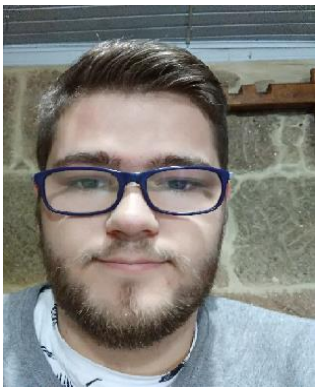
Entrega final

AUTORES

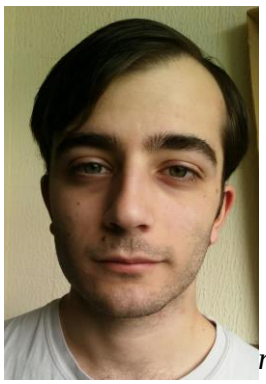


Diego Pérez Domínguez
dpdominguez@esei.uvigo.es
Emilia Pardo Bazán nº9 2ºc Ourense
Estudiante de grado de Ingeniería Informática en 3º curso
Conocimientos:

- *-Programación orientada objetos.*
- *-Java(avanzado),Python(básico),JavaScript(básico),PHP(básico-medio), C(básico). Bases de datos Oracle, MySql y lenguaje Sql y PlSql(básico).*
- *-Ingeniería del Software.*
- *-S.O Linux(medio-avanzado).*

Intereses:

- *-Estoy especialmente interesado en el ámbito de la Inteligencia Artificial, en todas las posibilidades que ofrece y también en como se puede aplicar la informática a la vida cotidiana y a la ciencia en general para mejorar la vida de las personas .*



Diego Pérez Solla
dpsolla@esei.uvigo.es
Emilia Pardo Bazán nº9 2ºc Ourense
Estudiante de grado de Ingeniería Informática en 3º curso
Conocimientos:

- *-Programación de aplicaciones web, APPs (Android e IOS),*
- *-Gestión y automatización de Servidores.*
- *-Lenguajes: Java, JavaScript, Python, Delphi, C/C++.*
- *-Idiomas: Castellano y Gallego como lenguas maternas, nivel medio de ingles.*
- *-Manejo de nivel Administrador tanto en Windows como en Linux.*

Intereses:

- *-Me interesa todo lo que tiene que ver con la Algoritmia y la automatización de tareas. También en como estas dos áreas se pueden entremezclar con la programación concurrente para diseñar Sistemas capaces de aprovecharse de la potencia del hardware que tenemos hoy día.*

David Prieto López
davurin76@gmail.com, rúa rodriguez de la passera nº16 2ºD
Estudiante de Ingeniería Informática en la ESEI (2016 – Act).
Técnico Superior en Administración de Sistemas Informáticos (2014 – 2016).

RESUMEN

En el presente documento se puede ver en síntesis el desarrollo de la Entrega 2 de Sistemas Inteligentes. Se puede ver el proceso de desarrollo de la práctica, así como problemas y dificultades encontradas durante el proceso y qué soluciones se les han dado. También se detallan las estructuras y estrategias seguidas para su realización.

INTRODUCCIÓN

Para esta práctica se ha partido del código votado por todos los alumnos matriculados. El ganador fue el código de Roi Pérez López y Martín Puga Egea. A partir del cual se modificó lo necesario en el juez, para cumplir los requisitos de esta entrega.

DESARROLLO

La explicación del desarrollo se va a dividir en tres partes, la relativa al juez, la relativa a los jugadores y al entorno.

Juez

En cuanto al juez es el encargado de gestionar la partida. Debe asegurarse que los movimientos son válidos y de tener en cuenta los posibles casos de que no lo sean y tomar las medidas necesarias. También se encarga de gestionar los turnos y ordenar el comienzo y fin del juego, así como los cambios de nivel y declarar ganadores.

Para conseguirlo se ha llevado a cabo la siguiente programación.

