Facultad
de
Ciencias

FANTASMAS INTELIGENTES PARA EL PAC-MAN BASADOS EN SISTEMAS MULTIAGENTE

(Intelligent Multiagent-Based Ghosts for Pac-Man)

Trabajo de Fin de Grado
para acceder al

GRADO EN INGENIERÍA INFORMÁTICA

Autor: Sergio Salomón García

Directora: Inés González Rodríguez

Septiembre - 2015

Agradecimientos

A mi directora de TFG, por haberme dado siempre ayuda y consejo a lo largo de este proyecto y de toda la carrera. Por su visión de las cosas, su forma de enseñar y su incansable esfuerzo.

A nuestros profesores, por el trabajo que hacen y todo lo que he aprendido con ellos.

A mis amigos, por haber sido respiro, compañía y, sobre todo, apoyo. Por estar a mi lado y recordarme la importancia de las cosas. En especial, a Silvia, Alejandro, Rosalía y Fran.

Por último, a mi familia, por poner el camino que me ha permitido llegar hasta aquí.

Gracias.

Resumen

Este proyecto parte de los objetivos de estudiar y aplicar técnicas de sistemas multiagente al popular videojuego de Pac-Man. En concreto, se busca diseñar un equipo de fantasmas inteligentes y cooperativos, esperando que ofrezca un mayor éxito en la caza del Pacman que un equipo de fantasmas inteligentes no coordinados.

Por esto, se estudian los conceptos clave y las técnicas más importantes de sistemas multiagente, se realiza una implementación del juego de Pac-Man preparada para el acoplamiento de agentes inteligentes como controladores de sus personajes, y se diseña un equipo de fantasmas multiagente.

En el diseño de los agentes cooperativos se emplearán convenciones sociales, reglas orientadas a obtener coordinación entre agentes, y la asignación de roles o clases con las que separar distintos comportamientos inteligentes.

Palabras clave: agentes inteligentes, Pac-Man, sistemas multiagente

Abstract

This project starts with the goals of studying and applying multiagent system techniques to the popular Pac-Man videogame. In particular, our aim is to design an intelligent and cooperative ghost team, in the hope that it is more successful chasing the Pacman than an intelligent but uncoordinated team.

To this end, we study the key concepts and the most relevant methods of multiagent systems, build an implementation of the Pac-Man game adequately adjusted in order to embed intelligent agents as controllers for the game characters, and design a multiagent ghost team.

In the design of cooperative agents, we utilize social conventions, rules oriented to accomplish coordination between agents, and role assignment, or the assignment of classes that separate different intelligent behaviours.

Keywords: intelligent agents, multiagent systems, Pac-Man

Índice general

Índice de tablas

Índice de figuras

Capítulo 1

Introducción

En este capítulo se expone la motivación detrás de este proyecto y los conceptos principales, se indican los objetivos a seguir marcados, y se detallan los distintos pasos llevados a cabo y el modo de trabajo.

1.1. Motivación

Dentro del área de inteligencia artificial, uno de los campos que ha empezado a ganar especial atención en los últimos años es el de los sistemas multiagente y la inteligencia artificial distribuida. A pesar de la vasta cantidad de técnicas vigentes que ignoran la presencia de otros agentes del entorno para la resolución de problemas, en el mundo real abundan los ejemplos donde múltiples entidades autónomas cohabitan. Es decir, una gran cantidad de los problemas en la actualidad son de tipo multiagente. Se considera **sistema multiagente** a aquel formado por un grupo de agentes que pueden interactuar entre sí y poseen distinta información, distintos intereses, o ambos casos. Uno de los ámbitos más claros de sistema multiagente es el de internet, donde la presencia de agentes software que se relacionan es cada vez más frecuente (y aumentará en el futuro notablemente con la expansión de la web semántica). El comercio electrónico es otro ejemplo donde considerar la presencia de otros agentes competitivos resulta crucial para la optimización de los objetivos propios. Pero también se encuentran sistemas multiagente en entornos tan diversos como grupos de rescate donde agentes robóticos colaboran con otros robots (o humanos); videojuegos en los que compiten varios personajes, humanos y artificiales; campeonatos deportivos entre equipos de robots; o sistemas de control del tráfico de una ciudad, en los que dispositivos como semáforos pueden comunicarse con los semejantes más próximos para organizarse. Las numerosas aplicaciones, en las que surgen distintos ti-

pos de relaciones entre agentes y cuyos entornos presentan características muy diferentes, nos revelan que la existencia de múltiples agentes puede ser un problema a tratar en sí mismo. En este proyecto, hemos querido poder trabajar en ese problema, al menos parcialmente, e investigar las distintas estrategias multiagente que pueden emplearse.

Por otra parte, los videojuegos suponen un ámbito ideal para el análisis de técnicas de inteligencia artificial. Nos aportan entornos muy ricos, que no requieren de material especializado, y permiten una cantidad ilimitada de pruebas. No sólo constituyen bancos de prueba excelentes, sino que facilitan en gran medida la comprensión de los algoritmos probados en ellos al resultar visuales, amenos y fáciles de interpretar. Por ello, un videojuego nos parece la aplicación perfecta para nuestra motivación inicial, el estudio de los sistemas multiagente. En la búsqueda de un videojuego sencillo de tipo claramente multiagente, nos vemos inspirados por la "Ms Pac-Man vs Ghosts Competition"[4] y llegamos al juego de Pac-Man.

1.2. Juego del Pac-Man

Pac-Man es un videojuego de *arcade* ampliamente conocido creado por Toru Iwatani en 1980, siendo uno de los más populares de toda la historia de los videojuegos. Gracias a esta fama, cualquier persona levemente familiarizada con el mundo de los videojuegos tiene una noción de su funcionamiento. El juego tiene un gran número de versiones y variantes, pero la mayoría conserva el mismo planteamiento clásico. En el Pac-Man típico (ver figura 1.1), el jugador controla al personaje Pacman a través de un laberinto, en el cual debe comer las píldoras que en éste se encuentran y evitar a otros cuatro personajes que aparecen, los fantasmas. Además, Pacman puede utilizar cuatro píldoras especiales para comer a los fantasmas durante un periodo de tiempo.

1.2.1. Reglas del juego

Se proporciona una descripción en profundidad del juego. Para este caso, se consideran sólo las reglas básicas esenciales de la versión tradicional del juego.

1. El escenario del juego se trata de un laberinto, que puede entenderse como una cuadrícula $N \times M$.

2. Algunas casillas de la cuadrícula conforman paredes u obstáculos para los personajes.

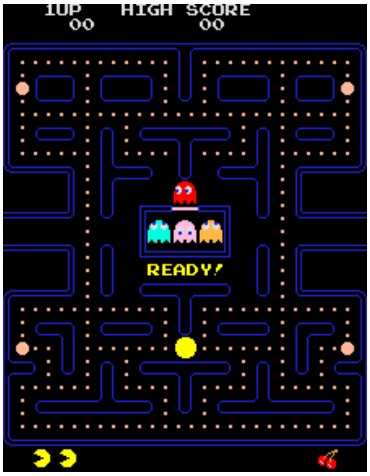

Figura 1.1: Apariencia del Pac-Man clásico

3. El jugador controla al personaje Pacman moviéndolo a través de este escenario con el objetivo de recoger píldoras esparcidas por el laberinto.

4. El movimiento de Pacman a través de la cuadrícula es ortogonal.

5. Se debe comer todas las píldoras para ganar la partida. Por tanto, todas las píldoras en el laberinto son accesibles para Pacman.

6. Existen cuatro personajes enemigos, los fantasmas, que el jugador debe evitar tocar, o en caso contrario perderá una vida.

7. El jugador dispone de tres vidas antes de perder el juego.

8. Cuando el jugador pierde una vida (y mientras aún no haya perdido el juego), Pacman y los fantasmas vuelven a su posición inicial pero se mantienen las píldoras restantes.

9. Los fantasmas se mueven por la cuadrícula de forma ortogonal.

10. El movimiento de los fantasmas por el laberinto es errático.

11. Los fantasmas aparecen inicialmente en un recinto central cerrado y son liberados al laberinto uno a uno tras un intervalo de tiempo determinado.

12. La guarida de los fantasmas, su punto de inicio, no es accesible desde el resto del laberinto.

13. Tanto Pacman como los fantasmas se mueven a la misma velocidad por el laberinto.

14. Los fantasmas sólo pueden cambiar de dirección en un cruce de caminos.

15. En distintos puntos del escenario (habitualmente, las esquinas del mismo) se sitúan píldoras especiales, de mayor tamaño, que darán al jugador la posibilidad de comer enemigos por un intervalo de tiempo.

16. Cuando el jugador recoja una píldora especial, a los fantasmas se les cambia el sentido de su movimiento actual automáticamente.

17. Después de que un fantasma sea comido por Pacman, éste vuelve a la situación de inicio en el recinto central y deja de ser comestible para Pacman hasta se coma otra píldora especial.

18. Cada vez que Pacman recoge píldoras o come fantasmas, recibe puntos en su marcador.

1.3. Objetivos del proyecto

Se enumeran a continuación los objetivos que hemos marcado para este proyecto:

1. Estudiar los conceptos fundamentales de sistemas multiagente e investigar los principales métodos de este campo en la literatura existente.

2. Diseñar e implementar el juego de Pac-Man con sus reglas básicas.

3. Adaptar el diseño e implementación para que el módulo software esté preparado para un uso didáctico en técnicas de inteligencia artificial, y pueda servir de banco de pruebas de dichas técnicas.

4. Diseñar e implementar un equipo de fantasmas multiagente utilizando para ello alguno de los métodos estudiados.

5. Evaluar empíricamente el rendimiento del equipo multiagente en el Pac-Man.

1.4. Procedimiento

Durante este proyecto se han llevado a cabo diversas tareas siguiendo los objetivos dados: de estudio e investigación, de especificación y diseño, de implementación y programación, y de experimentación.

