eman ta zabal zazu

Universidad del País Vasco
Euskal Herriko Unibertsitatea

Grado en Ingeniería Informática

Computación

Proyecto de Fin de Grado

Caso de estudio de agentes inteligentes en juegos: Pac-Man Vs. Ghosts

---

Autor

Xabier Garmendia Lasa

informatika fakultatea
facultad de informática

2018

Resumen

El objetivo de este proyecto es implementar y estudiar inteligencias artificiales con diferentes capacidades en el ámbito de los videojuegos. Para ello, se ha hecho uso de una implementación en Python del videojuego clásico *Pac-Man* y se ha seleccionado como agente a estudiar el propio *Pac-Man*.

A lo largo de este documento se explican las distintas versiones de IA que se han implementado y los resultados que ha obtenido cada una de ellas a la hora de completar el juego, usando como medida el tiempo necesitado y el número de vidas perdidas por el agente.

Índice general

Índice de figuras

Indice de tablas

1. CAPÍTULO

Introducción

En el contexto de los videojuegos, la inteligencia artificial basada en agentes consiste en producir personajes autónomos que obtienen información del entorno del juego, determinan que acciones llevar a cabo basándose en esa información y realizan esas acciones. El modelo de la inteligencia artificial en los videojuegos puede dividirse en tres sectores [Millington and Funge, 2009, pp. 8-10]: movimiento, toma de decisiones y estrategia.

En el caso de este proyecto, el movimiento y la toma de decisiones podrían considerarse lo mismo, puesto que las únicas decisiones que puede tomar el agente implementado consisten en moverse. En cuanto a la estrategia, se explorará en las versiones más avanzadas del agente a estudiar.

1.1. *Pac-Man*

Pac-Man es un videojuego que fue lanzado al mercado en 1980. El juego consiste en guiar a un personaje amarillo a través de un laberinto mientras se trata de evitar a cuatro fantasmas que intentan atraparlo. El objetivo es comer los pequeños puntos llamados *Pac-dot* que hay repartidos por todo el laberinto para así transcurrir al siguiente nivel. Este fue unos de los primeros videojuegos en hacer uso de la inteligencia artificial.

Al ser un videojuego bastante antiguo, la inteligencia artificial de los fantasmas es bastante limitada y su estrategia consiste en la mayoría de los casos en ir hacia el lugar donde se encuentra *Pac-Man*. Por lo tanto, estos agentes son conscientes en todo momento del lugar

donde se encuentra su presa, a diferencia de en juegos más modernos. En este proyecto se ha querido cambiar este aspecto del juego, creando para ello una versión modificada del videojuego que se explicará más adelante.

1.2. Herramientas

Todo el proyecto se ha realizado trabajando sobre una implementación en Python 2 de *Pac-Man*, creada por David Reilly y que se puede encontrar en el siguiente enlace: `https://github.com/greyblue9/pacman-python`. Esta implementación hace uso del modulo *Pygame* para generar los gráficos. En la figura 1.1 se puede ver una imagen del juego.

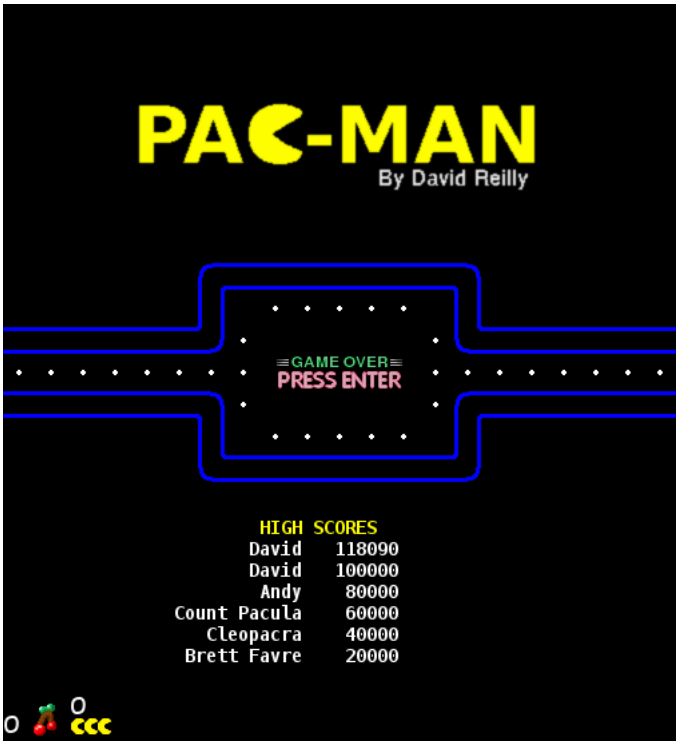

Figura 1.1: Pacman, por David Reilly

No se han usado más herramientas para la realización del proyecto y, por tanto, el programa debería funcionar en cualquier sistema que tenga instalado Python 2.6 - 2.7 y la correspondiente versión de *Pygame*.

1.3. Colaboración

Para la realización de este proyecto, he colaborado con un compañero del grado, Ibai Baglietto, que también ha tratado el tema de la inteligencia artificial en los videojuegos en su proyecto usando la misma implementación de *Pac-Man*, aunque, en su caso, centrándose en la inteligencia de los fantasmas. Para esto hemos colaborado para modificar la versión original de David Reilly, separando el código en diferentes archivos y añadiendo nuevas características. Después, hemos creado diferentes versiones de inteligencia artificial para nuestros agentes de manera individual, haciendo que compitan entre ellas. Por último hemos cooperado para diseñar y realizar las pruebas de las diferentes versiones.

2. CAPÍTULO

Documento de Objetivos del Proyecto

2.1. Alcance

Los objetivos principales del proyecto son los siguientes:

- Aprender diferentes técnicas usadas en la inteligencia artificial para videojuegos.
- Implementar versiones con diferentes capacidades de un mismo agente inteligente en Python.
- Comparar las diferentes versiones usando como medida la eficacia del agente para completar el juego.

