**FIUBA - 75.07**

**Algoritmos y programación III**

*Trabajo práctico 2: AlgoFormers*

1er cuatrimestre, 2016

(trabajo grupal de 4 integrantes)

Alumnos:

|  |  |  |
| --- | --- | --- |
| **Nombre** | **Padrón** | **Mail** |
| Franco Bragantini | 97190 | franco-xk94@hotmail.com |
| Federico Rossini | 97161 | federicorossini09@gmail.com |
| Mariano Baffetti | 93791 | marianobaffetti@hotmail.com |
| Pedro Martínez Peck | 97123 | Pedro\_peck\_8@hotmail.com |

**Supuestos**

Se han tenido en cuenta los siguientes supuestos para complementar los puntos que en el enunciado del trabajo hayan tenido, desde nuestro punto de vista, una interpretación abierta:

* Todos los Algoformers comienzan el juego en su estado Alterno.
* De ser posible, el movimiento de un Algoformer desde un casillero origen hacia un casillero destino será siempre en “diagonal”. Es decir, si el Algoformer se encuentra en el casillero [1,1] (fila 1, columna 1), y quiere ir al casillero [3,3] lo hará pasando por el casillero [2,2]. En caso de que el movimiento en diagonal no sea posible en su totalidad se comenzará, de todas formas, con un movimiento diagonal hasta coincidir en fila o en columna con el casillero destino. Esto es, si el movimiento es desde el casillero [1,1] al casillero [3,4], la secuencia de movimientos será: [2,2] 🡪 [3,3] 🡪 [3,4].
* El Algoformer podrá moverse desde un casillero origen hacia un casillero destino solamente si los casilleros por los que tiene que pasar, incluyendo el casillero destino, no están ocupados por otro Algoformer.
* Para capturar la chispa, el Algoformer solamente debe pasar por el casillero que la contiene.

**Modelo de Dominio**

En el diseño del programa, tiene una importancia particular la clase abstracta Algoformer. En ésta clase se agrupa el comportamiento común que poseen los seis Algoformers del juego. Parte del comportamiento de un Algoformer concreto depende del estado en el que esté (alterno o humanoide), y a su vez cada estado tiene asociada una estrategia de movimientos. Tanto las clases que representan los estados, como las que representan las estrategias de movimientos fueron diseñadas teniendo en cuenta el patrón strategy. El estado es responsable de contener atributos como los puntos de ataque, distancia de ataque, velocidad y estrategia de movimiento. Además en él de delega parte del comportamiento relacionado con la interacción del Algoformer con los casilleros.

La estrategia de movimiento asociada al estado del Algoformer es responsable principalmente de resolver cuál es el siguiente casillero en una secuencia de movimiento y de guardar los distintos “subestados” que se deriven de la interacción con los casilleros. Por ejemplo, si el Algoformer ocupa un casillero que posee un pantano, la estrategia de movimiento “recordará” que el Algoformer se encuentra “empantanado”.

Como se puede suponer de lo dicho anteriormente, una de las clases centrales ligadas al movimiento es la clase Casillero. Básicamente la secuencia de movimiento de un algoformer podría resumirse como:

1. Obtener siguiente casillero.
2. Desocupar casillero actual.
3. Ocupar siguiente casillero.
4. Interactuar con casillero.

…

En la interacción con el casillero, es el casillero quien se encarga de que los bonus y zonas interactúen con el Algoformer, aplicando daños, cambios de estado o bonus.

Con respecto al ataque de los Algoformers, la clase Arma es quien se encarga de verificar, en primera instancia, si es posible atacar. Esto es, que el Algoformer objetivo esté en rango. Además para evitar que las unidades del mismo equipo puedan atacarse entre sí, se crearon dos clases de Arma, ArmaDecepticon y ArmaAutobot. De ésta forma se resuleve a través del patrón doublé dispatch que sea posible o no atacar a un Algoformer concreto. Éstas clases también son responsables de contener información relacionada con el bonus de ataque. De forma análoga, la clase armadura contiene información relacionada con el bonus de defensa y la clase Piernas, información relacionada con el bonus de velocidad.

