Sistema Multi–Agente para la Resolución de Juegos

Roi Pérez López
rplopez@esei.uvigo.es, dirección de contacto
Alumno de la Escuela Superior de Ingeniería Informática

Martín Puga Egea
mpuga.esei.uvigo@gmail.com, dirección de contacto
Alumno de la Escuela Superior de Ingeniería Informática

RESUMEN

En este proyecto se utilizan agentes para la resolución de juego. El juego consiste en intercambiar las fichas de dos posiciones contiguas de forma que se cumplan las reglas de juego. Se usará un agente que actuará de juez y pedirá, validará y efectuará las jugadas propuestas por los agentes jugador en el orden que correspondan. Los agentes jugador tendrán por su parte el objetivo de realizar jugadas válidas y óptimas, para alcanzar el máximo número de puntos en función de las reglas del juego. El juego estará compuesto de dos niveles. El primero de los niveles representará un tablero relleno de fichas corrientes, en el que se permiten jugadas válidas dentro de los límites del tablero. El segundo nivel añade obstáculos al tablero, que no pueden ser escogidos por el jugador para ser movidos y que además no pueden ser removidos con una explosión ni participan de la caída de las fichas, las fichas tampoco lo pueden atravesar al caer. El juego termina cuando uno de los dos jugadores alcanza el límite de puntos de los dos niveles, cuando un jugador complete 50 jugadas, o cuando ambos hayan sido expulsados de la partida. El juego podrá finalizar tanto en el primer como en el segundo nivel si no se han alcanzado los puntos necesarios en el número de jugadas dado para cada nivel.

INTRODUCCIÓN

En esta segunda entrega del proyecto, con los conocimientos adquiridos de la primera aproximación a la realización de un proyecto en un lenguaje lógico, nos hemos centrado en partir de un código base que nos permitiera escalar de forma consistente el proyecto para cumplir las demandas de la entrega.

Después de conseguir una interfaz gráfica funcional, en el entorno JAVA, y que fuera este quien creara el tablero e iniciara la partida, dejamos esa parte para centrar toda la lógica del proyecto en el juez y el jugador, tratando de paso de conseguir un código lo más similar al paradigma lógico posible.

En un principio dejamos que el jugador mantuviera la aleatoriedad de sus decisiones para centrarnos en que el juez fuera capaz de reconocer los patrones que se recogían en el enunciado del problema. Y que una vez identificados fuera capaz de gestionarlos para eliminarlos del tablero, actualizarlos en el entorno para que se viera visualmente la desaparición del patrón, y actualizar en la base de conocimiento del juez que estas fichas habían desaparecido. Después le llegó el turno a gestionar la caída de las nuevas fichas, de la que también se encargaría el juez, y que luego enviaría la información actualizada del tablero tanto al entorno como a los jugadores.

Con el reconocimiento y gestión de patrones y con la gestión de la caída de las fichas, el siguiente paso fue generar las fichas especiales que estos patrones generaban y hacer que sus particularidades formaran parte de la dinámica del juego.

Una vez conseguido todo esto, le llegó el turno al jugador de tratar de hacer la mejor partida posible. Para ello utilizamos la lógica de reconocimiento de patrones del juez, y la asignación de puntos a los patrones y a las explosiones generadas por las fichas especiales que contienen. Con estos datos el jugador pasó a ser capaz de reconocer dentro del tablero qué jugada es la que le reporta un mayor número de puntos, atendiendo al patrón que realizan, a las explosiones de fichas especiales que contienen, y a las fichas que estas explosiones destruyen (incluídas las explosiones en cadena).

Para finalizar, implementamos el segundo nivel, la gestión de los puntos de ambos jugadores y la elección en base a estos del ganador de la partida.

FUNCIONAMIENTO DEL PROGRAMA

El programa empieza por el primer nivel. El tablero se encarga de generar el tablero del primer nivel, mandárselo al juez y de avisar a éste de que puede iniciar la partida.