2.2. Planificación

A continuación se detallarán las diferentes fases en las que se divide este proyecto, haciendo estimaciones de su duración. Estas estimaciones se pueden observar en la figura 2.1. Como se puede ver, se le da comienzo al proyecto el día 1 de enero y se pretende darlo por finalizado el día 18 de junio.

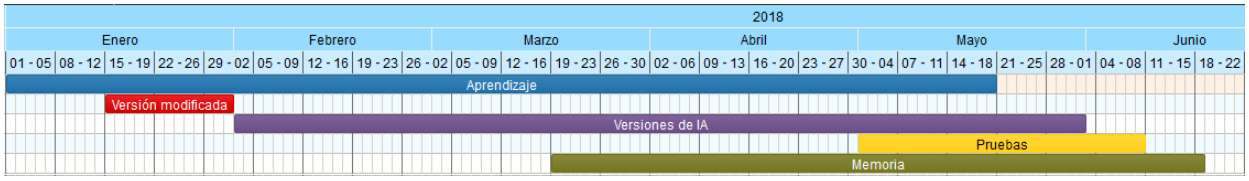

Figura 2.1: Diagrama de Gantt de la planificación del proyecto

2.2.1. Aprendizaje

Una parte importante del proyecto consiste en aprender técnicas usadas en inteligencia artificial. Esta fase será la primera en comenzar y se extenderá durante gran parte del proyecto, de enero a mediados de mayo, ya que a medida que se quieran crean más versiones del agente inteligente hará falta buscar más información.

2.2.2. Diseño e implementación de la versión modificada

Esta fase consiste en colaborar para diseñar e implementar las modificaciones a realizar a la versión de *Pac-Man* creada por David Reilly. Se pretende cerrar esta etapa del proyecto antes de empezar con la que se explicará a continuación, por lo que se estima que ocupará la segunda mitad de enero.

2.2.3. Diseño e implementación de las versiones de la IA

En esta fase se pretende trabajar de forma individual para crear las diferentes versiones de inteligencia artificial que conforman la parte más importante de este proyecto y, por tanto, una de las que más se prolongarán en el tiempo, empezando justo al terminar la versión modificada y finalizando antes de julio.

2.2.4. Diseño y realización de las pruebas

En esta etapa se diseñaran y realizarán de forma colaborativa las pruebas que se harán para comprobar la eficacia de los distintos comportamientos creados para los agentes. Esta etapa comenzará una vez se tengan varias inteligencias creadas, tanto de *Pac-Man* como de los fantasmas, y terminará a principios de julio, unos diez días antes del cierre del proyecto.

2.2.5. Redacción de la memoria

Esta fase comenzará hacia la mitad del proyecto, cuando haya suficiente material para comenzar a escribir la memoria, y finalizará unos dias antes de la fecha limite de entrega del proyecto, dándole cierre de esa forma.

2.3. Riesgos

Hay principalmente dos riesgos en este proyecto. El primero consiste en la pérdida del trabajo o parte del mismo, que es el más fácil de evitar. Debido a la naturaleza colaborativa de parte del proyecto, todo el código generado se guarda en la plataforma *Drive*, reduciendo el riesgo de que se pierda en un descuido. Además de esto, el trabajo individual desarrollado se guarda en dos sitios diferentes, evitando así la perdida del mismo.

El segundo gran riesgo es la falta de tiempo que puede surgir debido a exámenes y trabajos. Este riesgo es el más importante, puesto que es más difícil de prevenir, pero no debería suponer un problema si se organizan bien estas tareas.

3. CAPÍTULO

Desarrollo del proyecto

En la primera sección de este capítulo se habla de la implementación de *Pac-Man* en la que se ha basado este trabajo. La segunda sección detalla la versión modificada de este juego que se ha creado en colaboración con Ibai Baglietto, resumiendo en la siguiente sección las diferentes versiones de IA creadas por él para los fantasmas. Por último, el resto de secciones corresponden a cada una de las versiones de *Pac-Man* creadas por mí, donde se dará una descripción de los diferentes comportamientos y se explorará su implementación.

3.1. Pac-Man, por David Reilly

3.1.1. Descripción del juego

La implementación sobre la que se ha trabajado todo este proyecto es una versión en Python bastante fiel al *Pac-Man* original. El código fue creado por David Reilly y modificado posteriormente por Andy Sommerville.

El juego consiste en dirigir a *Pac-Man* a través del laberinto mediante las teclas de dirección del teclado (o un joystick), mientras se evita a los cuatro fantasmas que intentan capturarlo. Para completar el nivel, es necesario comerse unos puntos llamados *Pac-dot* que hay repartidos por todo el escenario sin perder las tres vidas que se tienen al empezar

el juego. Esta implementación tiene 9 niveles en total, pero se pueden añadir todos los deseados, puesto que el programa los lee de ficheros de texto.

Al igual que el videojuego original, esta versión también posee un sistema de puntuación, que premia al jugador por comerse los pequeños puntos y unas frutas que aparecen de vez en cuando moviéndose por el laberinto. Todo esto se puede ver en la figura 3.1.

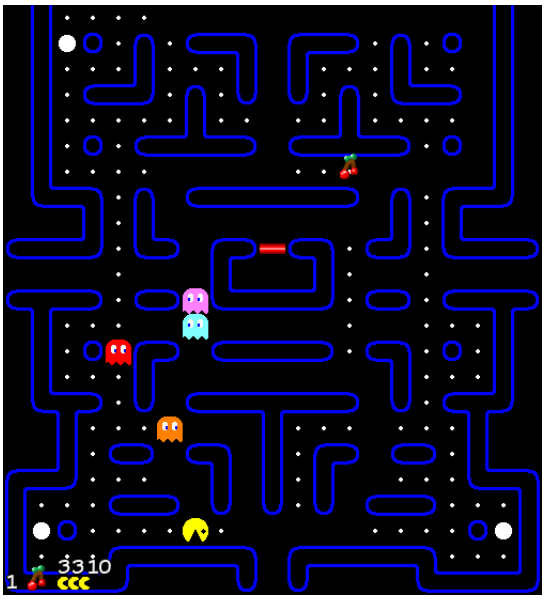

Figura 3.1: Primer nivel de *Pac-Man*. Abajo a la izquierda, el número del nivel, la puntuación y las vidas actuales del jugador. En el laberinto, los cuatro fantasmas, *Pac-Man*, los *Pac-dot* y una cereza.