Base de conocimiento:

contadores: se utilizan para retener toda la información que necesita el juez para gestionar la partida, tamaño máximo del tablero, número de puntos de cada jugador, puntos máximos de la partida, jugadores descalificados, jugador actual,….
.

findeJuego: es un flag que indica que el juego ha finalizado.

movimientoValido: comprobación de todos los parámetros que indican que un movimiento es válido, las celdas origen y destino no son las mismas, el color es distinto, la celda existe,..

direccionCorrecta: comprobación celda destino existente según la dirección, es decir que el movimiento no se sale del tablero.

plNumb: comprueba que le jugador actual es al que le corrsponde el turno.

nextPosition: determina las coordenadas de la ficha destino para un movimiento.

comprueba" ": comprueba, mediante llamadas a los plotios correspondientes a cada patrón que para el movimiento actual se genera un patrón.

vacioFila: comprueba si hay celdas vacías en la primera fila, para la posterior generación de fichas nuevas.

vacioDebajo: verifica que existe una ficha que tiene un vacío debajo.

vacioLeft: verifica que existe una ficha con un vacío a la izquierda.

ganador: guarda en la mente del juez el ganador de cada nivel.

"flags": diversas marcas usadas para controlar la ejecución de algunos planes.

Objetivos:

cominezoTurno: se encarga de dar comienzo a cada turno enviando al jugador correspondiente el mensaje, puedesMover. Incluye condiciones que detectan el final de los niveles y del juego para que no se produzcan turnos demás.

declararGanadorPuntos: muestra quien ha sido el ganador del nivel X y cuantos puntos tiene, en caso de empate indica que no hay un ganador claro.

declararGanadorPartida: muestra que jugador ha ganado la partida en base al número de niveles ganados, en caso de empate lo indica.

actuTableroPlayers: actualiza el tablero que se encuentra en las mentes de los jugadores.

compruebaAgru: si existe alguna agrupación para las coordenadas dadas explota todas sus fichas y genera la ficha especial correspondiente al patrón encontrado.

explotaFicha" ": se encarga de explotar la ficha o fichas (dependiendo del tipo) correspondientes y suma al jugador actual los puntos correspondientes por cada ficha.

fichaSpecial: genera las fichas especiales que resultan de la explosión de los patrones más complejos además le suma al jugador actual los puntos correspondientes a la generación de la ficha.

cambioTurno: se encarga de ir cambiando los turnos de un jugador a otro restando el número de jugadas restantes y aumentando los contadores de jugadas realizadas. Con condiciones de entrada para cada jugador y uno para cuando algún jugador ha sido descalificado y se cambia el turno consigo mismo.

Caída: comprueba si hay alguna ficha que pueda caer en el tablero y de ser así las hace caer una a una hasta su posición final llamando a caete.

Caete: si la ficha tiene un hueco debajo la hace caer y activa el flag que indica que ha caido, si no es así llama a rodar.

rueda: se encarga de hacer rodar a hacia la izquierda las fichas que ya han caído al menos una vez.

rellenarFichas: introduce fichas en los huecos de la parte superior del tablero.

revisarPatrones: después de que las fichas caigan tras la explosión de fichas generadas por un movimiento de un jugador, comprueba si las fichas que han caído han generado algún patrón mediante llamadas a compruebaAgru y de ser así vuelve a llamar a caída para rellenar el tablero otra vez.

finDeNivel: comprueba si ha terminado el nivel, bien sea por el límite de puntos o por el de jugadas, de ser así declara el ganador y si es el último nivel termina el juego.

Planes**:**

movimientoInvalido: el movimiento recibido no es válido, puede ser porque o ha salido fuera del tablero, se le notifica al jugador con un mensaje inválido y el motivo y vuelve a mover, si lo repite más de cuatro veces pierde el turno. El otro inválido posible es que la ficha seleccionada se mueva a otra del mismo color o que sean obstáculos o celdas vacías. Entonces se notifica al jugador con un mensaje tryAgain, no tiene límite de intentos.