Para las tareas de desarrollo, el software cumple un ciclo de vida **iterativo incremental**[3] organizado en tres iteraciones de diseño e implementación, y una cuarta iteración en la que sólo se efectúa mantenimiento (refactorizaciones). Los cambios en estas distintas iteraciones se señalan en el capítulo 3.

Seguidamente se ofrece una visión general resumida de todo el trabajo, indicando las distintas fases realizadas:

1. **Tarea de documentación y estudio sobre sistemas multiagente**.
   Se localizan referencias sobre el tema y se estudian los conceptos habituales. Se investigan los distintos enfoques y algoritmos para trabajar el problema de múltiples agentes. Los resultados de este proceso se exponen en la sección 2.3.

2. **Búsqueda y análisis de librerías gráficas**.
   Se analizan las ventajas y desventajas de varias opciones para el desarrollo de interfaces gráficas y videojuegos (sección 3.2). Se aprende a utilizar la librería elegida.

3. **Especificación, diseño e implementación del Pac-Man**.
   Se extraen las reglas a considerar del juego de Pac-Man (apartado 1.2.1). Se recogen requisitos sobre el software que se quiere construir (sección 3.3), se realiza el diseño de la aplicación (secciones 3.4 y 3.5), y se codifica el software (sección 3.6).

4. **Análisis del entorno del Pac-Man como problema**.
   Se analiza el juego del Pac-Man como un problema de inteligencia artificial, de cara a diseñar agentes inteligentes para el mismo. Esto es explicado en la sección 2.1.

5. **Estudio de la librería de inteligencia artificial AIMA**.
   Se estudia la estructura de una librería de inteligencia artificial para poder adaptarla a nuestro problema.

6. **Diseño e implementación de la interfaz para la librería AIMA**.
   Se construye una interfaz con la que proporcionar algoritmos de búsqueda predefinidos para nuestro entorno (apartado 3.5.4).

7. **Diseño e implementación de agentes inteligentes básicos**.
   Se organiza una estructura para el diseño de agentes inteligentes y se realizan varios controladores (apartado 3.5.3).

8. **Diseño e implementación de un equipo multiagente**.
   Se construye un equipo de fantasmas multiagente, descrito en la sección 2.4, a partir de técnicas multiagente estudiadas.

9. **Mantenimiento de la aplicación**.
   Se aplican refactorizaciones en el código para dar más claridad al software.

10. **Evaluación sobre el equipo multiagente**.
    Se realizan comparaciones entre varios lotes de pruebas para evaluar el comportamiento del equipo multiagente construido (capítulo 4).

Capítulo 2

Sistemas multiagente en el Pac-Man

A continuación se describen las dificultades del juego del Pacman como problema (desde la perspectiva del diseño de agentes inteligentes), se dan los tipos de agentes considerados y se indican los distintos métodos para el trabajo con sistemas multiagente. Posteriormente, se presenta el sistema multiagente planteado para nuestro problema, un equipo de fantasmas con coordinación.

2.1. Análisis del problema

En el juego del Pacman, su entorno posee una serie de características a tener en cuenta para el diseño de un agente, a saber:

- **determinista**, ya que cada acción de un agente sobre el entorno conduce a un único estado posible;
- **discreto**, tal que el número de estados posibles del juego es finito;
- **totalmente observable**, dado que los agentes pueden percibir por completo el estado actual del entorno (también dicho que el agente posee **información perfecta**);
- **dinámico**, al ser el estado del entorno variable con el tiempo;
- con fuertes restricciones de **tiempo** en el plazo de decisión de un agente;
- y con **múltiples agentes**, tal y como se explica a continuación.

En la versión clásica del juego y de forma habitual se distinguen cinco agentes: el Pacman y cuatro fantasmas. El agente del Pacman tiene el objetivo de comer todas las píldoras del laberinto, evitando ser comido por un fantasma y maximizando su puntuación. Las acciones posibles de este agente son cambiar su dirección a arriba, abajo, izquierda o derecha, o no hacer nada. Por otro lado, todos los fantasmas tienen el objetivo de quitar las vidas al Pacman para que no se coma todas las píldoras, intentando minimizar la puntuación y evitando ser comidos por éste. Los fantasmas poseen las acciones de cambiar de dirección o no actuar, pero sólo cuando estén en un cruce o esquina del laberinto. Esto último incrementa la necesidad de que los fantasmas tomen una decisión en el tiempo proporcionado, dado que en caso contrario pierden la oportunidad de ello hasta llegar al próximo cruce. Cuando un agente no decide en tiempo o decide no actuar, se le asigna automáticamente al personaje su última dirección. Así, continúa su rumbo hasta que se tope con una pared.

A partir de los objetivos descritos de cada agente, se deducen dos relaciones claras entre ellos. Existe una relación de **competitividad** entre el Pacman y el equipo de fantasmas, dado que sus objetivos son opuestos. Además existe otra relación de **cooperación** entre los fantasmas, ya que comparten el mismo objetivo todos.

2.2. Tipos de agentes

Se consideran una serie de tipos de agentes en este trabajo para el equipo de fantasmas:

- **Agente aleatorio**: cada fantasma toma decisiones de forma aleatoria, teniendo en cuenta exclusivamente las posiciones a las que puede moverse (sin muros).
- **Agente racional**[1]: los fantasmas emplean un algoritmo de búsqueda voraz (*Greedy Best-First Search*, o *BFS*[5, c. 3.5.1]), minimizando la distancia de Manhattan[5, c. 3.6] al Pacman como heurística, para simular un comportamiento inteligente en la caza del Pacman a través del laberinto. También tienen en cuenta el efecto de la píldora especial para escapar de Pacman (maximizando esta vez la distancia de Manhattan). No tienen cooperación dentro del equipo.

[1]Se denomina "agente racional" a aquel que trata de optimizar su actuación o comportamiento.[5, c. 2.2]

- **Agente cooperativo**: los fantasmas anteriores (racionales) incluyen técnicas multiagente para dotar de coordinación al equipo, como se explicará en la sección 2.4.

Además, para el Pacman se considera también el siguiente agente inteligente:

- **Agente racional**: busca píldoras por el camino más corto mediante una búsqueda de coste uniforme[2][5, c. 3.4.2], considerando a los fantasmas cómo obstáculos al igual que los muros.

Todos estos tipos se implementarán y se probará su comportamiento en el Pac-Man.

2.3. Métodos multiagente

La aparición de múltiples agentes, con objetivos y acciones que tienen repercusión entre sí, es la propiedad que nos sugiere usar técnicas de sistemas multiagente (MAS) para abordar el problema. Existen diversas estrategias dentro del área de los MAS y se comentan a continuación.

El enfoque más clásico para los MAS es el de la **teoría de juegos**[6, c. 3][7, c. 3]. En este caso, mediante métodos como **IESDA** (Iterated Elimination of Strictly Dominated Actions) o el cálculo de un **equilibrio de Nash** (Nash-Equilibrium, NE) se busca la combinación de acciones de los agentes preferida entre todas las posibles. Para la comparación de estos grupos de acciones se emplean funciones de utilidad. Todas las combinaciones junto a sus beneficios deben ser calculadas por cada agente individualmente en el momento de decisión, con un consiguiente coste computacional elevado. Estos métodos pueden no ser completos, no hallar una solución (en el caso de IESDA), o encontrar más de una combinación posible (en el caso del NE). Además, dado nuestro problema concreto, tienen un espacio de búsqueda demasiado grande como para tolerar un cálculo exhaustivo. Por estas razones, es necesario buscar alternativas que sean más prácticas computacionalmente.

En juegos con competición entre agentes, es habitual el uso de técnicas de **búsqueda con adversario**[5, c. 5] a través de algoritmos como **Minimax**. Sin embargo, en nuestro caso se descartan debido al gran factor de bifurcación que aparecerá en el árbol de búsqueda, aún incluyendo mejoras como **Poda Alfa-Beta** (a causa del número de acciones y al número de

[2]La búsqueda de coste uniforme es igual a una búsqueda A*[5, c. 3.5.2] con heurístico igual a cero.

agentes). El problema resulta intratable usando estos algoritmos sin realizar modificaciones probabilistas en los mismos.

Aparecen otras estrategias multiagente para el problema de **coordinación** entre varios agentes con objetivos no opuestos. En nuestro caso, estas otras pueden ser aprovechadas en el diseño del equipo de fantasmas. Debido a las estrictas limitaciones de tiempo que tiene la decisión de los fantasmas, se descartan aquellas que aplican formas de **comunicación** entre agentes[6, c. 8][8, c. 8]. Entre las técnicas de coordinación sin comunicación se encuentran las convenciones sociales[6, c. 2.4][5, c. 11.4.2] y la asignación de roles[7, c. 4.4][1] como las más eminentes, las cuales hemos empleado para tratar nuestro problema y son expuestas a continuación.

Una **convención social** (o ley social) es una regla o restricción en la elección de acciones de los agentes en un estado dado. De esta forma, eliminando ciertas estrategias para cada agente, se busca evitar combinaciones de acciones conflictivas o poco beneficiosas para el grupo y reducir a un subproblema del problema inicial. La regla es impuesta por igual a todos los agentes que deben coordinarse, y es de **conocimiento común** para todos. Un claro ejemplo de esto son las reglas de conducción para coordinar y mejorar el tráfico en el mundo real. En muchas de estas no existe la comunicación entre los conductores, sino que deben adoptarlas todos por igual; es el caso del sentido de circulación, las señales o los semáforos. Estos elementos reducen las estrategias disponibles de los agentes para el beneficio grupal. La gestión del tráfico es un problema manifiestamente multiagente, y donde no interesa un gran apoyo en la comunicación debido a que esto ralentizaría la circulación y la haría más susceptible a accidentes.
Al momento de aplicar las convenciones sociales, se debe analizar el problema concreto para encontrar reglas que se demuestren efectivas.