Además de los *Pac-dot* y la fruta, *Pac-Man* también puede comer unas bolas más grandes que hay en sitios concretos del laberinto, conocidas como *Power Pellet* en inglés. Al comerse una de estas, los fantasmas cambiarán de color durante un tiempo y *Pac-Man* podrá comérselos, sin miedo a perder vidas. En la figura 3.2 se puede ver un ejemplo de esto.

Por último, los laberintos contienen un número par de puertas de teletransporte, situadas al borde del mapa. Estas puertas están conectadas en parejas y si *Pac-Man* accede a una de estas, aparece en la puerta contraria.

En lo que respecta a la inteligencia artificial, los fantasmas usan la búsqueda A^* para localizar a *Pac-Man*, pero solo lo hacen una vez alcanzada la posición anteriormente localizada. Es decir, localizan a *Pac-Man* mediante el algoritmo A^*, siguen el camino necesario para llegar a esa posición y vuelven a ejecutar el algoritmo A^*. Este proceso hace que sea bastante simple evitar a los fantasmas.

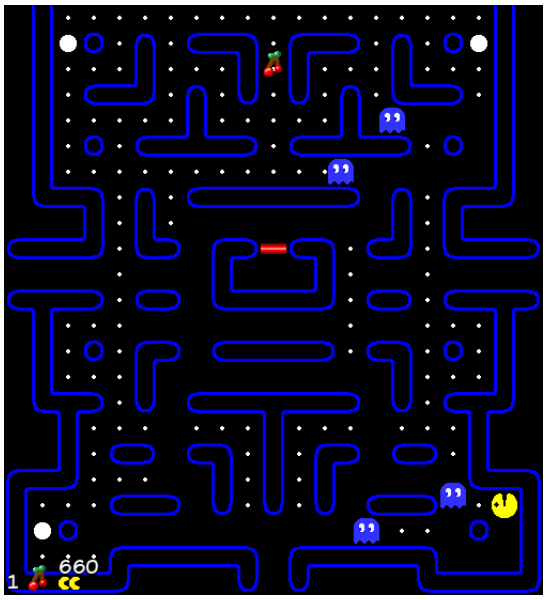

Figura 3.2: Los fantasmas se vuelven de color azul y huyen cuando *Pac-Man* come una *Power Pellet*.

3.1.2. Detalles de la implementación

La implementación del juego consta de las siguientes clases:

- *game*: se encarga de mover la pantalla, avanzar de un nivel a otro y gestionar la puntuación.
- *node*: define los nodos que se usan en la búsqueda A^*.
- *path_finder*: clase usada por los fantasmas para encontrar el camino a su objetivo.
- *ghost*: clase encargada de mover y dibujar a los fantasmas.
- *fruit*: clase encargada de mover y dibujar la fruta por todo el laberinto.
- *pacman*: clase encargada de mover y dibujar a *Pac-Man*.
- *level*: se ocupa de cargar los mapas desde ficheros de texto, dibujarlos, y gestionar la mayoría de cosas relacionadas con los mapas.

Además de estas clases, el código contiene las siguientes tres funciones sin clase:

- *CheckIfCloseButton*.

- *CheckInputs*.
- *GetCrossRef*.

Para este proyecto, se ha creído más conveniente modificar esta implementación para eliminar algunas cosas innecesarias y añadir otras. El resultado de estas modificaciones es lo que se ha denominado la versión modificada del juego.

3.2. Versión modificada

3.2.1. Descripción de la versión

La diferencia principal entre la versión original del juego y esta otra es que los agentes involucrados ya no son omniscientes. Tanto en el *Pac-Man* clásico como en la versión de David Reilly, los fantasmas saben en todo momento donde se encuentra *Pac-Man* y, por tanto, pueden perseguirlo sin problemas. Por otra parte, el jugador puede ver en la pantalla donde están los fantasmas, evitándolos así con bastante facilidad.

Hoy en día, lo más habitual es que los agentes que forman parte de un videojuego no tengan toda la información de su entorno. Por ejemplo, un enemigo puede guiarse por el sonido, el olor o la vista para encontrar a su objetivo, pero solo sabrá donde se encuentra ese objetivo mientras lo detecte mediante uno de esos sentidos. En está versión, hemos intentado simular algo parecido a los videojuegos más modernos, haciendo que los fantasmas imiten a depredadores que se guían por el olfato y la vista y haciendo que *Pac-Man* sea la presa que evita a sus cazadores orientándose mediante el oído. Para ello, *Pac-Man* irá dejando un rastro de olor y sus cazadores producirán un sonido que se expandirá a través del laberinto. Se ha tomado la decisión de que *Pac-Man* no tenga visión al ver que esto le daba una ventaja considerable. En la figura 3.3 se puede ver una muestra de estas nuevas características.

Para esta versión se ha creído innecesaria la puntuación, puesto que a la hora de experimentar se medirá la efectividad de las inteligencias creadas mediante el número de vidas que consume *Pac-Man* para superar el nivel y el tiempo necesitado para ello. Al eliminar la puntuación también se ha eliminado la fruta, al ser un elemento que solo sirve para aumentar los puntos obtenidos.