Con respecto a la relación que guardan las clases mencionadas entre sí, cada Algoformer posee una referencia a instancias de:

Arma

Armadura

EstadoAlgoformer

EstrategiaDeMovimiento

Piernas

Casillero

A su vez, cada Casillero posee una referencia a instancias de:

Algoformer

Bonus

Y dos instancias la clase Zona.

Finalmente, como se mencionó anteriormente, la clase Estado posee una referencia a una instancia de estrategia de movimiento.

**Detalles de Implementación**

Método atacarA de la clase Algoformer.

Algoformer delega el comportamiento en el método atacar de la clase Arma. El Arma del Algoformer puede ser de la clase concreta ArmaDecepticon o ArmaAutobot. Dependiendo de la clase concreta, el método atacar de la clase arma puede tener dos implementaciones distintas:

Clase ArmaDecepticon

@Override  
public void atacar(AlgoFormer algoformer, int ataque, int distanciaDeAtaque, Casillero casilleroOrigen) {  
  
 if (this.estaEnRango(casilleroOrigen,algoformer.obtenerCasillero(),distanciaDeAtque)){  
  
 int ataqueFinal = ataque;  
  
  
 if (this.turnosRestantesDobleCanion > 0) {  
  
 ataqueFinal = ataqueFinal \* 2;  
  
 }  
  
 algoformer.recibirAtaqueDeDecepticon(ataqueFinal);  
  
 } else {throw new AlgoFormerFueraDeAlcanceException();}  
}

Clase ArmaAutobot

@Override  
public void atacar(AlgoFormer algoformer, int ataque, int distanciaDeAtque, Casillero casilleroOrigen) {  
  
 if (this.estaEnRango(casilleroOrigen,algoformer.obtenerCasillero(),distanciaDeAtque)){  
  
 int ataqueFinal = ataque;  
  
  
 if (this.turnosRestantesDobleCanion > 0) {  
  
 ataqueFinal = ataqueFinal \* 2;  
  
 }  
  
 algoformer.recibirAtaqueDeAutobot(ataqueFinal);  
  
 } else {throw new AlgoFormerFueraDeAlcanceException();}  
}

Como se aprecia en la firma de los dos métodos, ambos reciben entre sus parámetros a un objeto de la clase Algoformer. No importa si el algoformer es de la clase Decepticon o de la clase Autobot. Éste “inconveniente” de no saber la clase concreta del objeto que es objetivo del ataque, se resuleve gracias a que en cada método se le envía un mensaje distinto al Algoformer que se pasa como parámetro.

En el caso de la clase ArmaDecepticon, el algoformer recibe el mensaje recibirAtaqueDeDecepticon y el en caso de la clase ArmaAutobot, el mensaje enviado es recibirAtaqueDeAutobot. El algoformer, dependiendo si es Decepticon o Autobot, responderá éstos mensajes de forma distinta. Si le “corresponde” recibir el ataque, disminuirá sus puntos de vida. Si por el contrario, quien lo ataque es de su mismo equipo lanzará una excepción.

Método moverA de la clase Algoformer

El método moverA de la clase Algoformer, se delega en el método moverA del estado en el que se encuentre. A su vez, el método moverA del estado, se delega en el método mover de la clase EstrategiaDeMovimiento. En éste método se agrupa todo el comportamiento común que poseen cada una de las estrategias de movimiento que pueda adoptar el estado.