turnoTerminado: se llama al finalizar cada turno, incluye condiciones para detectar el final de los niveles, hay uno por cada nivel. Al finalizar el nivel 1 se resetea el tablero, se generan los obstáculos mediante la llamada a nivel en el entorno y se les muestra el tablero a los jugadores. Si no se ha detectado el final del nivel, se cambia de turno normalmente.

fueraTurno: se encarga de descalificar a un jugador cuando este intenta mover fuera de su turno más de 3 veces.

intercambiarFichas: dada una celda y una dirección calcula la celda destino , intercambia la información de ambas celdas, comprueba si se produce una agrupación y de ser así ejecuta su explosión . Después actualiza los tableros de ambos jugadores y comprueba su territorio.

mostrarTablero: se encarga de comunicar a los jugadores el tablero, para que estos lo tengan también en sus mentes.

cambioTurno: se encarga de cambiar el turno entre los distintos jugadores, bien sea porque han hecho un movimiento valido o uno invalido que no resulte en un tryAgain.

cambioTurnoMismoJugador: si alguno de los jugadores esta descalificado se encarga de disminuir el número de jugadas restantes y aumentar el de realizadas del jugador que sigue jugando.

default: se encarga de dar salida a todo aquello que no esté contemplado en el resto del código.

Trigger:

moverDesdeEnDireccion: mensaje que recibe de los jugadores con el movimiento que desean realizar. Con la llegada de este mensaje pueden suceder varias cosas, el movimiento es válido, entonces se informa al jugador de que ha sido válido y se llama al plan itercambiarFichas, al plan turno terminado y de nuevo a comienzoTurno. Si el movimiento ha sido inválido, se descarta y se llama al plan movimiento inválido. También se puede dar que el movimiento se envie fuera de turno, si es así, se avisa al jugador y vuelve a mover, si repite esto más de tres veces es descalificado. Se añaden posibilidades de que se reciban mensajes de jugadores ajenos o inesperados, se tratan para evitar el paro del programa, solo se deshechan.

pasoTurno: mensaje que envía el jugador cuando ha sido informado de que ha movido fuera del tablero más de cuatro veces y pierde el turno. Dispara el cambio de turno y llama a comienzoTurno.

startGame: este lo despierta el entorno después de generar el tablero,muestra el tablero a los jugadores e inicia el juego.

Jugador

En cuanto al jugador, tenemos la siguiente programación:

Base de conocimiento:

Conocimiento recibido del juez: este jugador necesita recibir del juez en qué nivel se está jugando y el tamaño del tablero.

Reglas de mejorMov: el jugador rastrea todo el tablero en busca del mejor movimiento, estas reglas almacenan el mejor encontrado por cada turno.

movimientoValido: regla interfaz de llamada a las comprobaciones de color (regla compColor) y tipo, el jugador solo debe elegir movimientos de fichas de distinto color y que no sean huecos vacíos (regla compVacio) ni obstáculos (regla compObs).

validoMov: es una regla que comprueba que la celda de destino no está fuera del tablero.

plotioX: son las reglas que comprueban si hay alguna agrupación. Se llaman plotio seguido del nombre de la agrupación. Se hacen verdaderas si a partir de la ficha situada en las coordenadas que se le indican hay una figura en T, en Cubo, Cinco en fila, Cuatro en fila, Tres en fila.

obtenerPos y creaDir: son dos reglas para generar un movimiento aleatorio de ser necesario.

nextPosition: obtiene las coordenadas destino según la dirección.

Objetivos iniciales: no dispone de ellos. Está a la espera de un mensaje del juez.