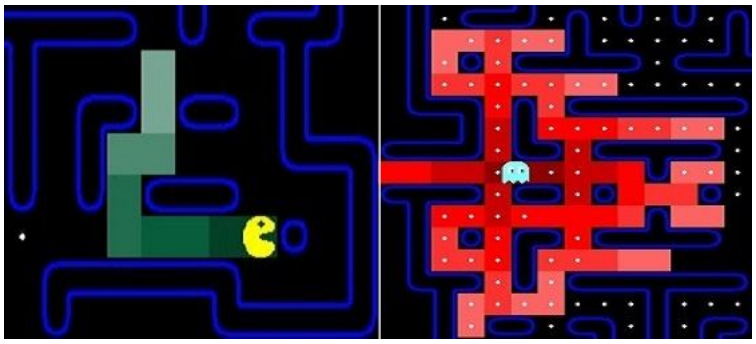

Figura 3.3: A la izquierda, el rastro de olor de *Pac-Man*. A la derecha, la propagación del sonido emitido por el fantasma.

En lugar de las vidas y la puntuación, en esta versión se pueden ver en pantalla las vidas que ha perdido *Pac-Man* y el tiempo transcurrido desde el comienzo del nivel. Estos datos han sido añadidos para facilitar el seguimiento de la eficacia de las capacidades programadas para los agentes. En la figura 3.4 se puede ver como aparecen representados en pantalla.

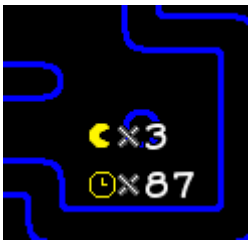

Figura 3.4: El número de muertes de *Pac-Man* y el tiempo transcurrido desde el comienzo del nivel, en segundos.

Se distinguen dos modos de juego en esta modificación del *Pac-Man* clásico. El primero se diferencia del juego original en que solo hay dos fantasmas. Por otra parte, en el segundo modo de juego hay dos *Pac-Man* y tres fantasmas. Cuando un fantasma atrape a una de sus presas en este modo, esta se detendrá hasta que también atrapen a la segunda presa, momento en el que volverán todos a sus posiciones iniciales y se contabilizará una muerte a *Pac-Man*.

Cada modo de juego tiene sus versiones de IA. En lo que respecta a los fantasmas, todas las versiones hasta el número nueve corresponden al primer modo de juego y el resto al segundo. En cuanto a *Pac-Man*, las versiones que se pueden seleccionar en el primer modo se pueden ver en las siguiente secciones: Versión 0, Versión 1, Versión 2, Versión 3, Versión 4 y Versión 5. Por otra parte, en las siguientes secciones se detallan las versiones

disponibles en el segundo modo de juego: Versión 6, Versión 7, Versión 8, Versión 9 y Versión 10.

Para facilitar las comparaciones a la hora de experimentar, hemos decidido reducir el número de niveles a dos. El primer laberinto es casi idéntico al del primer nivel creado por David Reilly, pero se ha modificado para eliminar el recinto desde el que salen los fantasmas al comienzo del juego, puesto que este recinto genera algunos problemas. La figura 3.5 ilustra la diferencia entre ambos laberintos. El segundo laberinto se ha creado desde cero.

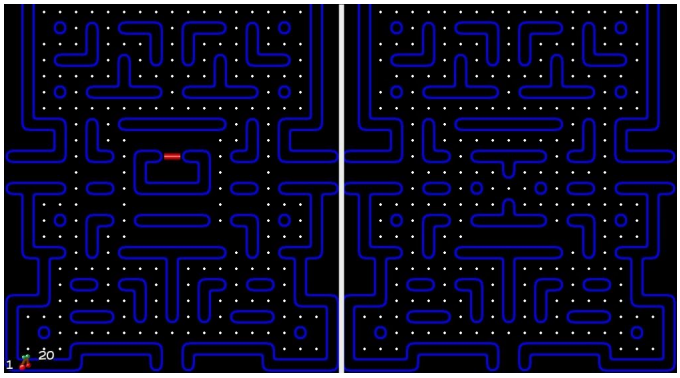

Figura 3.5: A la izquierda, el laberinto original. A la derecha, el laberinto modificado.

En resumen los cambios desde la versión original son los siguientes:

- Se ha añadido:
  - La simulación de un rastro de olor que deja *Pac-Man*.
  - La simulación del ruido que emiten los fantasmas.
  - Dos modos de juego y dos niveles.
  - La posibilidad de elegir diferentes versiones de IA.
  - Información de las vidas perdidas y el tiempo transcurrido.
- Se ha eliminado:
  - La puntuación y todo lo relacionado con ella.
  - La fruta.

3.2.2. Detalles de la implementación

Para esta versión inicial, se ha separado cada clase en un fichero diferente. Esto facilita el proceso de incluir las diferentes versiones de las inteligencias más adelante. A continuación se detallará el contenido de cada fichero, explicando más a fondo los métodos creados por mí.

game.py

Este fichero contiene la clase *game*, que se encarga de mover la pantalla y avanzar de un nivel a otro, entre otras cosas. A esta clase no se le ha añadido nada para esta versión, por lo que a continuación solo se dará una breve descripción de cada uno de sus métodos.

- *game()*: método constructor de la clase.
- *StartNewGame(nivel, jugador, fantasmas, numNivel)*: inicializa el juego, cargando el nivel indicado mediante su número.
- *DrawNumber(numero, (x,y), pantalla)*: dibuja el número precisado en la posición especificada de la pantalla.
- *SmartMoveScreen(jugador, nivel)* y *MoveScreen((x,y))*: se encargan de controlar la zona del laberinto que aparece en la ventana según el lugar donde se encuentre *Pac-Man*.
- *GetScreenPos()*: devuelve la posición absoluta de la pantalla contando desde la esquina superior izquierda del nivel.
- *GetLevelNum()* y *SetLevelNum(num)*: para consultar o modificar el número del nivel actual.
- *SetMode(modo)*: se usa para modificar el modo de juego. Se distinguen los siguiente modos:
  1. Normal.
  2. *Pac-Man* es atrapado por un fantasma.
  3. *Game Over*.
  4. Tiempo de espera al comenzar.

5. Tiempo de espera después de comer un fantasma.
6. Tiempo de espera al terminar el nivel.
7. Parpadeo de la pantalla al terminal el nivel.
8. Pantalla en blanco después del parpadeo.

