**Práctica 3: Pacman Multiagente**

*Fecha de entrega, del 20 al 22 del 11*

Pacman, ahora con más diversión!!!.  
Minimax, Expectimax,  
Evaluation.

**Introducción**

En este proyecto diseñaréis agentes para la versión clásica del Pacman, incluyendo fantasmas. Por el camino, implementaréis los algoritmos de búsqueda minimax y expectimax e intentaréis diseñar vuestras funciones de evaluación.

La base de código no ha cambiado mucho desde el proyecto anterior, pero comenzad instalando el código desde cero en un nuevo directorio. No mezcléis los archivos con los de la práctica 2. Lo que si podéis hacer es utilizar fragmentos del código que hubierais desarrollado en ([search.py](https://campusvirtual2.ub.edu/pluginfile.php/326385/mod_resource/content/5/curs12-13/laboratori/multiagent/docs/search.html) y [searchAgents.py](https://campusvirtual2.ub.edu/pluginfile.php/326385/mod_resource/content/5/curs12-13/laboratori/multiagent/docs/searchAgents.html)) de cualquier forma que consideréis oportuna.

El código para este proyecto contiene los siguientes ficheros, disponibles dentro del archivo [multiagent.zip](https://campusvirtual2.ub.edu/pluginfile.php/326385/mod_resource/content/5/curs12-13/laboratori/multiagent.zip).

**Ficheros clave que debéis leer**

|  |  |
| --- | --- |
| [multiAgents.py](https://campusvirtual2.ub.edu/pluginfile.php/326385/mod_resource/content/5/curs12-13/laboratori/multiagent/docs/multiAgents.html) | Donde residirá todo vuestro código de búsqueda multiagente. |
| [pacman.py](https://campusvirtual2.ub.edu/pluginfile.php/326385/mod_resource/content/5/curs12-13/laboratori/multiagent/docs/pacman.html) | El principal fichero que corre juegos de Pacman. En este fichero se encuentra también el tipoGameState que utilizaréis de forma exhaustiva en este proyecto. |
| [game.py](https://campusvirtual2.ub.edu/pluginfile.php/326385/mod_resource/content/5/curs12-13/laboratori/multiagent/docs/game.html) | La lógica detrás de como funciona el mundo de Pacman. Este fichero incluye diferentes tipos soporte como AgentState, Agent, Direction y Grid. |
| [util.py](https://campusvirtual2.ub.edu/pluginfile.php/326385/mod_resource/content/5/curs12-13/laboratori/multiagent/docs/util.html) | Estructuras de datos útiles para implementar algoritmos de búsqueda. |

**Ficheros que podéis ignorar**

|  |  |
| --- | --- |
| [graphicsDisplay.py](https://campusvirtual2.ub.edu/pluginfile.php/326385/mod_resource/content/5/curs12-13/laboratori/multiagent/docs/graphicsDisplay.html) | Gráficos Pacman |
| [graphicsUtils.py](https://campusvirtual2.ub.edu/pluginfile.php/326385/mod_resource/content/5/curs12-13/laboratori/multiagent/docs/graphicsUtils.html) | Soporte para los gráficos para Pacman |
| [textDisplay.py](https://campusvirtual2.ub.edu/pluginfile.php/326385/mod_resource/content/5/curs12-13/laboratori/multiagent/docs/textDisplay.html) | Gráficos ASCII para Pacman |
| [ghostAgents.py](https://campusvirtual2.ub.edu/pluginfile.php/326385/mod_resource/content/5/curs12-13/laboratori/multiagent/docs/ghostAgents.html) | Agentes que controlan a los fantasmas |
| [keyboardAgents.py](https://campusvirtual2.ub.edu/pluginfile.php/326385/mod_resource/content/5/curs12-13/laboratori/multiagent/docs/keyboardAgents.html) | Interfaz para controlar Pacman desde el teclado. |
| [layout.py](https://campusvirtual2.ub.edu/pluginfile.php/326385/mod_resource/content/5/curs12-13/laboratori/multiagent/docs/layout.html) | Código para trabajar con diferentes layouts |

**¿Qué debéis enviar?** Vosotros debéis rellenar porciones de [multiAgents.py](https://campusvirtual2.ub.edu/pluginfile.php/326385/mod_resource/content/5/curs12-13/laboratori/multiagent/docs/multiAgents.html) durante el proyecto. Debéis enviar este fichero con vuestro código y comentarios. Podéis también enviar ficheros de soporte (como [search.py](https://campusvirtual2.ub.edu/pluginfile.php/326385/mod_resource/content/5/curs12-13/laboratori/multiagent/docs/search.html), etc.) que utilicéis en vuestro código. Por favor *no* cambiéis el resto de ficheros de la distribución o enviéis ninguno de los ficheros originales excepto[multiAgents.py](https://campusvirtual2.ub.edu/pluginfile.php/326385/mod_resource/content/5/curs12-13/laboratori/multiagent/docs/multiAgents.html).

