lista3VArriba**, en el caso de las verticales, la estructura **celda(X,Y)** que sería la posición donde acabaría la ficha que queremos mover.

2. Realizamos una comprobación del tamaño de las listas. Para cada una de las listas cuyo tamaño sea mayor que 0 se realiza un bucle for para todos los miembros de la lista, donde realizamos un test de las posiciones fijas de la figura y luego comprobamos con un if la posición de la ficha a mover ya que en los laterales de la figura puede ser completada desde 3 direcciones diferentes y la del medio puede ser completada desde 2 diferentes. En cada uno de estos if se comprueba los valores de la ficha localizada y el valor de dirección correspondiente para colocarla en el sitio correspondiente. Se lanza **!contar1** con los valores correspondientes.

3. Se comprueba el valor de la creencia bandera. Si este valor es 0 se lanza **!movimientoFigura**. Si este valor es 1 se hace un test de **siguiente(moverDesdeEnDireccion(pos(X,Y),Dir))** que contiene el mejor movimiento analizado y lo envía. Resetea los valores de **bandera(X)** y **contador2(X)**.

+! movimientoFigura: Realiza la comprobación de la posibilidad de realizar líneas de 5 fichas en vertical u horizontal. Para esto realizamos los siguientes pasos:

1. Realizamos 2 **.findall** (1 **.findall** por cada una de las orientaciones de la figura) donde guardamos en las listas **ListaQuintupleH** y **ListaQuintupleV** la estructura **celda(X,Y)** que sería la posición donde acabaría la ficha que queremos mover.

2. Realizamos una comprobación del tamaño de las listas. Para cada una de las listas cuyo tamaño sea mayor que 0 se realiza un bucle for para todos los miembros de la lista, donde realizamos un test de las posiciones fijas de la figura y luego comprobamos con un if si la posición de la ficha a mover está arriba o abajo, en el caso de las horizontales, o si está a la derecha o izquierda, en el caso de las verticales. En cada uno de estos if se comprueba los valores de la ficha localizada y el valor de dirección correspondiente para colocarla en el sitio correspondiente. Se lanza **!contar2** con los valores correspondientes.

3. Se lanza el plan **!comprobarT3**.

+!comprobarT3: Realiza la comprobación de la posibilidad de realizar figuras T. Para esto realizamos los siguientes pasos:

1. Realizamos 4 **.findall** (1 **.findall** por cada una de las orientaciones de la figura) donde guardamos en las listas **ListaTAbajo, ListaTArriba, ListaTIzquierda y ListaTDerecha** la estructura **celda(X,Y)** que sería la posición donde acabaría la ficha que queremos mover.

2. Realizamos una comprobación del tamaño de las listas. Para cada una de las listas cuyo tamaño sea mayor que 0 se realiza un bucle for para todos los miembros de la lista, donde realizamos un test de las posiciones de la figura (como cada sentido de la T las tenemos en listas independientes sabemos la posición de cada ficha). Se comprueba el valor de dirección correspondiente para colocar la ficha a mover en el sitio correspondiente. Se lanza **!contar2**.

3. Se lanza el plan **!comprobar4.**

+!comprobar43: Realiza la comprobación de la posibilidad de realizar líneas de 4 fichas en vertical u horizontal. Para esto realizamos los siguientes pasos:

1. Realizamos 4 **.findall** (2 **.findall** para cada una de las orientaciones ya que se puede formar por cualquiera de las 2 fichas centrales) donde guardamos en las listas **Lista4HDerecha** y **Lista4HIzquierda**, en el caso de las horizontales, y en las listas **Lista4YArriba** y **Lista4YAbajo**, en el caso de las verticales, la estructura **celda(X,Y)** que sería la posición donde acabaría la ficha que queremos mover.

2. Realizamos una comprobación del tamaño de las listas. Para cada una de las listas cuyo tamaño sea mayor que 0 se realiza un bucle for para todos los miembros de la lista, donde realizamos un test de las posiciones fijas de la figura y luego comprobamos con un if si la posición de la ficha a mover está arriba o abajo, en el caso de las horizontales, o si está a la derecha o izquierda, en el caso de las verticales. En cada uno de estos if se comprueba los valores de la ficha localizada y el valor de dirección correspondiente para colocarla en el sitio correspondiente. Se lanza **!contar2.**

3. Se lanza el plan **!comprobarCuadrado**.

+!comprobarCuadrado3: Realiza la comprobación de la posibilidad de realizar cuadrados en el tablero.Para esto realizamos los siguientes pasos:

1. Realizamos 4 **.findall** (1 **.findall** por cada una de las esquinas del cuadrado) donde guardamos en las listas **ListaCuadradoUpI**, **ListaCuadradoUpD**, **ListaCuadradoDI** y **ListaCuadradoDD** la estructura **celda(X,Y)** que sería la posición donde acabaría la ficha que queremos mover.
2. Realizamos una comprobación del tamaño de las listas. Para cada una de las listas cuyo tamaño sea mayor que 0 se realiza un bucle for para todos los miembros de la lista, donde realizamos un test de las posiciones fijas de la figura y luego comprobamos con un if la posición de la ficha a mover ya que en cada esquina del cuadrado se puede completar desde dos direcciones diferentes.
   En cada uno de estos if se comprueba los valores de la ficha localizada y el valor de dirección correspondiente para colocarla en el sitio correspondiente. Se lanza **!contar2**.
3. Se lanza el plan **!comprobarTriple**.

+!comprobarTriple3: Realiza la comprobación de la posibilidad de realizar líneas de 3 fichas en el tablero tanto en vertical como en horizontal. Para esto realizamos los siguientes pasos:

1. Realizamos 6 **.findall** (3 **.findall** por cada una de las orientaciones de la figura) donde guardamos en las listas **Lista3HD**, **Lista3HMedio**, **Lista3HI**, en el caso de las horizontales, **Lista3VAbajo**, **Lista3VM** y **Lista3VArriba**, en el caso de las verticales, la estructura **celda(X,Y)** que sería la posición donde acabaría la ficha que queremos mover.
2. Realizamos una comprobación del tamaño de las listas. Para cada una de las listas cuyo tamaño sea mayor que 0 se realiza un bucle for para todos los miembros de la lista, donde realizamos un test de las posiciones fijas de la figura y luego comprobamos con un if la posición de la ficha a mover ya que en los laterales de la figura puede ser completada desde 3 direcciones diferentes y la del medio puede ser completada desde 2 diferentes. En cada uno de estos if se comprueba los valores de la ficha localizada y el valor de dirección correspondiente para colocarla en el sitio correspondiente. Se lanza **!contar2**.
3. Se comprueba el valor de la creencia bandera. Si este valor es 0 se lanza **!movimentoRandom**. Si este valor es 1 se hace un test de **siguiente(moverDesdeEnDireccion(pos(X,Y),Dir)**) que
   contiene el mejor movimiento analizado y lo envía. Resetea los valores de **bandera(X)** y **contador2(X)**.

+!contar1(X1,Y1,Tipo1,X2,Y2,Tipo2,X3,Y3,Tipo3,X4,Y4,Tipo4,X5,Y5,Tipo5,Dir1,X,Y): Este plan se utiliza para cuantificar las casillas que se obtendrían con la realización de una figura. Para esto se le pasan como parámetros las coordenadas de las 5 posibles fichas de las figuras con sus respectivos tipos.

En caso de que la figura cuente con menos de 5 fichas se rellena los parámetros sobrantes con un 6. Para la cuantificación, primeramente, se pone la creencia bandera a 1 y se resetea a 0 la creencia contador1.

Luego se le pasa al plan **!sumadorParcial** cada una de las fichas a evaluar por separado. Al finalizar se comprueba si contador1 tiene mayor valor que contador2, que sería la mejor figura a realizar analizada anteriormente. En caso afirmativo, se guarda el valor de contador1 en contador2 y se actualiza siguiente.

+!contar2(X1,Y1,Tipo1,X2,Y2,Tipo2,X3,Y3,Tipo3,X4,Y4,Tipo4,X5,Y5,Tipo5,Dir1,X,Y): Este plan tiene la misma función que el plan anterior con la salvedad de que en vez de llamar a **!sumadorParcial** llama a **!sumadorParcial2**.

+!sumadorParcial(X1,Y1,Tipo1,Dir1): Este plan va a contar el número de celdas que se obtendrían con la explosión de la coordenada pasada como parámetro. Para ello necesitamos el tipo de la ficha y la dirección del movimiento para determinar la explosión vertical u horizontal en el caso de que sea de tipo“IP”.

