Entrega final

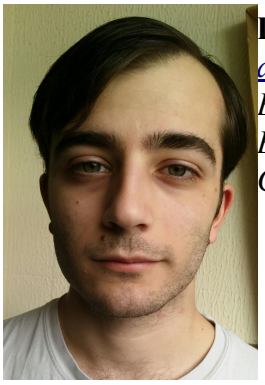
AUTORES

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 3 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 y tablero, para cumplir los requisitos de esta entrega final.

DESARROLLO

La explicación del desarrollo se va a dividir en tres partes, la relativa al juez y 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 niveles 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, numero de celdas conquistadas,puntos máximos de la partida,....

levelWinner: ganador de cada nivel.

final Winner: ganador de todo el juego.

specialSteak: correspondencia entre los patrones de explosión y sus fichas especiales asociadas.

generationPoints: puntos correspondientes a la generación de cada una de las fichas especiales.

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: jugador del turno actual.

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

comprobarPatrones: comprueba, mediante llamadas a pattern…. , para un movimiento dado si se genera alguna agrupación.

pattern…..: se corresponden con las distintas agrupaciones posibles que se pueden generar a partir de un movimiento.

emptyUnder: verifica que no existe una ficha por debajo de la posición.

emptyLeft: verifica que no existe una ficha por la izquierda de la posición.

"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. Ha sido modificado para añadir el nivel tres y que ahora el número de jugadas máximas varía con el nivel, antes eran siempre 100.

checkFinalWinner: declara el jugador ganador del juego según quién ganó los niveles. Gana quien haya ganado los tres niveles, o quien haya ganado dos niveles a uno del contrario, si no gana el que más puntos obtenga en total en los tres niveles.

showPoints: consulta los puntos según el nivel y el jugador y los muestra.

checkLevelWinner: compara las puntuaciones de ambos jugadores y almacena el ganador del nivel.

checkLevel3Winner: específico para el nivel 3 dado que usa las celdas conquistadas.

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.

generateObstacles: crea el número de obstáculos fijados de forma aleatoria.

generateDuenhoInicios: genera el número prefijado de celdas propiedad de cada jugador, se asignan de forma aleatoria, sin afectar a los obstáculos que no pertenecen a nadie.

exchangePatterMatch: llama a patternMatchExchangedPosition con la celda origen y destino.

patternMatchExchangedPosition: comprueba si existen agrupación desde la celda llamando al plan comprobarPatrones, genera las fichas especiales de ser necesario y rellena el tablero mediante gravity y refill

fullPatternMatch: recorre todo el tablero buscando agrupaciones después de la caída de las fichas, que vuelve a ser llamada si se detecta algún patrón y es explotado.

patternMatchPosition: dada una posición comprueba si hay patrones a partir de lla y llama a handlePattern que lo explotará, a setMovedSteak que marca la celda origen y destino como movidas y a generateSpecialSteak que generará una ficha especial si el patrón explotado así lo requiere.

gravity: hace caer las fichas mientras alguna ficha en el tablero tenga un hueco debajo mediante llamadas a fall por cada una de las fichas del tablero.

fall: de tener un hueco por debajo de la celda, hace caer la ficha mediante llamadas a fallAndRoll

fallAndRoll: hace caer o rodar la ficha dependiendo de si la ficha encuentra un hueco debajo o a la izquierda.

refill: crea fichas el las celdas que estén vacías de la primera fila, llama a refillSteak que borra la celda vacía y la rellena. Al acabar refill, se llama a gravity que la shace caer y de nuevo a refill para rellenar todos los huecos.

updatePlayersTableroBB: envía el mensaje de borrado del tablero a los jugadores y les muestra el nuevo llamando a mostrarTablero.

mostrarTablero: envía todas las creencias que concuerden con tablero a ambos jugadores.

handlePattern: hay uno por cada patrón y llama a la explosión ficha a ficha perteneciente a la agrupación. También las borra del modelo de datos.

setMovedSteak: marca la celda como movida para, de ser necesario, generar a posterior sobre llea la ficha especial resultante de la explosión.

dualExplosion: gestiona la explosión de dos patrones contiguos, para la correcta generación de fichas especiales.

generateSpecialSteak: coloca una ficha especial en la celda marcada como movida cuando el patrón explotado así lo requiere.

specialStickGenerationPoints: suma los puntos correspondientes a la generación de una ficha especial.

explosion: elimina la ficha cuyas coordenadas se le pasan, actualizando los puntos y actualizando el territorio. Esta gestión se realiza en los tres niveles aunque en el 1 y 2 es siempre neutro. El funcionamiento es el siguiente, si la celda donde se originó el movimiento es propiedad neutra, el territorio no varía, si no es neutro, las celdas que exploten pasan a ser propiedad del jugador dueño

de la ficha origen, esto puede redundar en una conquista del territorio del jugador que mueve o en beneficio para el enemigo.

specialExplosion: gestiona los distintos tipos de explosiones asociados a cada tipo de ficha. Según el tipo de ficha llama a explosión con las fichas extra que explotan, esto es, una columna entera, una fila entera, etc… En estos objetivos ha sido necesario dejar comentadas ciertos borrados redundantes que realizaban en el código base, se intentó adaptar y corregir pero al final la única solución para evitar fallos fue comentarlos.

coExplosion: se encarga de la explosión del tipo co en específico.

compruebaTerritorio: recorre todo el tablero contando cuantas celdas posee cada jugador. Actualiza los contadores de territorio y es llamado al final de cada turno.