public void mover(Casillero destino, AlgoFormer algoFormer) {  
  
 Casillero casilleroActual = algoFormer.obtenerCasillero();  
  
 int puntosDeVelocidad = algoFormer.piernas.modificarVelocidad(algoFormer.obtenerVelocidad());  
  
 if (this.movimientoValido(casilleroActual, destino, puntosDeVelocidad)){  
  
 int distancia = this.obtenerDistancia(casilleroActual, destino);  
 int movimientosDisponibles = puntosDeVelocidad;  
  
 while (distancia > 0 && movimientosDisponibles > 0 && algoFormer.estaVivo() ){  
  
 Casillero casilleroAnterior = algoFormer.obtenerCasillero();  
 algoFormer.desocuparCasillero();  
 casilleroActual = algoFormer.estado.obtenerSiguienteCasillero(casilleroActual, destino);  
 algoFormer.estado.ocuparCasillero(algoFormer, casilleroActual);  
 movimientosDisponibles--;  
  
 if (!casilleroAnterior.equals(algoFormer.obtenerCasillero())){  
  
 distancia--;  
 }  
 }  
  
 } else { throw new MovimientoInvalidoException(); }  
 }

El comportamiento común en el movimiento puede resumirse como:

1. Verificar si es válido el movimiento que se propone. (Es decir, si está en rango y si no hay ningún algoformer en el camino)
2. Mientras queden puntos de velocidad y no se haya llegado a destino repetir la secuencia:

* Obtener siguiente casillero
* Ocupar casillero

La delegación a la clase hija concreta ocurre en éstos dos últimos pasos. Es la estrategia de movimiento concreta, la que establece cuál será el próximo casillero (puede que el algoformer se encuentre empantanado y el próximo casillero resulte el mismo en el que se encuentra) y cómo se ocupa el siguiente casillero (si el algoformer es una unidad aérea lo ocupará de forma distinta a las unidades terrestres).

Para resolver el método obtenerSiguienteCasillero, la estrategia de movimiento concreta utiliza el método calcularSiguienteCasillero de la clase madre. Éste método no sólo contiene el algoritmo que resuelve las posiciones X e Y del siguiente casillero, sino que obtiene una referencia al casillero con esas coordenadas con en el contexto del juego.

**Excepciones**

## **AlgoFormerAtrapadoEnNebulosaNoSePuedeMoverException**: se creó con el fin de avisar al usuario que quiera mover a un algoformer en estado alterno aéreo que se encuentra atrapado en una nebulosa. Esta excepción se lanza antes de realizar la mecánica del movimiento del algoformer (clase Jugador, línea 131) y se atrapa en el evento que llama al método que hace mover al algoformer (clase BotonMoverHandler, línea 51). Una vez capturada la excepción se muestra un mensaje al usuario a través de la vista.

* **AlgoFormerFueraDeAlcanceException**: fue creada para notificar al usuario que el algoformer al que quiere atacar se encuentra fuera del alcance del algoformer que seleccionó como atacante. Se lanza luego de verificar si el algoformer enemigo se encuentra en rango (clases ArmaAutobot y ArmaDecepticon, línea 26 en ambas) y se atrapa en el evento que llama al método que efectúa al ataque (BotónAtacarHandler, línea 46). Una vez capturada se lanza un mensaje al usuario a través de la vista.
* **MovimientoInvalidoException**: se creó con el fin de notificar al usuario que quiera mover a un algoformer a un casillero que no se encuentra a su alcance. Se lanza luego de verificar que el movimiento es válido (clase EstrategiaDeMovimiento, línea 55) y se captura en el evento de la vista que llama al método para mover al algoformer (BotonMoverHandler, línea 44). Al capturarla se muestra un mensaje al usuario.
* **HumanoideNoSePuedeMoverSiEstaEnPantanoException**: fue creada para alertar al usuario que el algoformer en estado humanoide que seleccionó se encuentra atrapado en el pantano y por lo tanto no lo puede mover. Se lanza en la verificación anterior al llamado al método del movimiento (Clase Jugador, línea 133) y se captura en el evento que dispara el movimiento de un algoformer (Clase BotonMoverHandler, línea 59). También se muestra un mensaje al usuario.

Las excepciones anteriores son todas las que capturamos en nuestra implementación. Las excepciones creadas restantes no son capturadas o lanzadas ya que nuestra interfaz gráfica contempla los casos excepcionales, por ejemplo, deshabilitando botones de acciones que el usuario podría realizar. Fueron creadas con el objetivo de dejar constatado lo que no debería ocurrir durante la ejecución del programa.