La **asignación de roles** consiste en una evolución de las convenciones sociales que proporciona una organización más elaborada y una cierta jerarquía. En este caso, a cada agente se le asocia un **rol**, una clase que reduce sus posibles estrategias en un estado dado, con el objetivo de facilitar la tarea de coordinación y decisión. Dicho de otra forma, cada rol aplica una serie de leyes sociales a aquellos agentes que lo tienen asignado. Y, de nuevo, los agentes deben cumplir siempre las reglas que les impone su rol para lograr el beneficio de la coordinación. Puede existir cualquier número de roles, y cada uno puede ser asignado a ninguno, uno, o más agentes. Como fórmula general, se asume que un agente sólo puede tener asignado un único rol en un instante dado. Un ejemplo donde se aplica esta técnica para el problema de coordinación es en los deportes de equipos. En estos, a los jugadores se les suele adjudicar una figura, que determina parcialmente su comportamiento en el juego y les permite reacciones automáticas a ciertas situaciones.

Con esta técnica, hay dos aspectos que deben adaptarse según el entorno concreto: los roles definidos y el algoritmo o **criterios de asignación**. Para esto último, la asignación puede verse como fija, si los roles tienen una preasignación inicial; o dinámica, en caso de que la asignación dependa de valores de utilidad de los agentes al adoptar un determinado rol. Debe estudiarse la estabilidad que se proporciona para el sistema multiagente en el caso de que la asignación sea de tipo dinámico.

2.4. Aplicación MAS al Pac-Man

En la aplicación de la técnica de asignación de roles al juego de Pac-Man, se especifican los roles planteados, se define el criterio de asignación y se presenta el algoritmo para la toma de decisión resultante.

2.4.1. Roles definidos

Definimos los siguientes roles para el equipo de fantasmas, el grupo de agentes colaborativo:

- Rol de líder, que persigue al Pacman utilizando una búsqueda heurística para el cálculo del camino más corto a través del laberinto. Ignora otros factores del mundo.
- Rol alternativo, que persigue al Pacman intentando seguir una ruta diferente a la de otros fantasmas. Utiliza un algoritmo de búsqueda heurística al igual que el anterior rol, pero tiene la ley social de considerar a cualquier fantasma como un obstáculo más a sortear en su camino.
- Rol de apoyo, que se dirige siempre hacia la zona donde se encuentra el fantasma líder. Utiliza una búsqueda heurística para llegar a la posición del líder.
- Rol evasivo, que trata de evitar cruzarse con el Pacman. Para esto, siempre decide moverse en la dirección con mayor distancia de Manhattan al Pacman.

El último rol definido se concibe para la situación en la que el fantasma es vulnerable a ser comido por Pacman (una vez éste come una píldora especial), mientras que los otros tres roles se adoptan cuando el fantasma puede cazar a Pacman.

2.4.2. Algoritmo de asignación de roles

El criterio de asignación que elegimos para nuestro caso será híbrido. Se proporciona una preasignación fija de los roles a los fantasmas, dándoles un orden inicial (una convención social) y cuatros roles adjudicables: un rol de líder, dos roles alternativos y un rol de apoyo. Durante la ejecución, su rol preasignado se alternará con el rol evasivo cuando el agente se encuentre en peligro de ser comido por el Pacman. Así, el algoritmo de asignación que definimos es el de la figura 2.1.

Figura 2.1 Algoritmo de asignación de roles de tipo híbrido

Entrada:
R el vector (LIDER, ALTERNATIVO, ALTERNATIVO, APOYO)
$p \in \mathbb{N}$ el orden inicial del fantasma, la prioridad de asignación
c el valor booleano del estado del fantasma bajo una píldora especial
Salida:
r el rol asignado

if $c =$ cierto **then**
  $r \leftarrow$ EVASIVO
else
  $r \leftarrow R[p]$ ▷ Rol que le corresponde según su prioridad
end if

2.4.3. Decisión multiagente

Finalmente, se muestra el algoritmo de decisión que se ha elaborado para cualquiera de los fantasmas en la figura 2.2. Nótese en el algoritmo dado que todos las fantasmas implementan y pueden usar la función de decisión relativa a cualquier rol. Esto es, los comportamientos definidos de cada rol son conocimiento común para todos. La función *asignaRol* obedece al algoritmo del apartado 2.4.2.

Con el algoritmo de decisión elaborado y siendo los roles conocimiento común del equipo, resulta inmediato el cambio en el diseño del agente para permitir una **asignación de roles completamente dinámica** según otro valor heurístico como prioridad p, en lugar del orden inicial prefijado que usamos nosotros. Para este caso, debería hacerse una función que calculase heurísticamente la utilidad potencial de todos los agentes para cada rol, y de ahí cada agente asumiría el orden que le corresponde. El algoritmo genérico

de esta asignación que se sugiere puede verse en [7, c. 4]. Sin embargo, para nuestro problema el orden inicial nos proporciona estabilidad y un funcionamiento lo bastante bueno, por lo que no se realizan experimentos con este otro tipo.

Figura 2.2 Algoritmo de decisión de un fantasma coordinado por roles

Entrada:
s el estado actual del juego
r el último rol asignado a este agente
$p \in \mathbb{N}$ el orden inicial del fantasma
c el valor booleano del estado del fantasma bajo una píldora especial
Salida:
a la acción decidida

```
nuevoRol ← ASIGNAROL(p, c)
if nuevoRol ≠ r then
    BORRAMEMORIA()          ▷ Elimina planes anteriores si es necesario
end if

if nuevoRol = LIDER then
    a ← DECIDELIDER(s)
else if nuevoRol = ALTERNATIVO then
    a ← DECIDEALTERNATIVO(s)
else if nuevoRol = APOYO then
    a ← DECIDEAPOYO(s)
else if nuevoRol = EVASIVO then
    a ← DECIDEEVASIVO(s)
end if
```

Los algoritmos presentados también posibilitan una modularidad en los diferentes comportamientos inteligentes definidos; por ejemplo, el único rol que necesita preocuparse por ser vulnerable a Pacman es el evasivo.

Debido a las técnicas utilizadas, los agentes se mantienen autónomos e independientes de otros agentes o de un control central. Esto nos evitará en cada agente esperas y necesidad de sincronización con otros en la toma de decisiones, y les permitirá una respuesta más inmediata. También, al emplear estos métodos relativamente sencillos y ad-hoc, nos permite que todo el poder computacional se concentre en los algoritmos de búsqueda de los fantasmas a través del laberinto del juego.

Capítulo 3

Diseño y desarrollo

En este capítulo se recogen los aspectos relativos al análisis, diseño e implementación del software construido como entorno de pruebas para las técnicas y algoritmos que se describen en el capítulo 2. Dado que el software sigue un ciclo de vida iterativo incremental, se indican los cambios de las distintas iteraciones en el proceso.

3.1. Descripción general

La aplicación software desarrollada implementa una versión básica del juego del Pacman, representando todos los elementos esenciales de éste. El programa debe funcionar como módulo de pruebas para técnicas de inteligencia artificial, y se busca que pueda servir también como módulo educativo de esta materia. Así, éste tiene que modelar el mundo del Pacman, admitir el acoplamiento de un controlador inteligente (para dar vida al Pacman o a los fantasmas) y proporcionar una interfaz de funciones cotidianas para el diseño de un agente en el entorno del juego. Debido a los usos objetivo mencionados, se omiten algunas funcionalidades menores del juego, o de alguna de sus múltiples versiones, que tienen el propósito de aportar mayor entretenimiento y desafío a un jugador humano. Para nuestro caso, introducen una innecesaria complejidad y ruido a la hora de estudiar los algoritmos y analizar el comportamiento de los agentes inteligentes diseñados. Características que se excluyen por estas razones son el aumento gradual de dificultad o de nivel, los cambios de velocidad o la aparición de frutas como bonus extra de puntos para el personaje Pacman. Estas características ya se omitían en el listado de reglas presentado en 1.2.1.

Además, se quiere que la aplicación posea dos modos de ejecución: con simulación gráfica y sin ella. En el primer modo, se mostrará una pantalla

con gráficos 2D en la que podrá verse el progreso de una partida del juego, similar a como aparece convencionalmente el Pacman, para así poder apreciar el movimiento de los agentes a través del laberinto. En el segundo modo, no se lanzará ninguna interfaz gráfica y las partidas se realizarán de forma "silenciosa" en segundo plano, aportándose una ejecución a mayor velocidad. Éste permitirá lanzar lotes de varias partidas seguidas. En ambos modos se guardará los resultados de la ejecución en un fichero externo de texto.

3.2. Tecnologías y herramientas

Para la construcción de la pieza software se precisa la utilización de **Java** como lenguaje de programación. Java se trata de un lenguaje de propósito general, orientado a objetos, concurrente y compilado [2]. Frente a otros lenguajes, nos da la ventaja de que el programa resultante será multiplataforma. Además, éste es el lenguaje principal en las asignaturas de sistemas inteligentes y representación del conocimiento del grado en ingeniería informática, donde nuestro software podrá ser empleado.