- *DrawDeaths(pantalla, muertes)*: muestra en pantalla el número de vidas que ha perdido *Pac-Man* en el nivel actual hasta el momento.

- *DrawTime(pantalla, tiempo)*: muestra en pantalla el tiempo que ha transcurrido desde el comienzo del nivel actual, en segundos.

ghost.py

En este fichero se encuentra la clase *ghost*, que se encarga de mover y dibujar a los fantasmas y actualizar el ruido que emiten. El fichero no existe con el nombre indicado, puesto que cada versión de los fantasmas tiene su propio fichero.

Aunque las inteligencias de este agente han sido implementadas por Ibai Baglietto, el método que se encarga de actualizar el ruido que emiten los fantasmas ha sido creado por mí puesto que creímos conveniente que cada uno de nosotros creara las características básicas del agente rival que nuestro propio agente iba a necesitar. El procedimiento se llama *UpdateSound* y su pseudocódigo se puede ver en el apéndice A.1.

Este método genera una matriz del tamaño del laberinto que se guarda en una variable perteneciente a la clase *ghost*. Esta matriz guardará en cada posición el valor que tiene el ruido en cada casilla equivalente del laberinto. El valor del ruido se representa mediante un número que indica la distancia a la que se encuentra la fuente del sonido. Si una baldosa no es afectada por el ruido, tendrá un valor de 100. Se ha tomado la decisión de extender el ruido a diez casillas de distancia, incluidas las casillas en la que se encuentran los fantasmas.

Para conseguir esta matriz, el procedimiento *UpdateSound* realiza una búsqueda en anchura del laberinto, empezando en la casilla actual del fantasma. Este recorrido esta limitado por las paredes del laberinto, que siempre equivalen a 100, y la distancia límite a la que queramos que llegue el ruido.

level.py

Este fichero contiene la clase *level*, que se encarga de gestionar la mayoría de cosas relacionadas con los mapas. Contiene los siguiente métodos:

- *level()*: método constructor de la clase.
- *GetCrossRef()*: decide que baldosas forman el mapa usando un fichero de texto para saber qué imagen corresponde a cada código numérico.
- *SetMapTile((fila,columna), valor)* y *GetMapTile((fila,columna))*: para modificar o consultar la casilla de una posición del mapa.
- *IsWall((fila,columna))*: devuelve *True* si la posición consultada es una pared.
- *CheckIfHitWall((xJugador,yJugador), (fila,columna))*: devuelve *True* si el jugador en la posición especificada está en colisión con una pared.
- *CheckIfHit((xJugador,yJugador), (fila,columna), margen)*: devuelve *True* si el jugador en las posición especificada toca la casilla especificada.
- *CheckIfHitSomething((xJugador,yJugador), (fila, columna), jugador, juego, fantasmas)*: devuelve *True* si el jugador colisiona con un *Pac-dot*, una *Power Pellet* o una de las puertas de teletransporte.
- *DrawMap(juego, pantalla, jugador)*: enseña el mapa en pantalla.
- *LoadLevel(jugador, fantasmas, numNivel)*: carga el mapa desde el fichero correspondiente.
- *Restart(jugador, fantasmas)*: mueve los agentes a sus posiciones iniciales y restaura sus valores a los del inicio.
- *GetTileOdor(fila, columna, jugador)*: devuelve el valor del olor de *Pac-Man* en la casilla especificada.
- *GetTileVisible(fila, columna, jugador)*: devuelve la posición actual de *Pac-Man* si se le puede ver desde la posición indicada.
- *GetTileSound(fila, columna, fantasmas)*.
- *DrawSound(fantasmas, pantalla, imagenes, juego)*.

Los dos últimos métodos de la lista han sido implementados por mí, por lo que se explicarán más detalladamente a continuación.

El primero de ellos, *GetTileSound*, devuelve el valor del sonido en la posición especificada y su pseudocódigo se puede ver en el apéndice A.2.

Como se ha explicado anteriormente, el valor del ruido de un fantasma en una posición corresponde a la distancia a la que se encuentra el fantasma. Por ese motivo, si una casilla del laberinto es afectada por el ruido de ambos fantasmas, *Pac-Man* necesita conocer el ruido de menos valor, puesto que pertenecerá al fantasma que más cerca se encuentra. Ese valor es el que devuelve *GetTileSound*. Por otra parte, se puede apreciar que esta función consulta antes de nada si ya se han calculado las matrices de sonido para evitar errores.

El segundo método es *DrawSound* y su tarea consiste en dibujar las casillas afectadas por el ruido. Para ello, se han usado las cinco imágenes que se pueden ver en la figura 3.6.

Figura 3.6: Cinco tonos diferentes del color rojo.

Se ha duplicado cada una de las cinco imágenes para así asignar cada una de ellas a un valor del ruido del cero al nueve. El pseudocódigo del procedimiento se encuentra en el apendice A.3.

Tal y como se ve, el procedimiento recorre el laberinto casilla por casilla consiguiendo el valor del sonido en cada una de ellas mediante la función *GetTileSound* explicada anteriormente. Una vez conseguido este valor, dibuja la imagen correspondiente en la posición actual. Al devolver *GetTileSound* el valor del sonido mínimo en esa posición, el dibujo resultante es correcto aunque haya dos fantasmas cerca. Esto se aprecia en la figura 3.7.

pacman.py

La clase *pacman* es la que decide que acción debe efectuar *Pac-Man* y, por tanto, es la clase que contiene su inteligencia artificial, aunque no en esta versión. El fichero no existe con el nombre indicado, puesto que cada versión de la IA tiene su propio fichero. La clase básica está formada por los siguientes métodos:

- *pacman*: método constructor de la clase.