Una vez el juez recibe el tablero y la orden de iniciar la partida, le manda a ambos jugadores el contenido del tablero y avisa al primero de ellos de que puede comenzar a jugar.

El jugador, ya con el conocimiento del tablero, busca dentro de este la jugada que maximice su puntuación, reconociendo los patrones que hay, y escogiendo el que más puntos genere y/o más puntos añada con la explosión de fichas especiales. Una vez tiene las coordenadas y la dirección de la ficha que más puntos genera se la envía al juez para que la valide y la ejecute.

Si la jugada es válida, el juez la ejecuta y le manda los datos necesarios al tablero para que actualice también la vista, en caso contrario y en función de la infracción, no ejecuta la jugada y notifica al jugador en cuestión. Una vez la jugada ha sido válida y se ha registrado como tal, se añaden los puntos que correspondan al jugador que la haya hecho y se pasa al turno del siguiente jugador.

La dinámica del juego continua hasta que se consiguen los puntos necesarios para pasar de nivel, en cuyo caso se genera el nuevo tablero y se inicia el siguiente nivel, que sumará puntos al primero para decidir al ganador. Si no se consiguen los puntos necesarios antes de ejecutar las jugadas el juego termina en el primer nivel.
Una vez termina el segundo nivel, se suman los puntos de ambos jugadores y se termina designando al ganador y dando por finalizado el juego.

CREENCIAS INICIALES

judge

jugadasRestantes(100).

Almacena un contador con las jugadas restantes que quedan en el programa.

jugadasPlayer(player1,0).
jugadasPlayer(player2,0).

Contienen un contador con el número de jugadas válidas que llevan ejecutadas cada uno de los dos jugadores.

turnoActual(player1).

Establece el jugador al que le va a tocar jugar a continuación.

turnoActivado(0).

Establece con un 0 que no se le ha dado permiso para jugar al jugador actual, y con un 1 que tiene permiso para jugar.

fueraTablero(0).

Contiene un contador que establece el número de veces que se ha salido del tablero, permitirá pasar el turno del jugador si se ha equivocado varias veces.

fueraTurno(player1,0).
fueraTurno(player2,0).

Contiene un contador con la información del número de veces que un jugador ha jugado fuera de turno, si se ha superado el umbral fijado, el jugador será expulsado.

jugadorDescalificado(player1,0).
jugadorDescalificado(player2,0).

Establecerá con un 0 que el jugador sigue jugando, y con un 1 que ha sido descalificado y que por tanto no participa en lo que resta de partida.

recursivityFlag(1).

Flag utilizado como herramienta para implentar un mecanismo de recursividad en el algoritmo de explosion-gravedad-rellenado

explosionFlag(1).

Cuando el jugador provoca directamente una explosion, el flag se activa. Si las explosiones se producen por caidas, permanece desactivado.

dualExplosionFlag(0).

Flag que permite controlar situaciones de explosión de doble patron

level(1).

Indica el nivel en el que estamos, por defecto empieza en uno.

obstacles(10).

Numero de obstaculos a generar

limitPoints(1,N).

Puntuación a obtener para ganar el primer nivel.

limitPoints(2,N).

Puntuación a obtener para ganar el segundo nivel.

points(1,player1,0).
points(1,player2,0).
points(2,player1,0).
points(2,player2,0).

Puntuacion de cada jugador en cada nivel.

levelWinner(1,none).
levelWinner(2,none).

Ganador del nivel.

finalWinner(none).

Ganador al final del juego.

endGame(0).
Flag que indica la finalización del juego al terminar todos los niveles

specialSteak("4inLineH",ip).
specialSteak("4inLineW",ip).
specialSteak("5inLineH",ct).
specialSteak("5inLineW",ct).
specialSteak("Square",gs).
specialSteak("T",co).

Correspondencia entre el patron que explosiona y la ficha especial que se genera.

generationPoints(in,0).
generationPoints(ip,2).
generationPoints(gs,4).
generationPoints(co,6).
generationPoints(ct,8).