En la implementación de la simulación visual, se requiere una librería gráfica que permita gráficos 2D y pueda conectarse con Java. Se contrastan diversas librerías que cumplan esto, como Java Swing, LWJGL, Slick2D y LibGDX, y se escoge esta última. **LibGDX**[1] se trata de un framework para el desarrollo de videojuegos con Java. Proporciona una amplia API para el control de gráficos 2D y 3D, motor de físicas, audio, etc. Es configurable a través de la selección de paquetes, de forma que pueda adaptarse a las necesidades de cada proyecto. Además, esta librería ofrece un nivel de abstracción alto. A diferencia de las otras alternativas, LibGDX permite exportar directamente la misma aplicación desarrollada a plataformas móviles (Android, iOS, BlackBerry), de escritorio (Windows, Mac OS, Linux) y web, manteniendo en todo momento una total independencia de la plataforma o plataformas finales en el desarrollo. Asimismo, el framework ha recibido una creciente popularidad en los últimos años por su uso para videojuegos en dispositivos móviles, lo que brinda una mayor comunidad alrededor de esta tecnología y una documentación más completa.

Por último, se busca una librería de la que obtener diversos algoritmos de búsqueda de caminos ya implementados. Para este propósito, se elige la librería Java del repositorio de código oficial referente al libro académico "*Artificial Intelligence: A Modern Approach*"[5]. Esta librería ofrece una fuente fiable de algoritmos más comunes sobre inteligencia artificial. Nos referiremos a ella a partir de ahora como **librería AIMA**.

[1]Página oficial del framework LibGDX: .

Id	Descripción
RF001	El software implementará las reglas de juego especificadas en 1.2.
RF002	Se debe mostrar el estado y evolución del juego a través de una simulación gráfica.
RF003	El juego tendrá varios personajes (Pacman y fantasmas) que se desplazarán por el escenario.
RF004	El jugador controlará a Pacman mediante el teclado, pudiendo elegir en qué dirección mover al personaje.
RF005	Los fantasmas serán controlados mediante agentes *aleatorios* (sección 2.2) que decidirán sus direcciones.
RF006	El escenario del juego (el laberinto) se cargará desde un fichero de texto externo. De esta forma, para cambiar el escenario sólo es necesario modificar o sustituir el fichero.
RF007	La aplicación mostrará un mensaje para indicar el equipo ganador (Pacman o fantasmas) al final de la partida.
RF008	La aplicación permitirá reproducir una melodía de fondo mientras dure la partida.

Tabla 3.1: Requisitos funcionales de la primera iteración

3.3. Especificación de requisitos

Según las necesidades de nuestra aplicación, descrita en 3.1, y las reglas del juego Pac-Man, se extraen una serie de requisitos funcionales y no funcionales que debe cumplir el software y se enumeran en esta sección.

3.3.1. Primera iteración

Para la primera iteración del software, se trata exclusivamente de modelar el juego Pac-Man y toda su lógica con las reglas definidas (sección 1.2), requiriéndose también la vista de este modelo, la simulación gráfica. Pueden verse los requisitos funcionales y no funcionales en las tablas 3.1 y 3.2 respectivamente.

3.3.2. Segunda iteración

En la segunda iteración del desarrollo, se añaden funcionalidades para satisfacer el objetivo de servir como módulo de pruebas de inteligencia artificial, a través del modo de ejecución sin simulación gráfica, el almacenamiento

Id	Descripción
RNF001	La ejecución de cada agente debe mantenerse independiente de la ejecución del resto y del motor de juego a través de algún modelo de concurrencia.

Tabla 3.2: Requisitos no funcionales de la primera iteración

de resultados y la forma de ejecución de los agentes. Además se introducen mejoras en la eficiencia. Estos requisitos correspondientes se indican en las tablas 3.3 y 3.4 respectivamente.

3.3.3. Tercera iteración

En la tercera iteración se incluirá la implementación del agente inteligente cooperativo descrito en la sección 2.4 y que servirá de controlador para cada fantasma. Además, se requiere facilitar la elección de los controladores a utilizar en la ejecución.

3.4. Diseño de alto nivel

Se presenta el diseño de alto nivel de abstracción para la aplicación especificada en los anteriores apartados del capítulo.

3.4.1. Modelado del problema

En primer lugar, se modela el entorno de juego de Pac-Man. Para ello, se generan clases con las que representar el laberinto (*Maze*), una casilla cualquiera del laberinto (*Cell*), el contenido de una casilla (*CellType*), la acción tomada por un personaje (*AgentAction*), los personajes del juego (*Pacman* y *Ghost*) y el control de la lógica del juego (*GameEngine*). Todo esto se realiza en la **primera iteración** del software.

Se muestra un diagrama de dominio del modelo de nuestro juego en la figura 3.1.

3.4.2. Arquitectura general en capas

Para la arquitectura de la aplicación, se emplea una estructura por capas que nos aporta una mayor modularidad entre las principales partes: la lógica del juego, la representación gráfica y los agentes inteligentes. Las capas de alto

Id	Descripción
RF009	El motor de juego siempre ordenará tomar una decisión a todos los agentes a la vez, y aplicará todas sus acciones de forma simultánea. Este proceso completo constituirá un ciclo de juego.
RF010	Todos los agentes poseerán un tiempo igual fijo para elegir que acción realizar.
RF011	El software podrá lanzarse sin simulación gráfica.
RF012	Se impondrá un límite de ciclos de juego configurable para la ejecución sin simulación gráfica. Pasado el límite, se terminará la partida considerándose empate.
RF013	El software dará la posibilidad de ejecutar varias partidas seguidas cuando no se utilice la simulación gráfica.
RF014	Se debe guardar el resultado de cada partida en un fichero externo. Éste además indicará otros valores relevantes.
RF015	El Pacman podrá ser controlado por una inteligencia artificial racional (sección 2.2) que recoja píldoras por el camino más corto.
RF016	Se incorporará un agente inteligente racional (sección 2.2) para los fantasmas que emplee búsquedas y medidas heurísticas.

Tabla 3.3: Requisitos funcionales de la segunda iteración

Id	Descripción
RNF002	El software debe tener una estructura que facilite el acoplamiento de la implementación de un agente como controlador del Pacman o de los fantasmas.
RNF003	La implementación proporcionará una interfaz de algoritmos de búsqueda adaptados para el entorno definido y heurísticas.
RNF004	Los agentes inteligentes sólo estarán en ejecución cuando existan peticiones pendientes del motor de juego.
RNF005	Se incluirán otros laberintos distintos que poder cargar en el juego.

Tabla 3.4: Requisitos no funcionales de la segunda iteración

Id	Descripción
RF017	Se incorporará un agente cooperativo (sección 2.2) para los fantasmas.
RNF006	El usuario podrá especificar los tipos de agentes a utilizar como controladores de los personajes del juego.

Tabla 3.5: Requisitos funcionales de la tercera iteración

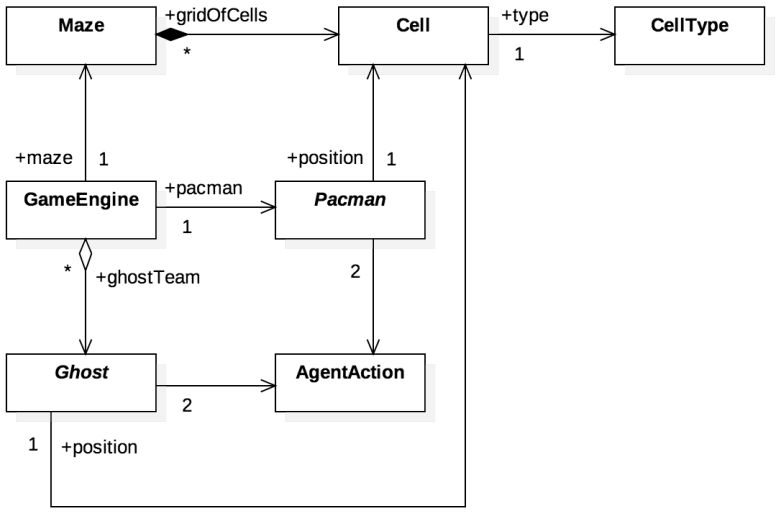

Figura 3.1: Diagrama de dominio del juego

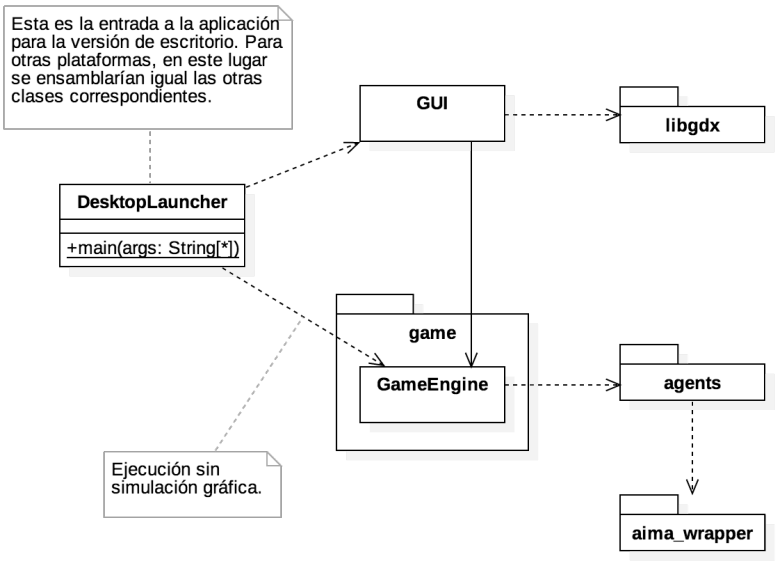

Figura 3.2: Diagrama de arquitectura en capas: paquetes principales

nivel mantendrán una correspondencia con los paquetes en los que se organiza la implementación. En las figuras 3.2 y 3.3 se muestran estos distintos paquetes y su separación en capas.