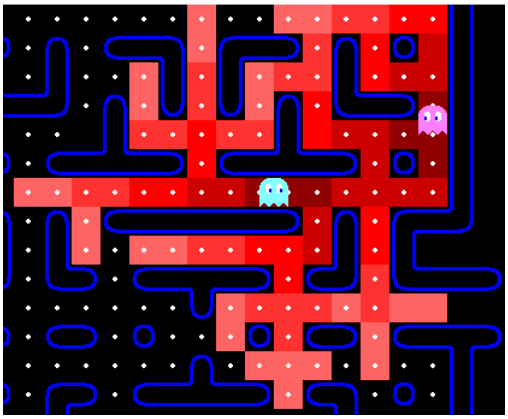

Figura 3.7: Ruido emitido por dos fantasmas.

- *Move(nivel, fantasmas, juego)*: mueve a *Pac-Man* en la dirección adecuada según su velocidad actual, mientras controla las colisiones con los fantasmas.
- *Draw(juego, pantalla)*: dibuja a *Pac-Man* en la ventana del juego, teniendo en cuenta su orientación actual y el movimiento de su boca.
- *Odor()*: genera una lista con las posiciones afectadas por el rastro de olor del agente.
- *DrawOdor(pantalla, imagenes, juego)*: dibuja en pantalla el rastro de olor del agente usando las imágenes proporcionadas.
- *Visible(nivel)*: genera una lista de posiciones desde las que *Pac-Man* es visible para otros agentes.
- *IsNewTile()*.

El último método ha sido añadido en esta versión, y sirve para controlar que el agente haya pasado de una casilla a otra del mapa. Esto es importante para asegurar que las decisiones solo se tomen una vez por cada posición. El funcionamiento se puede ver en el apéndice A.4.

main.py

Este fichero, además de las llamadas a la mayoría de métodos anteriormente mencionados, contiene las siguientes dos funciones:

- *CheckIfCloseButton*: detiene la ejecución del programa cuando el usuario cierra la ventana del juego.
- *CheckInputs*: decide que acciones llevar a cabo cuando el usuario pulsa algunos botones del teclado.

En lo que respecta al código principal, lo primero que hace el programa es inicializar *pygame* y cargar las imágenes necesarias. A continuación, se le pide al usuario que escoja el modo de juego, las versiones de los agentes y el nivel. Dependiendo de las respuestas del usuario, se generarán los agente correspondientes y se cargará el nivel. El resto del fichero consiste en una iteración infinita que llama a diferentes métodos dependiendo del modo en el que se encuentre el programa[1].

3.3. Versiones de los fantasmas

3.3.1. Versión 0

En esta primera versión los fantasmas seguirán a *Pac-Man* si encuentran su rastro de olor o si lo ven. Se considera que un fantasma está viendo a su objetivo cuando ambos comparten fila o columna en el laberinto y no hay ninguna pared entre ellos.

3.3.2. Versión 1

En esta versión los fantasmas empiezan a trabajar en equipo. Cuando un fantasma esté persiguiendo a *Pac-Man*, le pasará información al otro fantasma, dependiendo de si está viendo a la presa o solo oliéndola. Si la está viendo, el otro fantasma sabrá automáticamente donde está *Pac-Man*, pero si solo la está oliendo, el otro fantasma solo recibirá una posición aproximada.

[1] Estos modos ya han sido mencionados en el apartado *game.py* de esta misma sección y no hay que confundirlos con los dos modos de juego que se le dan a elegir al usuario.

3.3.3. Versión 2

Esta versión le suma un nuevo mecanismo a la anterior: la defensa de la comida. Uno de los fantasmas protegerá los *Pac-dot*, intentando moverse a su alrededor y evitando hasta cierto punto que *Pac-man* se los coma. Mientras tanto, el otro fantasma se moverá por el laberinto intentando encontrar a su presa.

3.3.4. Versión 3

La versión 3 es similar a la 2, pero haciendo que cada fantasma se limite a una zona del laberinto. En este caso, los dos fantasmas se dedicarán a proteger los *Pac-dot*, pero uno lo hará en la parte superior del mapa y el otro en la parte inferior.

3.3.5. Versión 4

El cambio en esta versión solo afecta al usarlo contra versiones de *Pac-Man* con la capacidad de usar el poder de las *Power Pellet*. Cuando la presa coma una de estas cápsulas, los fantasmas se volverán vulnerables, por lo que huirán de la presa en lugar de perseguirla.

3.3.6. Versión 5

Esta vez los cazadores tienen la capacidad de bloquearle el paso a la presa en los teletrasnportadores, acercándose al túnel por el que creen que va a salir.

3.3.7. Versión 6

Esta versión añade un cambio en la estructura del juego con la inclusión de las *Ghost Pellet*. Cuando un fantasma consuma una de estas capsulas podrá atravesar un número determinado de paredes. Hay dos subversiones:

Versión 6.1

Esta subversión se basa en la versión 5, de tal forma que usa el poder de las *Ghost Pellet* para llegar antes a bloquearle el paso a la presa en los teleportadores.

Versión 6.2

Esta segunda subversión se basa en la versión 4, por lo que los cazadores usarán el nuevo poder para cortarle el paso a la presa cuando la persigan.

3.3.8. Versión 7

En esta versión se han combinado las dos subversiones anteriores, haciendo que el cazador atraviese paredes para llegar a lo que tenga más cerca de sus dos objetivos (la presa o el teletransportador).

3.3.9. Versión 8

Esta versión es similar a la anterior pero añadiéndole a los fantasmas la posibilidad de cambiar de velocidad. Así, cuando un fantasma esté protegiendo los *Pac-dot* se moverá más despacio, emitiendo menos ruido.

3.3.10. Versión 9

La versión 9 es la última del primer modo de juego. Se basa en la versión 7, con el añadido de que los fantasmas tienen memoria a corto plazo. En versiones anteriores, los cazadores podían usar la visión para detectar a su presa y seguirla mientras la vieran. En esta versión, los fantasmas irán a la posición en la que hayan visto a *Pac-Man* por última vez, aunque no lo estén viendo ya.

3.3.11. Versión 10

Esta versión es la primera del segundo modo de juego y, por tanto, la primera que está pensada para funcionar con tres fantasmas. Se basa en la versión 9, con la diferencia de que esta vez hay un tercer fantasma que recorre todo el laberinto en lugar de limitarse a una zona designada como los otros dos.