Correspondencia entre la ficha que explota y la puntuación que genera.

REGLAS

player

pensarJugadaAleatoria(X1,Y1,Dir)

Devuelve un valor aleatorio de X1 e Y1, que serán las coordenadas de la ficha que se desea jugar por parte del jugador, y Dir, que será la dirección en la que se moverá y que la escoge el jugador en función de si en esa dirección no hay una ficha del mismo color. Si está rodeada de fichas del mismo color devolverá “up” por defecto, para provocar la restricción en el juez y que le mande mover de nuevo.

comprobarPatrones(Color,X,Y,StartsAtX,StartAtY,Direction,Pattern)

Comprueba a partir de Color, X e Y si hay un patrón en esas coordenadas. Devuelve la posición en la que empieza ese patrón (StartAtX y StartAtY van a ser la primera posición del patrón, a la izq para los patrones horizontales, arriba para los verticales, arriba a la izquierda para el cuadrado y donde une la T para el patrón en T). Devuelve también el Patrón de más valor que encuentra como una cadena de texto en Pattern. Direction sólo se usa para el patrón en T, e indica la dirección de la T, si está de pie, dada la vuelta, hacia la derecha o hacia la izquierda. Comprueba por orden de más puntos a menos todos los patrones listados abajo, y devuelve los datos del primero que sea cierto.

pattern5inLineH(Color,X,Y,StartsAtX,StartAtY)
pattern5inLineW(Color,X,Y,StartsAtX,StartAtY)
patternT(Color,X,Y,Direction)
patternSquare(Color,X,Y,StartsAtX,StartAtY)
pattern4inLineH(Color,X,Y,StartsAtX,StartAtY)
pattern4inLineW(Color,X,Y,StartsAtX,StartAtY)
pattern3inLineW(Color,X,Y,StartsAtX,StartAtY)
pattern3inLineH(Color,X,Y,StartsAtX,StartAtY)

Cada una de estas reglas comprueba si el patrón se cumple, y en qué coordenadas empieza en caso de que así sea.

judge

movimientoValido(pos(X,Y),Dir).

Valida que haya una posición del tablero con esas coordenadas y que el movimiento es válido.

validacion(X,Y,Dir,COrigen).

Valida que haya una posición del tablero con las coordenadas de la casilla a la que se va a mover la ficha y que las dos fichas que se intercambian no tienen el mismo color.

mismoColor(COrigen,CDestino).

Comprueba que el color de la ficha de origen es igual al de la ficha de destino.

direccionCorrecta(pos(X,Y),Dir).

Comprueba que haya una casilla con las coordenadas de la ficha y con las coordenadas a las que se va a mover.

movimiento(X,Y,Dir) .

Comprueba que la casilla a la que se va a mover unifica con una presente en el tablero.

colorFichasDistintos(pos(X,Y),Dir).

Valida que haya una posición del tablero con esas coordenadas y que el movimiento es válido.

plNumb(A,PlNumb).

Recibe el nombre de uno de los players y devuelve el número entero que le corresponde para el valor Own de la celda.

nextPosition(P1,P2,NX,NY,Dir).

Da como parámetros las coordenadas de una celda (P1,P2) , y la dirección en la que se pretende mover su ficha, y devuelve las coordenadas de la celda a la que se mueve (NX,NY).

comprobarPatrones(Color,X,Y,StartsAtX,StartAtY,Direction,Pattern)

Comprueba a partir de Color, X e Y si hay un patrón en esas coordenadas. Devuelve la posición en la que empieza ese patrón (StartAtX y StartAtY van a ser la primera posición del patrón, a la izq para los patrones horizontales, arriba para los verticales, arriba a la izquierda para el cuadrado y donde une la T para el patrón en T). Devuelve también el Patrón de más valor que encuentra como una cadena de texto en Pattern. Direction sólo se usa para el patrón en T, e indica la dirección de la T, si está de pie, dada la vuelta, hacia la derecha o hacia la izquierda. Comprueba por orden de más puntos a menos todos los patrones listados abajo, y devuelve los datos del primero que sea cierto.