Planes**:**

movimientoInvalido: el movimiento recibido no es válido, puede ser por que 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. El nivel 1 y 2 finaliza cuando un jugador alcanza el límite de puntos establecido, o se agotan el número de jugadas permitidas. El nivel 3 presenta condiciones distintas, este finaliza si algún jugador conquista todo el territorio y alcanza el número de puntos marcado o por las condiciones anteriores, el límite de jugadas posibles. Al finalizar el nivel 1 se resetea el tablero, se generan los obstáculos con el objetivo generateObstacles y se les muestra el tablero a los jugadores. Al finalizar el nivel dos se hace lo mismo y además se generan un número de celdas propiedad de cada jugador, con el objetivo generateDuenhoInicios. En ambos niveles es necesario informar a cada jujgador de que nuevo nivel está empezando. Al finalizar el nivel 3, se consulta quien ha ganado y se declara el fin del juego con el flag endGame. Si no se ha detectado el final del nivel, se cambia de turno normalmente.

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.

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.

addTablero: recibe la celda del entorno y la transforma al formato del juez.

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.

yoSoy: reglas para que jugador sepa quién es y así poder tener en cuenta la posesión de territotio.

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. En el nivel 3, esta no es la estrategia, de existir, elegiría el movimiento guardado como mejorMovTerritorio, porque conquista nuevo territorio. Si no es posible, elige el envío de mejorMovTerritorioNeutro, este da la mayor cantidad de puntos desde una celda que no es de nadie y así no beneficiar al enemigo. Si no fuera posible ninguna de estas dos opciones, envía un movimiento aleatorio, dado que hay menos posibilidades de que se forme agrupación y que sea negativa para el propio jugador.

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.

deleteTableroBB: trigger que dispara el borrado del tablero y la actualización del tablero virtual

Planes:

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

calculaTerritorio: recibe las coordenadas de una celda y comprueba si no es del propio jugador, si esto es así aumenta un contador territorio en 1 si ya es suya, no hace nada. Para esto el jugador debe saber quién es, usando la acción interna my_name.

guardadoMovimiento: una de las partes más importantes del jugador, se encarga de actualizar los movimientos guardados como mejores. Dispone de tres movimientos mejores, mejorMovPuntos, mejorMovTerritorio, mejorMovTerritorioNeutro. Si el movimiento actual da más puntos que el guardado, lo actualiza(mejorMovPuntos), a su vez si este movimiento parte de una celda neutra actualiza mejorMovTerritorioNeutro. 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. Por último, si este movimiento actual, proporciona más celdas conquistadas que el almacenado, se actualiza mejorMovTerritorio. Para ello dispone de dos contadores, puntos y territorio, se ponen a cero con cada agrupación y calculaPuntos y calculaTerritorio los actualizan.

Entorno

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

Modelo:

Se encarga de recoger toda la información referente al estado actual del tablero, fichas dentro las celdas, dueño de cada una de las celdas y obstáculos. Para ello dispone de un conjunto de metodos:

- **Reset:** elimina el tablero de la mente del juez, para posteriormente sobrescribir el que está guardado en el modelo y volver a pasarlo al juez.
- **Exchange:** intercambia el contenido de 2 celdas.
- **Delete:** elimina el contenido de una celda.
- **Put:** añade una ficha a una celda.
- **PutObstacle:** añade un obstáculo a una celda.
- **PutDuenho:** modifica el dueño de una celda.
- **GetJudgeColor:** obtiene la correspondencia entre el código de color del entorno y el juez.
- **GetEnviromentColor:** obtiene la correspondencia entre el código de color del juez y el entorno.

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 representar el territorio de cada jugador en el tablero y de llamar a los correspondientes draw de las fichas.
- **DrawSteak:** se encarga de representar las fichas, para cada color utiliza una forma distinta.
- **DrawSpecialSteak:** representa las fichas especiales de misma forma que **DrawSteak** solo que a mayores indica el tipo de ficha con un tag.
- **DrawObstacle:** 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. También surgieron problemas en la gestión de la propiedad de las celdas, según los niveles. Se optó por gestionarla en los tres, con la diferencia que en el uno y dos siempre es neutral.

La máxima complejidad ha sido hacer lo más inteligente posible al jugador. Se añaden nuevas estrategias según el nivel, por lo que debe intercambiarlas según en el que esté en cada momento. Se optó por que guardase los tres mejores movimientos, uno según sus puntos, otro según el territorio y otro según el territorio neutro. Y así implementar la selección de estrategia en el envío del movimiento, según el nivel en el que se encuentra envía uno u otro. Esto supuso un problema a mayores, el número de comprobaciones que debía realizar lo que ralentizaba el desarrollo del juego.

Para la realización de estas comprobaciones y todos los cálculos se optó por la utilización de un tablero "virtual" debido a la aparición de problemas de duplicidades de las creencias de tablero enviada por el juez y las almacenadas por el propia jugador.

REFERENCIAS

Web site:

API de JASON.

http://jason.sourceforge.net