PRÁCTICA 5
Case-Based Reasoning (CBR)
Design Document. Grupo 02

Estructura de los casos
Usaremos los siguientes atributos para la resolución de los problemas que se presentarán a lo largo del juego.

	Tipo	**Atributo**
GENERAL	*int*	levelScore
	int	pillsLeft
	int	powerPillsLeft
	int	edibleGhosts
PACMAN	*int*	livesLeft
	int	distanceToPill
	int	distanceToGhost
	MOVE	lastPCMove
GHOSTS	*int*	distanceToPacMan
	int[]	ghostsPositions
	int	lairTime
	int	distanceToPowerPill
	MOVE[]	lastGhostsMoves
	int	edibleTime

- La representación de un caso, si es deseado o no, lo vamos a simbolizar son valores decimales entre 0 y 1, siendo 0 nada deseado y 1 muy deseado.

Persistencia
La base de casos con respecto al PacMan se dividirá en **distintos** csv para cada uno de los grupos de fantasmas a los cuales se enfrentará el PacMan. Además, dentro de cada csv se realizará una **partición** por estados del PacMan con el objetivo de organizar los casos en función de la situación. Por ejemplo, los casos en los que el PacMan pueda comer o no serán completamente distintos ya que el comportamiento no será prácticamente el mismo. No

hemos considerado la posibilidad de dividir los casos según el nivel ya que queremos que sea lo más genérico posible.

Para el caso de los Fantasmas posiblemente se cree un fichero para cada csv para cada uno de los fantasmas que contenga cada uno de los casos obtenidos por el fantasma en cuestión. Además se tendrá una partición similar a la del PacMan, teniendo en cuenta los estados del Ghost

No obstante, no existirá exclusión para el acceso de los casos para cada sección. Podemos acceder a los casos del PacMan para un problema del Ghost, y viceversa, si es necesario.

Organización en memoria

Para la memoria, se utilizará una organización lineal de los casos, en pos de una mayor efectividad a la hora de recuperar los casos para la resolución de un nuevo problema, siempre apuntando a una búsqueda que sea menor o igual a *O(n)*, siendo n el número de casos. Además como los casos se encuentran organizados de cierta manera, consiguiendo una especie de subdivisión del conjunto de casos, la complejidad de la búsqueda difícilmente será del orden del número de casos.

Recuperación

Para la recuperación de casos en el estado de ***Retrieval*** buscaremos en la base de datos el o los vecinos más similares a la consulta actual, acotando la búsqueda a los atributos que indica el problema, así reducimos el tiempo de búsqueda y nos centramos en los más relevantes y los que más convienen.

Como es obvio, para algunos casos podremos recuperar distintos vecinos en función del problema (ya sea cuando se reciba un caso al comer un fantasma, comer una powerPill, llegar a una intersección, o incluso perder una vida).
Como explicaremos más adelante, en el ***Re*-use** , en principio usaremos una **votación mayoritaria** para elegir la solución más óptima. El número de vecinos que recuperaremos en cada uno de los casos serán cinco, para así facilitar y delimitar la selección.

Para filtrar los casos, indicando si son similares o no, usaremos de manera global la función de similitud *Promedio* para tener en cuenta los pesos de los atributos.
De manera local, usaremos varios tipos de funciones de similitud en función del sub-caso/atributo (univaluado o multivaluado)/problema. Algunas de ellas podrían ser :

- *Interval:* Indica la similitud entre dos variables en base a un intervalo límite de acuerdo con el contexto de tipo de variable.
- *Threshold :* Indica si la similitud de dos variables es alta si su diferencia es menor que un umbral. De nuevo, dicho umbral depende del contexto de la variable.
- *Equals:* Indica si dos variables son iguales. En dicho caso devuelve uno, de lo contrario cero.

Sin embargo no negamos la posibilidad de crear una propia función de similitud que se adapte mejor a las necesidades y al tipo de atributo.

En algunos atributos/parámetros usaremos pesos para dar mayor importancia a la hora de comparar. Por ejemplo para las distancias entre Ghosts y PacMan se usarán diferentes pesos cada tipo de fantasma, ya que no todos se comportarán de la misma manera, o al menos eso asumimos.

Reutilización

Para la selección de una solución con respecto a diferentes vecinos recuperados, usaremos una votación mayoritaria, que indique cuál fue la selección más común entre los vecinos seleccionados. No obstante, no descartamos la posibilidad de migrar o combinar a una **Reutilización Transformacional**, que obtenga la mejor solución entre los vecinos y adaptarla (si hace falta) al problema.

En caso de que la votación indique una selección poco óptima pero muy común se comprobará si la mejor solución entre los vecinos es actualmente la más común.

En caso afirmativo, se tomará una solución aleatoria entre los vecinos. Esto nos permite recopilar más casos para esa solución en concreto, así el algoritmo puede encontrar soluciones óptimas a partir de soluciones no aparentemente tan óptimas.

En caso negativo, se tomará la solución más óptima pero no tan común para comprobar si sigue siendo efectiva.

Revisión

Para la revisión, en función del caso, se comprobará cada 4 intersecciones.

No podemos generalizar la revisión de todos los casos ya que algunos dependen del tiempo o de la distancia, con el objetivo de comprobar si la solución obtenida es óptima para el problema o si debemos reconsiderarlo y cambiarla.

Recuerdo

En principio, las nuevas experiencias adquiridas, tanto con resultados positivos como negativos, se almacenarán en un fichero al final de la partida.

Por otro lado, **no todos** los casos serán almacenados ya que por ejemplo si existe un caso cuya similitud es muy alta con respecto a otro caso, no será de utilidad almacenar dicha información ya que se estará, por así decirlo, repitiendo.