pattern5inLineH(Color,X,Y,StartsAtX,StartAtY)
pattern5inLineW(Color,X,Y,StartsAtX,StartAtY)
patternT(Color,X,Y,Direction)
patternSquare(Color,X,Y,StartsAtX,StartAtY)
pattern4inLineH(Color,X,Y,StartsAtX,StartAtY)
pattern4inLineW(Color,X,Y,StartsAtX,StartAtY)
pattern3inLineW(Color,X,Y,StartsAtX,StartAtY)
pattern3inLineH(Color,X,Y,StartsAtX,StartAtY)

Cada una de estas reglas comprueba si el patrón se cumple, y en qué coordenadas empieza en caso de que así sea.

```
emptyUnder(X,Y) :- tablero(celda(X,Y,_),ficha(_,_)) & tablero(celda(X,Y+1,_),e).
emptyLeft(X,Y) :- tablero(celda(X,Y,_),ficha(_,_)) & tablero(celda(X-1,Y,_),e).
```

Reconocimiento de posicion vacia bajo una ficha, a su izquierda o su derecha para el algoritmo de caida

```
levelWinner(P1,P2,Winner) :- P1 > P2 & Winner = player1.
levelWinner(P1,P2,Winner) :- P2 > P1 & Winner = player2.
levelWinner(P1,P2,Winner) :- P1 = P2 & Winner = draw.
```

Asigna los ganadores en función de P1 (Puntos jugador 1) y P2 (Puntos jugador 2)

PLANES

player

+puedesMover[source(judge)]

Se recibe del juez, y llama al plan de !realizarJugada.

+!realizarJugada

Llama al objetivo !pensarJugada, y luego comunica la jugada en !comunicarJugada.

+!comunicarJugada

Comunica al juez que quiere moverse desde las coordenadas X,Y en la dirección Dir. Estos argumentos los recoge de su base del conocimiento en las condiciones del plan.

+!pensarJugada

Recoge en una lista todas las instancias del tablero y lanza el plan !comprobarTablero.

+!comprobarTablero(Lista,N)

Itera recursivamente sobre si mismo para recorrer la lista y ejecutar los planes de !comprobarDireccion y !comprobarPuntos.

+!comprobarDireccion(X,Y)

Comprueba en las coordenadas el resultado de mover la ficha hacia abajo o hacia la derecha con !comprobarDown y !comprobarRight. Y comprueba que es la jugada con más puntos con !comprobarPuntosDown y !comprobarPuntosRight.

+!comprobarDown(X,Y)
+!comprobarRight(X,Y)

Comprueba los puntos que se consiguen si se ejecuta una jugada de la ficha X,Y en dirección abajo o a la derecha.

+!comprobarPuntosDown(X,Y)
+!comprobarPuntosRight(X,Y)

Comprueba que en la comprobación anterior se batido el máximo de puntos. En cuyo caso se guardan los puntos y la dirección en la que se ha movido para conseguirlos.

+!comprobarPuntos(X,Y)

Comprueba que la jugada ha superado el máximo de puntos y permite almacenar en el valor las coordenadas, los puntos máximos, y a dónde se ha dirigido la ficha.

+!comprobarCeroPuntos

Comprueba que no se han podido hacer puntos con ninguna jugada del tablero, y en consecuencia ejecuta una jugada aleatoria.

+!handlePattern(Color,StartsAtX,StartsAtY,Direction,Pattern)

Ejecuta el plan !explosion en función del patrón que se haya formado, para comprobar las consecuencias del patrón en la puntuación.

+!handleT(Color,StartsAtX,StartsAtY,Direction,Pattern)

Maneja las particularidades del patrón en T.

+!explosion(X,Y)

Comprueba que hay una ficha donde se pretende hacer la explosión y llama a !specialExplosion.