Así, el paquete *game* agrupará todo el modelo de juego y su lógica, *agents* comprende los elementos para el diseño de agentes inteligentes, *aima_wrapper* proporciona la interfaz de algoritmos de búsqueda (empleando la librería AIMA), y *libgdx* simboliza el framework LibGDX.

La capa de agentes inteligentes y sus respectivos paquetes se añaden a partir de la **segunda iteración** del desarrollo, mientras que el resto de la estructura está presente desde la **primera**. También aparecerá otro paquete más llamado *utils* a partir de la **segunda iteración**.

Todos estos elementos se exponen en detalle en la sección 3.5.

3.5. Diseño detallado

En las siguientes subsecciones se describe el diseño de bajo nivel de las distintas partes del software.

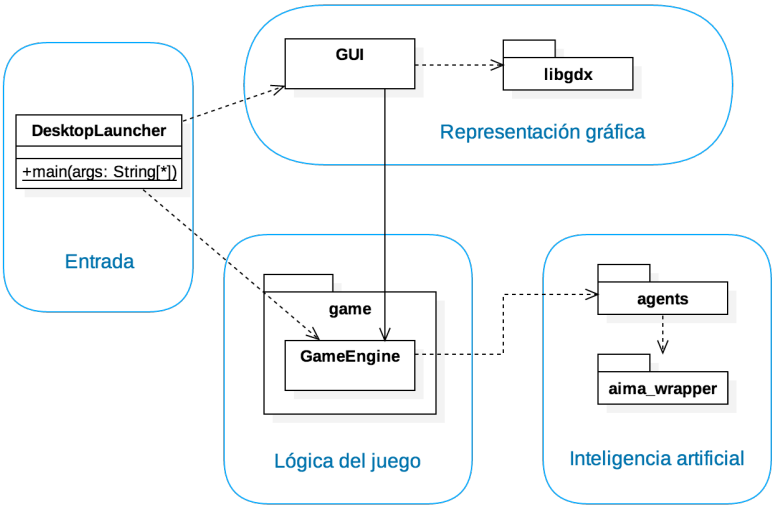

Figura 3.3: Diagrama de arquitectura en capas: distinción de las capas

3.5.1. Motor de juego

Se muestra el diseño con menor abstracción del modelo de juego descrito en 3.4.1 y del control o motor de juego, componentes ubicados en el paquete *game*. Estos están presentes desde la **primera iteración**.

En la figura 3.4 se muestra la organización general del paquete. Se añaden sobre el modelo de juego las clases *GameState*, para representar el estado de juego en un instante, y *GameResult* para representar el resultado de una partida. La clase *GameState* constituirá la información sobre el juego que se dará a un agente para que éste pueda realizar su decisión inteligente. Además, la clase *AgentExecution* controla la ejecución independiente (como hilo Java, heredando la clase *Thread*) de un agente (Pacman o fantasma), y su información del juego es actualizada por *GameEngine*. La clase de *GameEngine* completa se muestra en la figura 3.5.

Las clases *Pacman*, *Ghost* y *AgentAction* del modelo de juego se trasladan en la **segunda iteración** al paquete *agents*, que se explica en el apartado 3.5.3.

3.5.2. Simulación gráfica

Para el diseño de la vista gráfica de la aplicación, se crean las clases *GUI*, *Config* y *AppAssets*, expuestas en la figura 3.6. La clase *GUI* sigue un esquema dado por LibGDX, heredando la clase *Game* suministrada por este

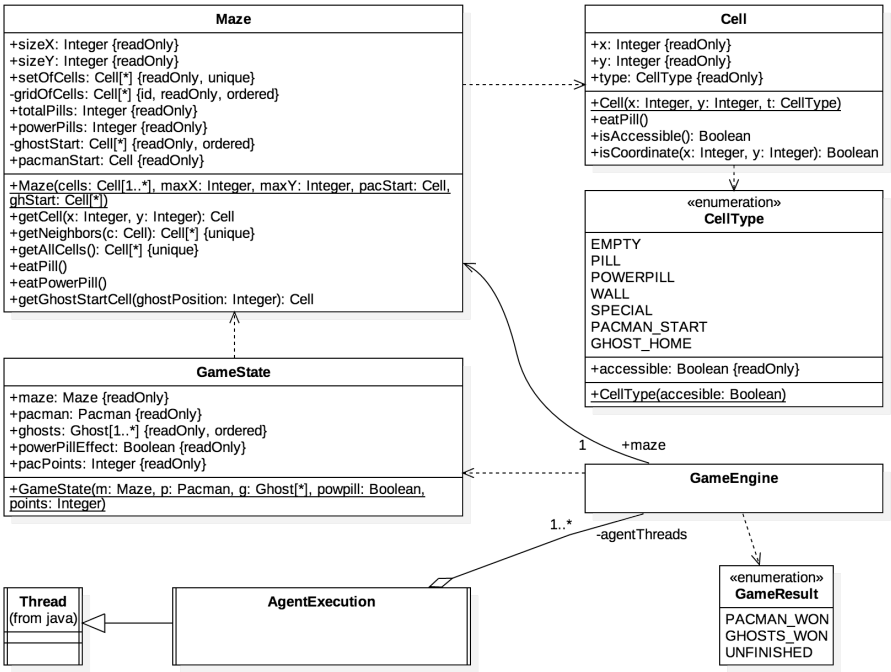

Figura 3.4: Diagrama de clases del modelo de juego

framework. La clase *AppAssets* almacena los nombres de los ficheros necesarios para simulación (las imágenes, o "sprites", de los elementos del juego, ficheros de los laberintos, etc). La clase *Config* posee todas las constantes y valores configurables del juego (como el tiempo de decisión, el límite de ciclos o los controladores inteligentes a utilizar por defecto). Además, el enumerado *AppState* indica los posibles estados de ejecución de la simulación gráfica. Todas estas clases aparecen en la **primera iteración** software.

3.5.3. Jerarquía de los agentes inteligentes

Para facilitar la implementación de agentes inteligentes a nuestro entorno, desde la **segunda iteración** del desarrollo el paquete *agents* organiza todos los elementos necesarios y concibe una jerarquía a la que debe adaptarse la implementación de un controlador. En la figura 3.7 se muestra el diseño del funcionamiento de un agente, incluyendo la clase *AgentExecution* perteneciente al paquete *game*. En la figura 3.8 puede verse cómo la implementación de un controlador inteligente debe heredar de las clases abstractas *Pacman* o *Ghost* (según el personaje a controlar), heredando a su vez ambas de la clase general *Agent*. En este diagrama aparece el agente cooperativo incluido