3.3.12. Versión 11

Esta versión está pensada para contrarrestar las versiones más avanzadas de *Pac-Man*. En estas versiones, cuando una de las presas muere, su compañera puede resucitarla acercándose a ella. Para evitar esto, un fantasma vigilará al *Pac-Man* muerto, impidiendo así que el otro se acerque.

3.3.13. Versión 12

La versión 12 modifica las prioridades de los fantasmas a la hora de cazar a sus contrincantes. Hasta ahora, estos cazadores iban a por la presa que olían o veían en ese momento. En esta versión, si uno de los fantasmas detecta a una presa, se lo comunicará al resto para que los tres se centren en ella.

3.3.14. Versión 13

En este versión se intenta aumentar la eficiencia de los fantasmas a la hora de bloquearle el paso a *Pac-Man*. Para ello, cuando un fantasma localice a su objetivo, los otros dos decidirán cual de los dos irá al teletransporte por el que creen que podría salir la presa y cual irá directamente hacia ella. Para ello, se le da prioridad al teletranporte, por lo que el que está más cerca irá en esa dirección, dejando como objetivo del otro fantasma cortarle el paso a *Pac-Man*

3.4. Versión 0

3.4.1. Descripción de la versión

Esta primera versión de la inteligencia de *Pac-Man* es la más básica de todas, puesto que lo único que hace es evitar a los fantasmas usando el ruido producido por estos como ayuda. No es capaz de encontrar los *Pac-dot* ni de usar el poder de las *Power Pellet*. Por todos esto motivos, esta versión necesita mucho tiempo para terminar el nivel, puesto que el agente no intenta completarlo, solo trata de sobrevivir.

3.4.2. Detalles de la implementación

Para esta versión se ha añadido el fichero *pacmanV0.py*, donde se llevará a cabo la toma de decisiones.

pacmanV0.py

Se ha añadido el método *DecideSpeed*, que decide la velocidad que tendrá *Pac-Man*, que será lo mismo que decidir la dirección en la que irá. Su pseudocódigo se puede observar en el apéndice A.5.

Lo primero que se hace en este procedimiento es comprobar que *Pac-Man* esté alineado con la casilla en la que se encuentra. Es decir, se evita que una parte de *Pac-Man* atraviese una pared al cambiar de dirección. Después de esto, se comprueba que el agente esté en una casilla nueva. Esto es necesario porque el programa puede llamar a este procedimiento varias veces por casilla y no tendría sentido que *Pac-Man* cambiara de dirección más de una vez por posición.

El resto del programa es equivalente a recorrer el árbol de decisión que se puede ver en la figura 3.8. El árbol de decisión es un modelo que se usa de forma habitual en los videojuegos por la sencillez a la hora de entender. Se puede usar para controlar la toma de decisiones de un agente, como en este caso, o incluso para controlar animaciones. En esta versión, los únicos datos que se usan para responder a las preguntas del árbol son el valor del sonido que detecta *Pac-Man* en el instante de la toma de decisión y el valor que había detectado en la decisión anterior.

Una vez conseguidos estos datos, el agente los usará para saber si está cada vez más cerca o más lejos de la fuente del ruido. En el caso de que se haya acercado al origen del sonido, es decir, a uno de los fantasmas, *Pac-Man* intentará dar la vuelta y huir en dirección contraría. En el caso de que esto no sea posible, tomará uno de los dos caminos perpendiculares a su orientación, intentando así evitar seguir en la dirección que le ha acercado a un fantasma.

Si el agente detecta que el ruido está cada vez más lejos, continuará en la misma dirección siempre y cuando una pared no le bloquee el paso, en cuyo caso tomará uno de los dos caminos perpendiculares a su orientación.

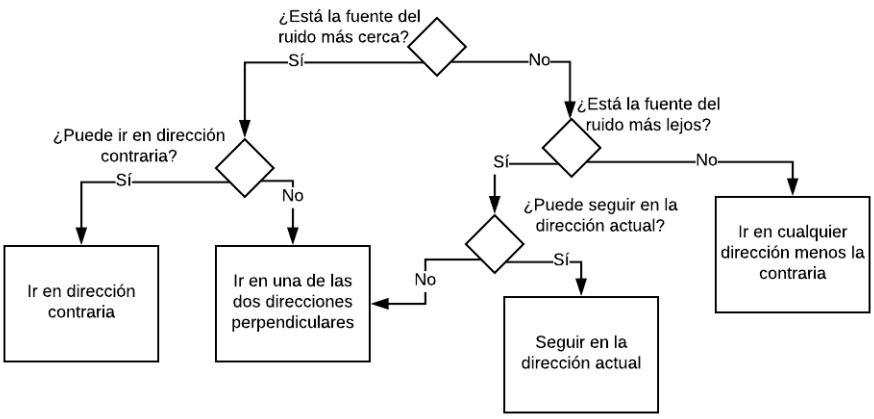

Figura 3.8: Árbol de decisión de la versión 0.

Por último, si la fuente del ruido está a la misma distancia que la última vez, puede ser debido a dos motivos: *Pac-Man* está alejado de los fantasmas y no detecta ningún sonido o uno o dos fantasmas están siguiendo a *Pac-Man*, con lo cual se mantienen a la misma distancia respecto a él. En cualquiera de estos casos, *Pac-Man* elegirá al azar cualquiera de los caminos disponibles, exceptuando el que va en dirección contraria a su orientación. Esto último se debe a que de lo contrario el agente puede quedarse indefinidamente moviéndose hacia delante y hacia atrás o puede acercarse a un fantasma.

3.5. Versión 1

3.5.1. Descripción de la versión

En esta versión, se le ha dado a *Pac-Man* la capacidad de detectar los *Pac-dot* cercanos y comerlos. Así, el objetivo es que el agente necesite menos tiempo para completar el juego. Esto es un gran avance respecto a la versión anterior, porque menos tiempo también significa que los fantasmas tienen menos oportunidades para comerse a *Pac-Man*.