+!specialExplosion(X,Y,C,T,D)

Maneja la explosión de las fichas en función del tipo de ficha que se indique en T, y de la dirección desde la que se ha movido, y que se indica en D.

+!coExplosion(X,Y)

Maneja la explosión de de las celdas contiguas a una ficha co.

+deleteTableroBB [source(judge)]

Borra todas las instancias del tablero del jugador.

+tablero(Celda,Ficha)[source(judge)]

Actúa como interfaz, y permite añadir como creencias propias las que envía el juez.

+valido[source(judge)]

Recibe del juez que ha sido un movimiento válido e imprime por pantalla que ha sido así

+tryAgain[source(judge)]

Recibe del juez que la jugada no cumple con la restricción de movimiento del color de las fichas, y llama de nuevo a !realizarJugada.

+invalido(fueraTablero,N)[source(judge)]

Recibe del juez que se ha salido del tablero. Si N es menor o igual que 3 imprime por pantalla que se ha salido del tablero y vuelve a ejecutar !realizarJugada, si no imprime que se ha equivocado demasiadas veces y manda al juez pasoTurno.

+invalido(fueraTurno,N)[source(judge)]

Recibe del juez que ha intentado mover fuera de su turno y lo imprime por pantalla.

+Default[source(A)]

Imprime por pantalla que ha recibido un mensaje de A, pero que no lo reconoce en ningún otro lugar del código.

judge

+!startGame

Inicia la partida, manda a los jugadores la información del tamaño y el contenido del tablero. Da por comenzado el juego e inicia el primer turno.

+mostrarTablero(P)

Recoge en una lista todas las creencias de la forma tablero(celda,ficha) y se las envía a el jugador P.

+!comienzoTurno

Comprueba en primera instancia que se haya alcanzado el final del juego y anuncia los ganadores de ambos niveles y el de la partida al final.

Luego comprueba si ambos jugadores están descalificados. Lo muestra por pantalla y da por finalizada la partida.

En un tercer intento comprueba quién es el jugador al que le toca jugar, las jugadas restantes de la partida, las jugadas que lleva ese jugador, y si ha jugado menos de 50 veces. Le activa el turno para que pueda jugar al jugador que corresponde, y le manda un mensaje de que puede mover.

En un cuarto intento comprobar si las jugadas restantes son 0, y da por finalizada la partida.

Si ninguna de estas ha sido alcanzada, prueba a comprobar que el jugador actual haya llegado a las 50 jugadas, con lo que declara que se ha llegado al máximo por jugador y finaliza la partida.

Por último tiene una entrada por defecto si ninguna otra a unificado.

+!checkFinalWinner(W1,W2)

Comprueba cual es el ganador de la partida y guarda el que corresponda como una creencia. W1 y W2 son los ganadores de cada nivel respectivamente.

+cambioTurno(P)

Coge las jugadasRestantes y las jugadasPlayer y llama a cambioTurno(P,N,J).

+cambioTurno(P,N,J)

Comprueba primero si P es el player1, o si este está descalificado. Pone turnoActual(player2) y resta 1 a las jugadas restantes y le añade una al player1.

Si P es el player2 ejecuta el mismo proceso para el caso contrario.

+cambioTurnoMismoJugador(P)

Coge las jugadasRestantes y las jugadasPlayer y llama a cambioTurnoMismoJugador(P,N,J).

+cambioTurnoMismoJugador(P,N,J)

Ejecuta las mismas acciones que cambioTurno pero para el caso en el que ya sólo queda un jugador jugando.

+moverDesdeEnDireccion(pos(X,Y),Dir)[source(P)]

Si el jugador es el siguiente en jugar, el movimiento es válido y el juez le ha activado el turno, ejecuta la jugada, envía la información de valido al jugador y restablece el turno para continuar la ejecución del juego.

Si el movimiento no es válido envía al jugador la información de que su movimiento no ha sido validado.