Figura 3.5: Diagrama en detalle del motor de juego

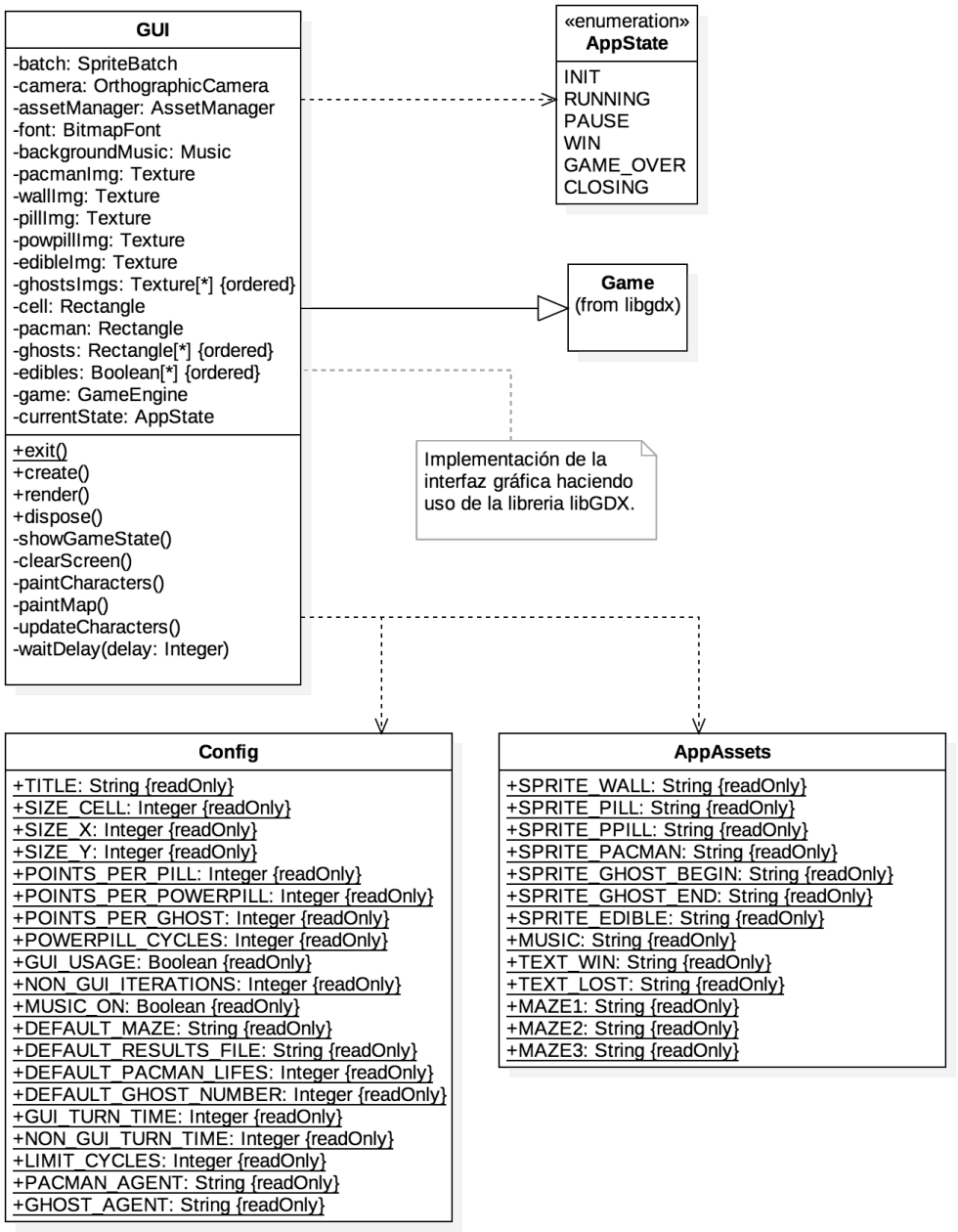

Figura 3.6: Diagrama de clases para la simulación gráfica

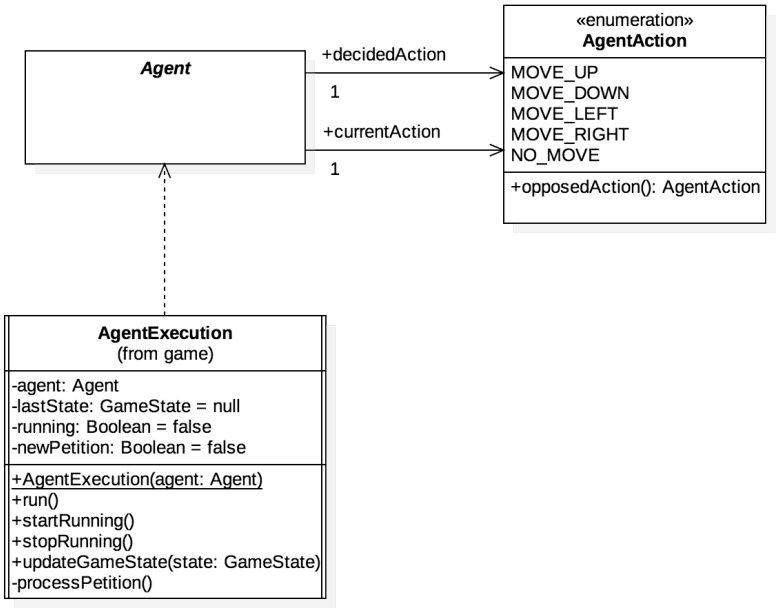

Figura 3.7: Diagrama de clases relativas a los agentes inteligentes

para la **tercera iteración**, *MASGhost*. *HumanPacman* representa al controlador del Pacman humano, a través del teclado; *BasicPacman* y *BasicGhost* representan los controladores racionales.

3.5.4. Interfaz de búsquedas

En la **segunda iteración** software, se diseña la interfaz de algoritmos de búsqueda especificada en los requisitos. Ésta posibilita el empleo de los propios algoritmos implementados en la librería AIMA para nuestro entorno. Para ello, se sigue la estructura que en la propia librería se usa a la hora de modelar un problema. Las clases generadas pueden verse en detalle en la figura 3.9.

La clase *AimaWrapper* provee la interfaz de funciones a emplear de forma externa al paquete *aima_wrapper*, con un gran surtido de opciones para personalizar la búsqueda. La clase *AimaSearch* enumera los algoritmos de búsqueda disponibles.

El resto de clases modelan internamente el problema como lo hace la librería AIMA. *AimaPacGameNode* define un estado de juego (será un nodo en el árbol de búsqueda). *AimaPacFunctionFactory* proporciona las funciones que determinan las acciones disponibles en un estado, y el resultado de

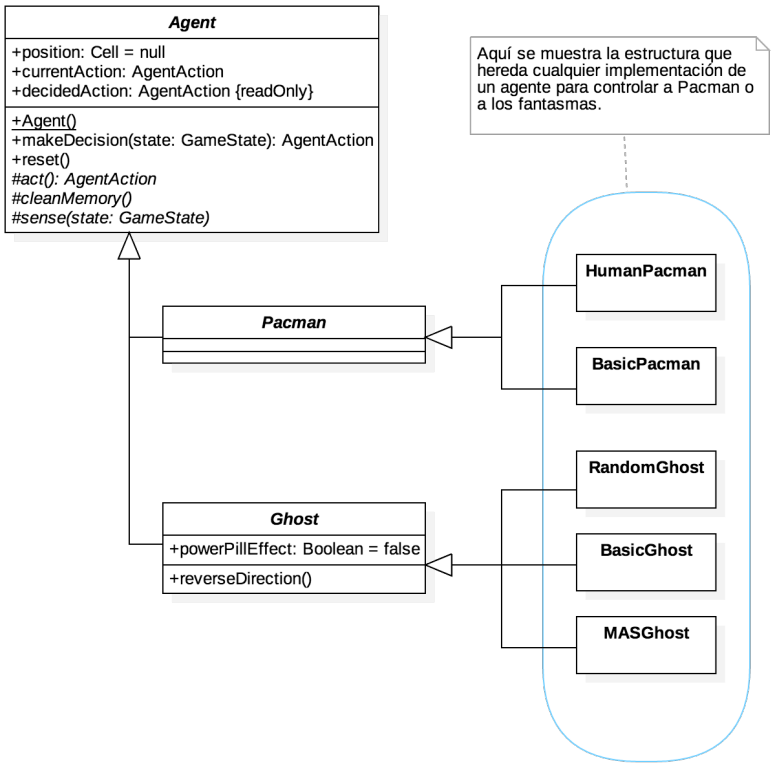

Figura 3.8: Diagrama de la jerarquía de agentes

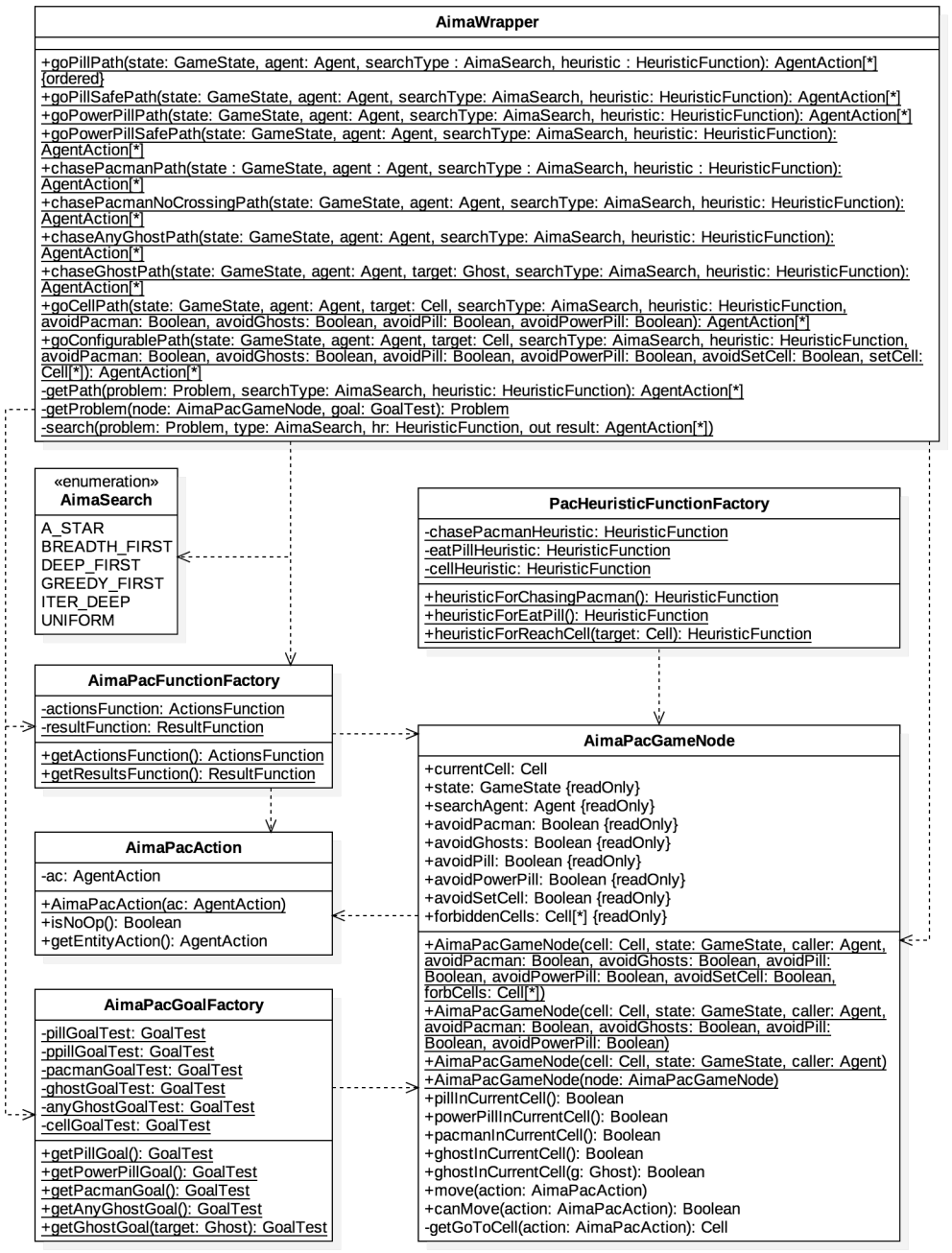

Figura 3.9: Diagrama de clases de la interfaz de búsquedas

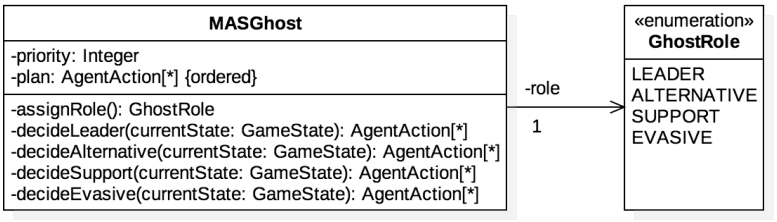

Figura 3.10: Diagrama de clases del agente multiagente

aplicar una acción al estado. La clase *AimaPacAction* modela nuestra acción *AgentAction* (del paquete *agents*) como una acción con el esquema de la librería. *AimaPacGoalFactory* proporciona las funciones que determinan si un estado es final según distintos objetivos definidos. Por último, la clase *PacHeuristicFunctionFactory* proporciona varias funciones heurísticas para nuestro entorno.