El objetivo es buscar el movimiento que capture más celdas, priorizando el robo de celdas al rival. Para ello se usa un contador que se incrementa en función de cada caso, explicado a continuación:

En primer lugar, se consultan las creencias para obtener el Owner del propio jugador y el Owner del rival, estos valores se almacenan en las variables **OwnerPlayer1** y **OwnerPlayer2**.
Para evitar posibles errores todo el código de este plan se va a ejecutar si las coordenadas X1 y Y1 son positivas y no son mayores que el tamaño del tablero (N-1).
A continuación, mediante varios ifs comprobamos el Tipo de ficha para saber si se dará lugar una explosión especial:

Si la ficha es de tipo “IN”:

Se consulta el **Owner** de esa celda, y en función de este valor asignamos un número de puntos, de modo que, si la celda pertenece al rival, sumaremos 2 puntos. Si la celda no tiene propietario sumaremos 1 punto y por último si la ficha nos pertenece sumaremos 0.05 puntos. De este modo conseguimos priorizar un movimiento que robe celdas al rival.

Si la ficha es de tipo “GS”:

Se consulta el **Owner** de la celda, si la celda pertence al rival sumamos 3 puntos (2 por la

obtención de una celda del rival y 1 punto por la posible obtención de una celda al producirse la explosión especial). Si la celda no pertenece a nadie, sumaremos 2 puntos, por último, si la ficha nos pertenece, sumaremos 0.50 puntos ya que la ficha aleatoria que explote puede pertenecer al rival, a nadie o a nosotros mismos.

Si la ficha es de tipo "CO":

Para este caso comprobamos el propietario de las celdas que explotarían y sumamos los puntos en función del propietario.
Para ello comprobamos las fichas superiores que explotarían (X-1, Y-1), (X, Y-1), (X+1, Y-1) y las inferiores (X-1, Y+1), (X, Y+1), (X+1, Y+1). De este modo comprobaremos el propietario en las 6 coordenadas mencionadas.

Comprobamos que la coordenada X y Y no sea el límite del tablero, ya que posteriormente realizamos los test de las posiciones mencionadas, de este modo evitamos que se haga un test de una creencia que no existe. Si X, Y no es el límite del tablero comprobamos el **Owner** de la posición que explotaría y en función de este valor, como se ha explicado anteriormente, se asigna un número de puntos, obteniendo más puntos si la celda pertenece al rival.

Si la ficha es de tipo "IP":

En función de la Dirección obtenida por el parámetro **Dir1**, sabemos si la explosión será vertical o horizontal.

Si la explosión es vertical, es decir, si el valor de **Dir1** es "1" o "0", realizamos tres búsquedas en las creencias mediante un .**findall**.

Búsqueda de las celdas que pertenezcan al rival: El resultado se almacena en la lista **"ListaFilaRival"**. La longitud de esta lista será el número de celdas que pertenecen al rival en la columna que va a explotar. Añadimos al contador, el tamaño de la lista multiplicado por dos, ya que, obtener celdas del rival es lo más prioritario.

Búsqueda de las celdas sin propietario: El resultado se almacena en la lista "**ListaFilaSin"**. La longitud de esta lista será el número de celdas sin propietario en la columna que va a explotar. Añadimos al contador, el tamaño de la lista.

Búsqueda de las celdas que nos pertenecen: El resultado se almacena en la lista **"ListaFilaSin12"**. La longitud de esta lista será el número de celdas que nos pertenecen en la columna que va a explotar. Añadimos al contador, el tamaño de la lista multiplicado por 0.05, ya que, el objetivo es obtener celdas del rival o celdas sin propietario.

Si la explosión es horizontal, es decir, si el valor de **Dir1** es "2" o "3", se repite el proceso explicado, con la diferencia de que la búsqueda se realiza en la fila.

Si la ficha es de tipo "CT":

Se consulta el Color de la **ficha (X, Y)** y se realizan tres búsquedas en las creencias mediante un findall.

Búsqueda de las fichas con el Color consultado y que la celda pertenezca al rival: El resultado se almacena en la lista **"ListaColorRival1"**. La longitud de esta lista será el número de celdas que pertenecen al rival con la ficha de Color dado. Añadimos al contador, el tamaño de la lista multiplicado por dos, ya que, obtener celdas del rival es lo más prioritario.

Búsqueda de las fichas con el Color consultado y que la celda no tenga propietario: El resultado se almacena en la lista **"ListaColorSin1"**. La longitud de esta lista será el número de celdas sin propietario con la ficha de Color dado. Añadimos al contador, el tamaño de la lista.

Búsqueda de las fichas con el Color consultado y que la celda nos pertenezca: El resultado se almacena en la lista **"ListaColorSin14"**. La longitud de esta lista será el número de celdas que nos pertenecen con la ficha de Color dado. Añadimos al contador, el tamaño de la lista multiplicado por 0.05, ya que, el objetivo es obtener celdas del rival o celdas sin propietario.