Si no tiene el turno activado se imprime por pantalla que el agente debe esperar a que el juez le active el turno.

Si el jugador no tiene el turno actual ni el turno activado se imprime por pantalla que un agente externo intenta realizar una jugada.

Por último recoge una salida por defecto si no ha unificado con ninguno de los casos anteriores.

+turnoTerminado(P)

Si el otro jugador ha sido descalificado llama a cambioTurnoMismoJugador(P), si no a cambioTurno(P).

+intercambiarFichas(X,Y,Dir,P)

Recibe las coordenadas a las que moverse de nextPosition(X,Y,NX,NY,Dir), y el entero que corresponde al dueño de la ficha a mover de plNumb(P,PlNumb). Una vez los tiene actualiza la información que tiene del tablero para la ficha que se mueve y la que se intercambia por ella, y actualiza esa misma información para cada uno de los jugadores.

+fueraTurno(P,N)

Comprueba si un jugador ya ha jugado fuera de su turno 3 veces, y si es así actualiza la información de jugadorDescalificado para que no pueda jugar en lo que queda de partida.

+movimientoInvalido(pos(X,Y),Dir,P)

Comprueba primero si el movimiento ha sido inválido porque se ha intentado jugar fuera del tablero. Si es así le vuelve a dar el turno al jugador e incrementa en 1 el contador de fueraTablero. También envía al jugador el fallo que ha cometido y el número de veces que lo ha cometido en ese turno.

Comprueba si no fue el primer caso que haya sido porque el color de las fichas a intercambiar es el mismo, en cuyo caso simplemente permite al jugador repetir la jugada sin penalización.

Si ninguno de estos dos casos se han producido se imprime que se ha producido un movimiento invalido no controlado.

+pasoTurno[source(P)]

El jugador P pasa turno y se vuelve a actualizar el contenido del contador fueraTablero, y se empieza de nuevo la partida.

+Default[source(A)]

Imprime por pantalla que ha recibido un mensaje de A, pero que no lo reconoce en ningún otro lugar del código.

+size(N).

Plan que recibe el tamaño del entorno. Está únicamente para que no salte la salvaguarda que hemos puesto para un plan indefinido.

+addTablero(Celda,Ficha)[source(percept)]

Este plan es lanzado desde el entorno, se encarga de crear las creencias tablero(Celda,Ficha) para nuestro juez.

+!showPoints

Muestra por pantalla las puntuaciones de ambos jugadores.

+!checkLevelWinner

Comprueba cuál de los dos jugadores tiene más puntos y lo almacena como ganador en su base del conocimiento.

+!generateObstacles

Genera de forma aleatoria entre 0 y 9 obstáculos.

+!generateObstacle(X,Y)

Genera una creencia de tablero de tipo obstáculo y manda al entorno una llamada para que lo represente gráficamente.

+!fullPatternMatch

Comprueba en todo el tablero si se ha producido algún patrón. Por cada posición del tablero llama a !patternMatchPosition(I,J). Una vez comprobados, llama al plan !gravity para la caída de las fichas y al plan !refill para el rellenado del tablero.

+!patternMatchPosition(I,J)

Comprueba para la posición dada si se ha producido algún patrón, si se ha producido gestiona ese patrón, y en caso de ser necesario añadir una ficha especial lo hace.

+!gravity

Controlla la caída de las fichas para todo el tablero.

+!fall(X,Y)

Controla la caída de una ficha en la posición indicada por X,Y. Si está vacía la ficha de abajo, llama a !fallAndRoll(X,Y) .

+!fallAndRoll(X,Y)

Hace caer la ficha, y si se deja hueco la hace rodar después.

+!refill

Gestiona el rellenado del tablero, generando una ficha aleatoria y haciéndola caer, para cada posición en la que haya una ficha vacía.

+!refillSteak(X,Color)

Coloca la nueva instancia de tablero en su base del conocimiento e informa al entorno para que incluya la ficha en el modelo.