Objetivos:

actuVirtual: para no modificar el tablero enviado por el juez y así tener siempre un tablero fiable y real, el jugador hace sus comprobaciones y búsquedas sobre tableroVirtual, es una copia del tablero recibido del juez. Este objetivo se encarga de actualizarlo.

eligeMov: objetivo encargado de la simulación de movimientos. Este jugador para la búsqueda del mejor movimiento, simula movimientos celda a celda y comprueba que agrupaciones se han generado, siempre en su tableroVirtual.

agrupación: objetivo encargado de comprobar si hay agrupación o no a partir del movimiento proporcionado. Llama a los planes calculaPuntos y calculaTerritorio.

agrupacionRandom: objetivo que genera un movimiento aleatorio.

seleccionEstrategia: este objetivo elige los movimientos a enviar según la estrategia correspondiente en cada nivel. Como se explicará más adelante, el plan

guardadoMovimiento se encargará de guardar distintos movimientos como mejores, siguiendo distintos criterios. seleccionEstrategia se encarga de recuperar en el nivel 1 y 2 el movimiento que da más puntos, guardado como mejorMovPuntos, en caso de no existir elegiría el envío de uno aleatorio. Si no fuera posible envía un movimiento aleatorio..

Trigger:

puedesMover: trigger que le indica al jugador que puede mover, arranca todo el proceso del jugador de elegir y enviar el movimiento.

valido: mensaje que le indica al jugador que su movimiento fue válido.

invalido: mensaje enviado por el juez a los jugadores por dos motivos. Han movido fuera del tablero, si superan el límite de tres inválidos por este motivo, el jugador pasa turno. El segundo motivo es que envíe fuera de turno, si supera el límite de 3 veces, el juez lo descalifica.

tryAgain: mensaje recibido cuando se ha intentado mover fichas del mismo color o el envío de un movimiento invalido por algún motivo que no encaja en los dos anteriores.

Planes:

calculaPuntos: recibe las coordenadas de una celda y calcula los puntos que daría explotar esa ficha, actualizando el contador puntos.

guardadoMovimiento: una de las partes más importantes del jugador, se encarga de actualizar el movimiento guardado como (mejor mejorMovPuntos). Si el movimiento actual da más puntos que el guardado, lo actualiza(mejorMovPuntos). Por otro lado, si el número de puntos es el mismo, prioriza el movimiento de fichas situadas más abajo en el tablero, y así producir un mayor movimiento de las fichas y posibilitar más agrupaciones. Para ello dispone de un contador, puntos , se pone a cero con cada agrupación y calculaPuntos.

Entorno

En cuanto al entorno, tenemos la siguiente programación, la dividimos en tres apartados:

Modelo:

Se encarga de crear el tablero y de actualizar la información referente a las posiciones del grid que conforma el mismo:

- **Constructor:** crea un nuevo tablero y se lo comunica al juez.
- **ColocarFicha:** intercambia el contenido de 2 celdas.
- **Delete:** elimina el contenido de una celda.
- **Nivel:** genera un nuevo tablero con obstáculo para el nivel 2.
- **setDuenho:** modifica el dueño de una celda.
- **GetFicha:** devuelve la ficha de una posición del tablero

Vista:

Se encarga de representar la información recogida en el modelo de manera gráfica. Para ello dispone de los siguientes métodos:

- **Draw:** método por defecto al que se llama para dibujar el grid, se encarga de llamar a los correspondientes draw de las fichas.
- **DrawSteak:** se encarga de representar las fichas, los tipos se indican mediante un label.
- **DrawObstaculo:** representa los obstáculos en el tablero, mediante cuadrados negros.

Execute_Action: _____

Esta es la interfaz que media entre las llamadas del juez al entorno y los métodos Java implementados en el mismo. Así como el parseo de los parámetros necesarios para esos métodos.

CONCLUSIÓN

Como conclusión destacar la dificultad de lidiar en ciertas ocasiones con la sintaxis del lenguaje, así como paradas inesperadas sin error aparente.
El hecho de tener que adaptar código ajeno también es complicado en muchas ocasiones, ya sea por la difícil comprensión de un código ajeno o el no conocer las intenciones de hacer una cosa de una forma concreta.
Uno de los mayores problemas encontrados en esta entrega ha sido la correcta elaboración de la caída de las fichas y las revisiones de patrones que conlleva.
A medida que se profundiza en el funcionamiento del lenguaje y de este tipo de programación hace que sea más fácil, en contrapartida a los problemas encontrados.

REFERENCIAS

Web site:

API de JASON.

http://jason.sourceforge.net