+!sumadorParcial2(X1,Y1,Tipo1,Dir1): Este plan va a contar el número de puntos que se obtendrían con la explosión de la coordenada pasada como parámetro. Para ello necesitamos el tipo de la ficha y la dirección del movimiento para determinar la explosión vertical u horizontal en el caso de que sea de tipo "IP".

El objetivo es buscar el movimiento que realice más puntos. Para ello se usa un contador que se incrementa en función de cada caso, explicado a continuación:
Para evitar posibles errores todo el código de este plan se va a ejecutar si las coordenadas X1 y Y1 son positivas y no son mayores que el tamaño del tablero (N-1).
A continuación, mediante varios ifs comprobamos el Tipo de ficha para saber si se dará lugar una explosión especial:

Si la ficha es de tipo "IN":
Se suma 1 a **contador1**.

Si la ficha es de tipo “GS”:
Se le suma 5 a **contador1.**

Si la ficha es de tipo “CO”:
Se le suma 12 a **contador1**.

Si la ficha es de tipo “IP”:
En función de la Dirección obtenida por el parámetro **Dir1**, sabemos si la explosión será vertical o horizontal.

Si la explosión es vertical, es decir, si el valor de **Dir1** es “up” o “down”, realizamos una búsqueda en las creencias mediante un **.findall**.

Búsqueda de los obstáculos y posiciones vacías en la fila o columna: El resultado se almacena en la lista **“listaObstacle”**. La longitud de esta lista será el número de celdas sin propietario en la columna que va a explotar. Añadimos al contador, 2 más el size menos el tamaño de la lista.

Si la explosión es horizontal, es decir, si el valor de **Dir1** es “right” o “left”, se repite el proceso explicado, con la diferencia de que la búsqueda se realiza en la fila.

Si la ficha es de tipo “CT”:
Se consulta el Color de la **ficha (X, Y)** y se realizan tres búsquedas en las creencias mediante un **.findall**.

Búsqueda de las fichas con el Color consultado y que la celda no tenga propietario: El resultado se almacena en la lista **“ListaColorSin1”**. La longitud de esta lista será el número de celdas sin propietario con la ficha de Color dado. Añadimos a **contador1** el tamaño de la lista más 8.

+!movimientoRamdon: Genera 3 números aleatorios, 2 de ellos para escoger la posición de la ficha a mover y el restante para escoger la dirección de movimiento. Para escoger la **ficha(X, Y)** mediante random tenemos un número entre 0 y 1 que posteriormente multiplicamos por el tamaño del tablero menos 1 y luego redondeamos para obtener un valor entre 0 y 9.

Para escoger la dirección de movimiento **Dir** utilizamos el mismo método, pero obtenemos valores entre 0 y 3 y consultamos su equivalencia con las creencias almacenadas **Dir**.
Una vez hemos obtenido la posición a mover y la dirección se lo enviamos al juez mediante

un send **moverDesdeEnDireccio**n pasándole por parámetro los valores mencionados anteriormente.

+**puedesMover**: Recibe la notificación por parte del juez para realizar un movimiento. Llama al plan **!comprobarQuintuple**.

+**invalido(fueraTablero,VECES)**: Recibe la notificación por parte del juez de la realización de un movimiento. Llama al plan **!comprobarQuintuple**.

+**!deleteTableroBB**: Realiza un **.abolish** de todas las creencias de tablero.

+**!tryAgain**: Recibe la notificación por parte del juez de intentar un nuevo movimiento. Llama al plan **!comprobarQuintuple**.

+**invalido(fueraTurno,VECES)**: Recibe la notificación por parte del juez de la realización de un movimiento fuera de turno.

+**!valido**: Recibe la notificación por parte del juez de la realización de un movimiento correcto.

C- Tablero
No hemos añadido nada nuevo.

CONCLUSIÓN
En la elaboración de esta práctica no tuvimos demasiados problemas ya que respecto a lo ya hecho no teníamos que realizar demasiadas cosas más, players usamos los nuestros que ya los teníamos de otras prácticas y los perfeccionamos un poco. Por el resto hubo que crear planes nuevos para los nuevos patrones ya que funcionan de forma diferente a los que ya había de las otras prácticas. Alguna cosa de algún patrón nuevo si que nos dió algún problema pero nada importante.

REFERENCES
Web site:
API documentation for Jason (Java-based interpreter for an extended version of AgentSpeak) from http://jason.sourceforge.net/api/