+!updatePlayersTableroBB

Manda a los jugadores que borren sus creencias de tablero y a continuación les manda actualizado el nuevo.

+!mostrarTablero(P)

Comunicacion del tablero al jugador indicado.

+!handlePattern(Color,StartsAtX,StartsAtY,Direction,Pattern)

Manda al entorno borrar las fichas que componen el patrón del modelo y ejecuta el plan !explosion en función del patrón que se haya formado, para comprobar las consecuencias en la puntuación.

+!handleT(Color,StartsAtX,StartsAtY,Direction,Pattern)

Maneja las particularidades del patrón en T.

+!setMovedSteak(X,Y,Pattern)

Establecer celda como movida para generar sobre ella la ficha especial cuando la explosión es directamente realizada por el movimiento del jugador.

+!generateSpecialSteak(Pattern,Color)

Generación de la ficha especial acorde al patrón que le corresponde tras una explosión.

+!dualExplosion

Controla que se ejecuten bien dos explosiones contiguas.

+!generateSpecialSteak(Pattern,Color)

Genera una ficha especial en el entorno y en la base del conocimiento.

+!specialStickGenerationPoints(T)

Suma los puntos correspondientes a la generación de una ficha especial.

+!explosion(X,Y)

Comprueba que hay una ficha donde se pretende hacer la explosión y llama a !specialExplosion.

+!specialExplosion(X,Y,C,T,D,A,P,L)

Maneja la explosión de las fichas en función del tipo de ficha que se indique en T, y de la dirección desde la que se ha movido, y que se indica en D.

+!coExplosion(X,Y)

Maneja la explosión de de las celdas contiguas a una ficha co.

ENTORNO

Nuestro entorno consta de tres clases: Tablero, TableroModel y TableroView.

La clase Tablero, que se encarga de instanciar el modelo y la vista del tablero. Contiene además los códigos de colores para cada una de las fichas y el tamaño que va a tener el tablero.

Al iniciarse, le pasa al juez el tamaño del tablero y de instanciar el modelo y la vista. Su función a mayores es la de esperar que el juez le envíe una petición, cuyas respuestas contiene dentro de un switch en executeAction. Puede recibir del juez las peticiones de intercambiar dos fichas del tablero, de borrar una ficha, de añadirla, de poner un obstáculo en el tablero o de resetear el tablero. Todas estas acciones incluyen una llamada a la función correspondiente en el modelo, parseando la entrada que llega desde el juez cuando es necesario.

La clase TableroModel contiene los métodos a los que llama la clase Tablero para realizar operaciones sobre el modelo, y dos métodos auxiliares más para gestionar los códigos de color del Tablero y del juez. En el constructor crea el tablero con el método reset y avisa al juez de que puede iniciar la partida.

La clase TableroView se encarga únicamente de la representación gráfica. Contiene métodos para dibujar fichas sobre el tablero. Y recibe las órdenes de dónde y cuándo alterar la representación gráfica del modelo.

CONCLUSIÓN

En esta segunda entrega hemos aprendido no sólo a mejorar los conocimientos obtenidos en la primera entrega, sino además a gestionar un entorno que da soporte gráfico a la lógica del juego. También se ha notado que la programación lógica se nos empieza a hacer más natural, hasta el punto de tener ya cierta soltura a la hora de transformar en código las características del proyecto, y de interpretar el código que nuestro compañero.

De cara a la tercera entrega esperamos llegar a ganar más experiencia especialmente en lo que se refiere a consistencia del código y a las buenas prácticas dentro del lenguaje lógico, para terminar así de afianzar los conocimientos adquiridos a lo largo del proyecto.

REFERENCES

Bordini, R. H., & Hübner, J. F., & Wooldridge, M. (). (2007). Programming Multi-Agent Systems in AgentSpeak using Jason. Wiley.

Bordini, R. H., & Hübner, J. F. . Jason, a Java-based interpreter for an extended version of AgentSpeak. From http://jason.sourceforge.net