**Pacman multiagente**

En primer lugar, jugad un juego de Pacman clásico:

python pacman.py

Ahora ejecutad el agente ReflexAgent en [multiAgents.py](https://campusvirtual2.ub.edu/pluginfile.php/326385/mod_resource/content/5/curs12-13/laboratori/multiagent/docs/multiAgents.html):

python pacman.py -p ReflexAgent

Observad que juega muy mal incluso en disposiciones sencillas:

python pacman.py -p ReflexAgent -l testClassic

Inspeccionar su código (en [multiAgents.py](https://campusvirtual2.ub.edu/pluginfile.php/326385/mod_resource/content/5/curs12-13/laboratori/multiagent/docs/multiAgents.html)) y aseguraos de que entendéis lo que está haciendo.

***Cuestión 1 (3 puntos)***Mejorar el ReflexAgent en [multiAgents.py](https://campusvirtual2.ub.edu/pluginfile.php/326385/mod_resource/content/5/curs12-13/laboratori/multiagent/docs/multiAgents.html) para que juegue de forma respetable. El código del agente reflejo ofrece algunos ejemplos sobre como consultar el GameState para obtener la información que necesitáis. Un buen agente reflejo deberá considerar la situación de la comida y las posiciones de los fantasmas para funcionar de forma efectiva. Vuestro agente debe ser capaz de superar de forma sencilla y sin problemas la disposición testClassic:

python pacman.py -p ReflexAgent -l testClassic

Probad vuestro agente reflejo en la disposición por defecto mediumClassic con un fantasma o dos (eliminad la animación para dar más velocidad a la visualización):

python pacman.py --frameTime 0 -p ReflexAgent -k 1

python pacman.py --frameTime 0 -p ReflexAgent -k 2

¿Qué tal lo hace? Probablemente muera a menudo con 2 fantasmas a no ser que vuestra función de evaluación sea muy buena.

*Nota:* No podéis tener más fantasmas que los que la [disposición](https://campusvirtual2.ub.edu/pluginfile.php/326385/mod_resource/content/5/curs12-13/laboratori/multiagent/layouts/mediumClassic.lay) permite.

*Nota:* Probad a utilizar como características los recíprocos de valores importantes (como la distancia a la comida) en lugar de los propios valores.

*Nota:* La función de evaluación que estáis escribiendo está evaluando acciones; en partes posteriores del proyecto, evaluaréis estados.

*Opciones:* Los fantasmas por defecto se comportan de forma aleatoria; podéis jugar contra fantasmas ligeramente más inteligentes (direccionales) utilizando -g DirectionalGhost. Si la aleatoriedad os impide saber si vuestro agente esta mejorando, podéis usar -f para ejecutar con una semilla aleatoria fijada (misma semilla -> mismas elecciones aleatorias en cada partida). También podéis jugar varios juegos seguidos con -n. Anulad los gráficos con -q para correr muchos juegos rápidamente.

No perdáis mucho tiempo en esta cuestión, ya que el grueso de la práctica está por llegar.

***Cuestión 2 (5 puntos)***Ahora escribiréis un agente para búsqueda con adversario en la clase MinimaxAgent en[multiAgents.py](https://campusvirtual2.ub.edu/pluginfile.php/326385/mod_resource/content/5/curs12-13/laboratori/multiagent/docs/multiAgents.html). Vuestro agente minimax agent debería funcionar con cualquier número de fantasmas, así que tendréis que escribir un algoritmo ligéramente más general que el que hemos visto en clase de teoría. En particular, vuestro árbol minimax debe tener múltiples etapas min (una para cada fantasma) para cada etapa max.

Vuestro código debe ser capaz de expandir el árbol del juego hasta una profundidad arbitraria. Asignad la puntuación de las hojas del minimax con la función self.evaluationFunction, que por defecto calcula scoreEvaluationFunction.MinimaxAgent extiende MultiAgentAgent, que da acceso a self.depth y self.evaluationFunction. Aseguraos que el código minimax hace referencia a estas dos variables donde sea apropiado, ya que estas variables se rellenan con la información que introduciremos al llamar al juego desde la línea de comandos.

*Importante:* Un turno de búsqueda se considera un movimiento del pacman y la respuesta de todos los fantasmas, así que la búsqueda a profundidad 2 considera 2 movimientos del pacman y de cada fantasma.

***Ayudas y Observaciones***

* La función de evaluación para esta parte ya la tenéis escrita (self.evaluationFunction). No debéis cambiar esta función, pero sí daros cuenta de que estáis evaluando \*estados\* en lugar de acciones, que es lo que evaluabais con el agente reflejo. Agentes con vista-de-futuro evalúan los estados futuros, mientras que los agentes reflejos evalúan únicamente las acciones del estado actual.
* Los valores minimax del estado inicial en la disposición minimaxClassic son 9, 8, 7, -492 para profundidades 1, 2, 3 y 4 respectivamente. Observad que vuestro agente minimax ganará a menudo (de 15 a 20 juegos en nuestro caso) a pesar de la dura predicción de minimax a profundidad 4.