3.5.5. Agente multiagente

En la **tercera iteración** se incluye el agente racional cooperativo, *MASGhost*, que se describía en la sección 2.4. Se muestra en la figura 3.10.

3.5.6. Otros módulos

Se añade además un paquete *utils* a partir de la **segunda iteración** del desarrollo. A este paquete se traslada la clase *MazeReader*, encargada de cargar escenarios de juego desde ficheros de texto. También incluye una clase con funciones auxiliares comunes utilizadas desde todas las capas, *ToolFunctions*, y la clase *Tracer*, usada para guardar los resultados de una partida en un fichero externo. Estas clases pueden verse en la figura 3.11.

3.6. Implementación

Aquí se incluyen imágenes del resultado final después de llevarse a cabo la codificación de la aplicación diseñada. En concreto, se presenta la simulación gráfica con la ejecución de los controladores inteligentes implementados, en las figuras 3.12, 3.13 y 3.14. Han sido creados "sprites" propios para la representación gráfica de este juego.

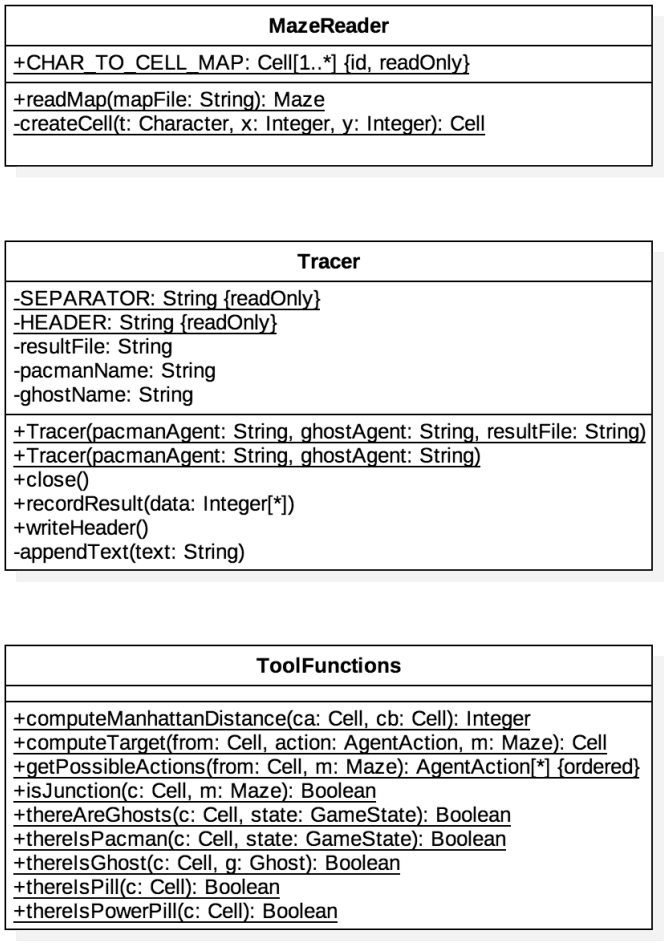

Figura 3.11: Diagrama de clases auxiliares del paquete *utils*

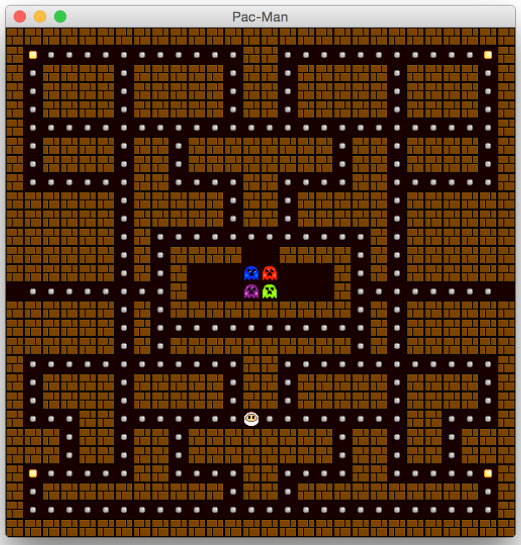

Figura 3.12: Captura de la aplicación implementada: inicio

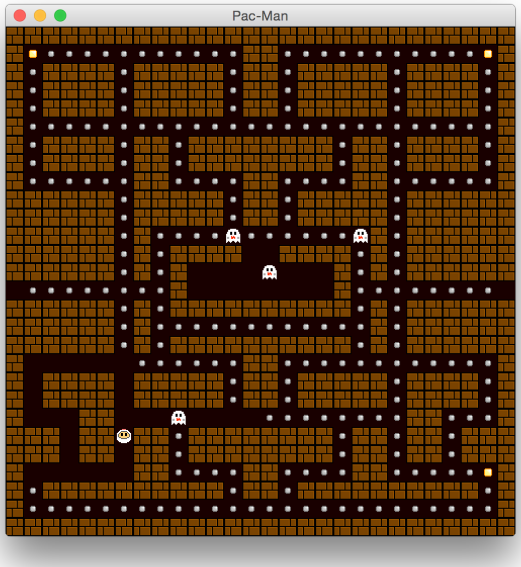

Figura 3.13: Captura de la aplicación implementada: fantasmas vulnerables

Figura 3.14: Captura de la aplicación implementada: fin de una partida

Capítulo 4

Experimentación

En este capítulo se describe el proceso de experimentación y evaluación llevado a cabo para la comparación de los agentes inteligentes implementados.

4.1. Diseño de las pruebas

En primer lugar, diseñamos las pruebas que se llevarán a cabo. Para ello, se determinan los siguientes valores a anotar para la toma de resultados al final de cada partida:

- **Victoria**: se indica si gana la partida Pacman, los fantasmas, o queda en empate.
- **Puntos**: número de puntos que consigue Pacman en la partida.
- **Vidas**: número de vidas que pierde el Pacman durante la partida.
- **Fantasmas**: número de veces que Pacman se come algún fantasma.
- **Turnos**: cantidad de ciclos de juego que dura la partida.

Por otro lado, los agentes considerados para las pruebas a realizar la experimentación se corresponden con los definidos en la sección 2.2, siendo los siguientes:

- **BasicPacman**: controlador racional que emplea una búsqueda para alcanzar píldoras por el camino más corto. Trata de evitar fantasmas en su ruta, pero no va en su captura cuando está activo el efecto de una píldora especial.

Elemento	Puntos
Píldora	10
Píldora especial	50
Fantasma	200

Tabla 4.1: Asignación de puntos en el juego

- **RandomGhost**: controlador aleatorio que elige moverse a una de las casillas adyacentes disponibles (que no son obstáculo) de forma aleatoria.
- **BasicGhost**: controlador racional que emplea una búsqueda para cazar al Pacman mientras el fantasma no sea vulnerable. Cuando éste está bajo el efecto de una píldora especial, elige desplazarse en la dirección con mayor distancia de Manhattan a Pacman.
- **MASGhost**: controlador multiagente elaborado, descrito en la sección 2.4, que además de ser racional emplea la asignación de roles para conseguir cooperación.

Finalmente, se lanzan lotes de 20 pruebas con cada combinación de agentes (tipo Pacman contra tipo Ghost). La duración del efecto de píldora especial se fija a 40 ciclos, y se limita la duración de la partida a 3000 ciclos. Los puntos son proporcionados siguiendo la correspondencia de la tabla 4.1. Como es habitual en el Pac-Man, el número de vidas de Pacman se fija a 3 y el equipo de fantasmas se establece a 4 fantasmas.

4.2. Análisis de resultados

Tras realizarse las pruebas, se obtienen los datos mostrados en las tablas 4.2, 4.3 y 4.4, correspondientes respectivamente al equipo de fantasmas de comportamiento aleatorio (*RandomGhost*), al equipo de fantasmas racionales pero sin coordinación (*BasicGhost*) y al equipo de fantasmas racionales y coordinados (*MASGhost*), todos ellos en competición con el mismo controlador Pacman (*BasicPacman*).

A la vista de estos resultados, se comprueba a *grosso modo* un aumento notable en las victorias del equipo de fantasmas con el controlador *MASGhost*, y también una reducción del número de fantasmas comidos del controlador aleatorio a los dos controladores racionales.