python pacman.py -p MinimaxAgent -l minimaxClassic -a depth=4

* Para incrementar la profundidad de búsqueda disponible para vuestro agente, eliminad la acción Directions.STOP de la lista de acciones posible de Pacman. A profundidad 2 debería ser bastante rápido pero a profundidades 3 o 4 será lento. No os preocupéis en la siguiente cuestión lo aceleraremos un poco.
* Pacman es siempre el agente 0, y los agentes se mueven en orden creciente de índice.
* Todos los estados en minimax deberían ser GameStates, o bien pasados a getAction o generados viaGameState.generateSuccessor. En este proyecto no es necesario abstraer a estados simplificados.
* En tableros más grandes como openClassic y mediumClassic (por defecto), encontraréis que Pacman es bueno en no morir, pero muy malo para ganar. Frecuentemente lo veréis dando vueltas si progresar. Incluso puede darle vueltas a un copo sin comérselo porque no sabe a donde ir después de comerse el copo. No os preocupéis, en la cuestión 5 lo arreglaréis.
* Cuando Pacman cree que su muerte es inevitable, intentará terminar el programa tan pronto como pueda, debido a la penalización por vivir. Algunas veces, contra fantasmas aleatorios, ésta no es la actuación más correcta, pero minimax siempre asume que los contrarios se comportarán de forma óptima:

python pacman.py -p MinimaxAgent -l trappedClassic -a depth=3

Aseguraos de que entendéis por qué Pacman se abalanza al fantasma más cercano en este caso.

***Cuestión 3 (3 puntos)***Hacer un nuevo agente que utilice la poda alfa-beta para explorar el árbol minimax de forma más eficiente, en AlphaBetaAgent. De nuevo, vuestro algoritmo será ligeramente más general que el pseudocódigo visto en clase, así que parte del reto es extender la poda alfa-beta apropiadamente a múltiples agentes minimizadores.

Deberíais presenciar una aceleración (quizá el alfa-beta de profundidad 4 correrá tan rápido como el minimax de profundidad 2).. Idealmente, profundidad 3 en smallClassic debería correr únicamente en unos segundos por movimiento o más rápido.

python pacman.py -p AlphaBetaAgent -a depth=3 -l smallClassic

Los valores minimax del AlphaBetaAgent deberían ser idénticos a los del MinimaxAgent, aunque las acciones que seleccione pueden variar en función del mecanismo de resolución de empates. De nuevo, los valores minimax del estado inicial en la disposición minimaxClassic son 9, 8, 7 and -492 para profundidades 1, 2, 3 and 4 respectivamente.

***Cuestión 4 (3 puntos)***Los fantasmas aleatorios desde luego no son agentes minimax óptimos, y por lo tanto modelizarlos con búsqueda minimax puede no ser apropiado. Rellenad ExpectimaxAgent, donde vuestro agente no tomará el mínimo entre las acciones de los fantasmas, sino la esperanza de acuerdo con el modelo de como se comportan los fantasmas que tenga vuestro agente. Para simplificar vuestro código, asumid que los únicos enemigos existentes son fantasmas RandomGhost que seleccionan entre sus getLegalActions siguiendo una distribución aleatoria uniforme.

Deberíais observar un aproximación más caballerosa a los fantasmas. En particular, si Pacman percibe que él podría estar atrapado pero tiene alguna opción de escapar para comer algo más, como mínimo lo intentará. Investigad los resultados de estos dos escenarios:

python pacman.py -p AlphaBetaAgent -l trappedClassic -a depth=3 -q -n 10

python pacman.py -p ExpectimaxAgent -l trappedClassic -a depth=3 -q -n 10

Deberíais ver que vuestro ExpectimaxAgent gana la mitad del tiempo, mientras que vuestro AlphaBetaAgent siempre pierde. Aseguraos de que entendéis por que el comportamiento aquí difiere tanto del caso minimax.

***Cuestión 5 (6 puntos)***Escribir una mejor función de evaluación para Pacman en la función betterEvaluationFunction. La función de evaluación deberá evaluar estados en lugar de acciones como hacía vuestro agente reflejo. Podéis usar cualquier herramienta a vuestra disposición para la evaluación, incluyendo vuestro código de búsqueda del último proyecto. Con profundidad de búsqueda 2, vuestra función de evaluación debería pasar la disposición smallClassic con dos fantasmas aleatorios más de la mitad de las veces para obtener los 6 puntos manteniendo un tiempo de ejecución razonable.

python pacman.py -l smallClassic -p ExpectimaxAgent -a evalFn=better -q -n 10

¡Documentad vuestra función de evaluación! Tengo gran curiosidad en ver las grandes ideas que tenéis, así que no seais tímidos. Me reservo el derecho de dar puntos extra a las mejores soluciones y mostrarlas en clase.

***Ayudas y observaciones***

* Como para la función de evaluación de vuestro agente reflejo podríais querer usar los recíprocos de valores importantes (como la distancia a la comida) en lugar de los valores mismos.
* Una manera en la que podríais querer escribir vuestra función de evaluación es usando una combinación lineal de características. Es decir, computar valores para diferentes características del estado que creáis importantes para a continuación combinar esos valores multiplicándolos por pesos y sumando los resultados. Debéis decidir multiplicar cada característica por un número que refleje su importancia (para vosotros).

*Se terminó la tercera práctica. ¿Qué más le esperará a Pacman?*