Para poder examinar en profundidad los resultados, se proporciona la tabla 4.5. En esta se resume cada lote de pruebas en una fila y se presentan,

Id	Victoria	Puntos	Vidas	Fantasmas	Turnos
001	Fantasmas	1870	3	0	211
002	Pacman	3230	2	1	412
003	Pacman	3230	1	1	397
004	Fantasmas	3140	3	1	416
005	Pacman	3430	1	2	379
006	Pacman	3430	1	2	402
007	Fantasmas	2730	3	1	330
008	Pacman	3430	0	2	448
009	Fantasmas	3300	3	2	370
010	Pacman	3430	2	2	371
011	Pacman	3030	2	0	393
012	Fantasmas	3530	3	3	380
013	Pacman	3030	1	0	435
014	Pacman	3230	2	1	392
015	Fantasmas	2780	3	0	333
016	Pacman	3230	1	1	429
017	Fantasmas	1680	3	0	199
018	Fantasmas	2870	3	0	378
019	Pacman	3030	1	0	401
020	Pacman	3030	0	0	392

Tabla 4.2: Resultados de las pruebas: *BasicPacman* - *RandomGhost*

Id	Victoria	Puntos	Vidas	Fantasmas	Turnos
001	Pacman	3030	0	0	388
002	Pacman	3030	0	0	398
003	Pacman	3030	1	0	433
004	Fantasmas	2520	3	0	320
005	Pacman	3030	1	0	435
006	Pacman	3030	1	0	417
007	Pacman	3030	0	0	398
008	Fantasmas	2270	3	0	269
009	Fantasmas	2790	3	0	375
010	Pacman	3030	1	0	395
011	Fantasmas	2510	3	0	300
012	Pacman	3030	1	0	405
013	Pacman	3030	2	0	404
014	Pacman	3230	1	1	397
015	Pacman	3030	1	0	432
016	Pacman	3030	1	0	403
017	Pacman	3030	1	0	397
018	Pacman	3030	1	0	380
019	Pacman	3030	2	0	408
020	Fantasmas	2610	3	0	300

Tabla 4.3: Resultados de las pruebas: *BasicPacman* - *BasicGhost*

Id	Victoria	Puntos	Vidas	Fantasmas	Turnos
001	Fantasmas	2660	3	0	313
002	Fantasmas	1350	3	0	168
003	Fantasmas	2160	3	0	254
004	Fantasmas	1750	3	0	204
005	Fantasmas	2080	3	0	236
006	Pacman	3030	2	0	411
007	Fantasmas	2010	3	0	228
008	Fantasmas	1240	3	0	162
009	Fantasmas	2810	3	0	347
010	Fantasmas	2910	3	0	378
011	Fantasmas	2840	3	0	330
012	Fantasmas	2420	3	0	263
013	Fantasmas	2900	3	0	363
014	Fantasmas	1960	3	0	227
015	Fantasmas	2270	3	0	258
016	Fantasmas	1810	3	0	236
017	Fantasmas	1780	3	0	219
018	Fantasmas	2220	3	0	273
019	Fantasmas	2170	3	0	258
020	Fantasmas	2820	3	1	321

Tabla 4.4: Resultados de las pruebas: *BasicPacman* - *MASGhost*

Agente	Victorias	Puntos	Vidas	Fantasmas	Turnos
RandomGhost	40 %	3033.0	1.90	0.95	373.40
BasicGhost	25 %	2917.5	1.45	0.05	382.70
MASGhost	95 %	2259.5	2.95	0.05	272.45

Tabla 4.5: Comparación de los resultados, valores medios

por orden, las siguientes columnas:

- El controlador utilizado para los fantasmas en el lote de pruebas.
- El porcentaje de victorias del equipo de fantasmas sobre Pacman.
- El valor medio de puntos obtenido en el lote.
- El valor medio de vidas perdidas por Pacman.
- El valor medio de fantasmas comidos en el lote.
- El valor medio de turnos empleados.

Frente a la tabla de valores medios dada, podemos hacer un análisis más riguroso de los resultados. Primero, puede verse claramente la superioridad en porcentaje de victorias del controlador coordinado (137.5 % frente al aleatorio, y del 280 % frente al racional no coordinado). De esto podemos deducir que, con la asignación de roles, nuestro propósito se cumple y la distribución de los comportamientos inteligentes es efectiva (en este caso, distribuyéndose los fantasmas por distintas rutas del laberinto). Sobre esto se aprecia también la reducción de puntos obtenidos por Pacman (25 % y 22 % frente al aleatorio y racional respectivamente). También, se observa un cambio que puede parecer extraño, y es la rebaja de victorias del agente racional frente al aleatorio (37.5 %). Esto se debe a que, mientras los agentes aleatorios se reparten por el laberinto aleatoriamente, los racionales terminan agrupados en las mismas zonas. Con ello, estos últimos huyen del Pacman cuando se activa el efecto de una píldora especial, lo que resulta en que todo el equipo termina siempre igual de alejado del Pacman y sin distribución por el laberinto.

Se puede apreciar también la reducción de fantasmas comidos que hay entre el controlador aleatorio y los controladores racionales (94 %). Esto es así dado que tanto el racional como el cooperativo implementan el mismo comportamiento huidizo bajo los efectos de una píldora especial (en el caso del cooperativo, a través del rol evasivo). Además, debe recordarse que nuestro Pacman racional consideraba únicamente el objetivo de comer píldoras intentando evitar a los fantasmas, pero no trataba en ningún caso de cazarlos.

Tanto los valores de vidas perdidas del Pacman, como los de ciclos de juego, son reflejos de los porcentajes de victorias: para que el Pacman no gane, debe perder las tres vidas, y si el Pacman pierde, la partida terminará antes.

Capítulo 5

Conclusiones y trabajo futuro

A continuación exponemos las conclusiones que surgen de la realización del presente trabajo y mencionamos sugerencias para posibles ampliaciones del mismo.

5.1. Conclusiones

De acuerdo con los objetivos planteados para este trabajo (sección 1.3), se han estudiado diversos enfoques de sistemas multiagente, se ha implementado el juego de Pac-Man y se ha elaborado un equipo de fantasmas multiagente mediante técnica de roles.

El estudio del ámbito de los sistemas multiagente no ha sido una tarea fácil debido a la diversidad de enfoques la literatura encontrada con pocos puntos en común y a la falta de soluciones prácticas, en comparación con otros temas de inteligencia artificial de mayor madurez. Hemos necesitado consultar una buena variedad de fuentes para conseguir formarnos una idea general del campo y encontrar soluciones válidas para nuestro problema. Creemos que esto cambiará con el paso de los años y el aumento de la demanda en técnicas multiagente en la resolución de problemas reales, lo que aportará una perspectiva más práctica.

Tras la implementación del Pac-Man y su posterior adaptación para nuestros intereses, apreciamos de nuevo la gran capacidad de los videojuegos, y de éste en concreto, para el estudio y evaluación en técnicas de inteligencia artificial. El Pac-Man nos proporcionó un entorno en el que de forma inmediatamente visual se podía distinguir el comportamiento de los agentes, el desarrollo de sus búsquedas por el laberinto y el buen o mal rendimiento de los algoritmos probados.

En la utilización de la librería AIMA para nuestro problema, hemos dado

con una eficiencia en los algoritmos que no era la esperada. Las búsquedas de esta librería en nuestro entorno nos han dado unos costes y tiempos considerablemente grandes.

Del diseño multiagente del equipo de fantasmas, hemos obtenido una serie de beneficios de índole varia. Por los algoritmos utilizados, conseguimos un grupo de agentes autónomos y que efectivamente consiguen un cierto grado de cooperación. La asignación de roles nos implica un coste computacional mínimo, gracias a su sincronización implícita y a los algoritmos livianos que la implementan, lo que hemos podido valorar en mayor medida debido al elevado coste en nuestras búsquedas. Además, los roles nos han mostrado una separación modular notablemente útil de los comportamientos inteligentes a seguir. Por último, en la evaluación del multiagente, pudimos comprobar el beneficio del diseño al incrementar sus victorias en la caza del Pacman.

Por todo esto, podemos considerar que se han alcanzado las metas pretendidas para este proyecto de forma exitosa.

5.2. Trabajo futuro

Como propuestas para la ampliación de este proyecto, se dan una serie de líneas de trabajo futuro:

- Aplicación de la asignación de roles dinámica (que se menciona en el apartado 2.4.3) y el consiguiente estudio de la estabilidad en la asignación. Para ello será necesario el diseño de una función (o funciones) de utilidad para el problema.
- Sustitución de los algoritmos de búsqueda que se obtienen de la librería AIMA, debido al bajo rendimiento obtenido, por los de otra librería o incluso por implementaciones propias adaptadas al problema concreto.
- Profundización en el ámbito de la teoría de juegos para la busca de técnicas, más avanzadas que IESDA o NE (indicadas en la sección 2.3), que puedan resultar prácticas computacionalmente para este problema.
- Investigación y aplicación en el juego de algoritmos de búsqueda con adversario con modificaciones probabilistas, similares al método MCTS ("*Monte Carlo tree search*") que se menciona en [4] y es habitualmente empleada en la resolución de juegos.
- Aplicación de técnicas de coordinación con comunicación, haciendo un estudio comparativo en la variación de tiempos de decisión y ejecución que éstas puedan incorporar.

- Estudio de los sistemas multiagente y sus técnicas desde el enfoque de la concurrencia, el paralelismo y los sistemas distribuidos, donde los MAS resultan idóneos.
- Aplicación de las técnicas y algoritmos que se han diseñado en este trabajo para otro tipo de problemas y entornos.

Bibliografía

[1] Castelpietra, C. et al. (2000). *Coordination among heterogeneous robotic soccer players*. Proceedings. 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2000), 2.

[2] Gosling, J. et al. (2005). *The Java Language Specification*. Addison-Wesley.

[3] Larman, C., Basili, V. R. (2003). *Iterative and incremental development: A brief history*. Computer, vol. 6, pp. 47-56.

[4] Rohlfshagen, P., Lucas, S. M. (2011). *Ms Pac-Man versus Ghost Team CEC 2011 competition*. In 2011 IEEE Congress of Evolutionary Computation (CEC) (pp. 70–77). IEEE.

[5] Russell, S., Norvig, P. (2009). *Artificial Intelligence: A Modern Approach, 3rd edition*. Prentice Hall.

[6] Shoham, Y., Leyton-brown, K. (2009). *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press.

[7] Vlassis, N. (2007). *A Concise Introduction to Multiagent Systems and Distributed Artificial Intelligence*. Synthesis Lectures on Artificial Intelligence and Machine Learning, University of Amsterdam.

[8] Wooldridge, M. (2002). *Introduction to Multiagent Systems*. John Wiley & Sons, LTD.