Esta es, en conclusión, la primera versión en la que *Pac-Man* de verdad trata de completar el juego, no solo de huir.

3.5.2. Detalles de la implementación

Para esta versión solo ha sido necesario añadir un nuevo fichero que contiene otra clase *pacman*.

pacmanV1.py

Este fichero se basa en el detallado en el apartado *pacmanV0.py*. Para conseguir el comportamiento explicado anteriormente se ha añadido un nuevo método a la clase y se ha modificado el procedimiento *DecideSpeed*.

El nuevo método se llama *FindNearestFood* y su tarea es encontrar el camino al *Pac-dot* más cercano al agente en un rango limitado de diez casillas. El pseudocódigo se puede ver en el apéndice A.6.

Como se puede ver, la función lleva a cabo un recorrido en anchura del laberinto, que empieza en la posición actual de *Pac-Man* y termina cuando encuentra un *Pac-dot* o llega a la distancia limite de diez casillas. Esta función se utiliza en *DecideSpeed*, que ahora tiene el pseudocódigo del apéndice A.7.

El procedimiento se basa en su homónimo de la versión 0, pero con unos cuantos cambios en el árbol de decisión que recorre. Este árbol se puede ver en la figura 3.9.

El primer cambio realizado es que en esta versión el agente solo continuará alejándose de la fuente del ruido si se encuentra muy cerca. Si no es el caso, le dará prioridad a encontrar los *Pac-dot* y, por tanto, completar el nivel. Para hacer esto, primero comprueba si hay algún *Pac-dot* adyacente y, si es así, procede a comérselo. Si *Pac-Man* no tiene comida adyacente, comprobará si tiene algún camino calculado en una iteración anterior. De no tener ningún camino, intentará encontrar uno llamando a la función *FindNearestFood*. En el caso de que este intento fracase, *Pac-Man* elegirá al azar cualquiera de los caminos disponibles, ignorando el que va en dirección contraria a su orientación.

Si el agente ya había calculado un camino anteriormente o si lo calcula en esta misma iteración, intentará seguirlo, a no ser que vaya en dirección contraria. Con esto se evita que *Pac-Man* vaya hacia un fantasma que le sigue. En este caso, *Pac-Man* elegirá de nuevo de manera aleatoria uno de las direcciones disponibles, exceptuando la que va hacia atrás. Por supuesto, si *Pac-Man* toma cualquier decisión diferente a seguir el camino, hay que eliminar el camino guardado en su memoria, puesto que será inservible cuando se mueva.

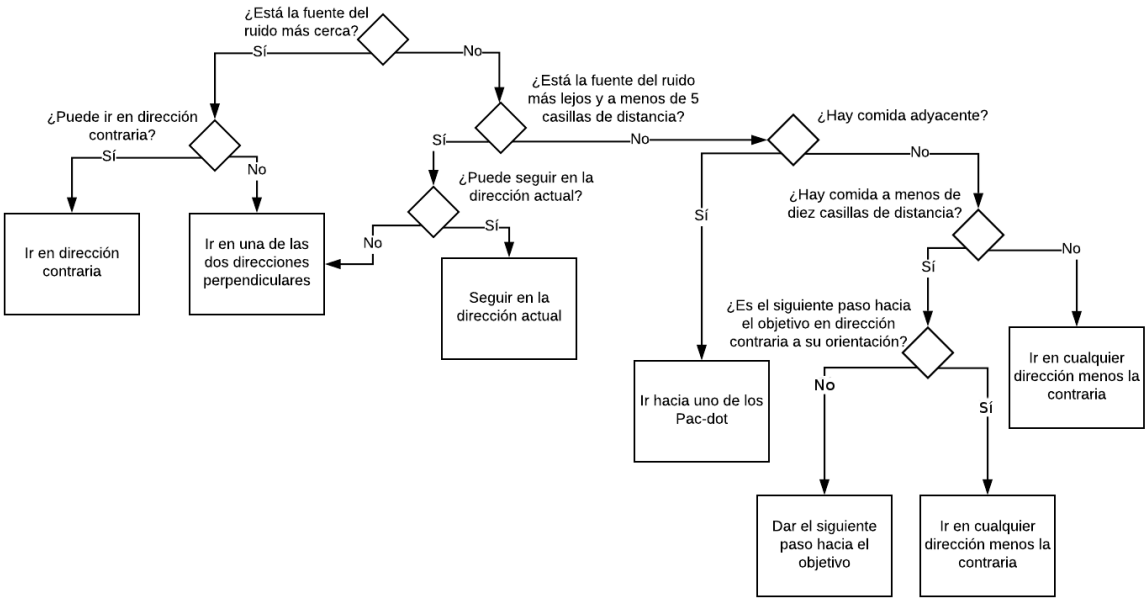

Figura 3.9: Árbol de decisión de la versión 1.

En conclusión, solo dos decisiones del árbol acercan a *Pac-Man* hacia su objetivo final de completar el nivel, pero las pruebas demuestran que es suficiente para conseguir buenos resultados.

3.6. Versión 2

3.6.1. Descripción de la versión

Para la versión 2 se ha puesto el foco en la capacidad de *Pac-Man* de huir de los fantasmas, dotándolo de un nuevo mecanismo de huida, que es el de usar los túneles de teletransporte del laberinto. Aunque nada impide a *Pac-Man* usar estos teletransportadores en las versiones anteriores, no los usa de manera inteligente. En esta nueva versión *Pac-Man* tendrá conocimiento de la dirección a tomar desde cualquier punto del laberinto para poder llegar al túnel más cercano. Solo se dirigirá hacia un teleportador cuando note que uno de los fantasmas le sigue de cerca.

Esto es útil para contrarrestar el trabajo en equipo de los fantasmas, puesto que si evita que un fantasma le siga durante demasiado rato, el otro fantasma no tendrá tiempo de cortarle el paso.

3.6.2. Detalles de la implementación

Para implementar este mecanismo se han hecho cambios en dos ficheros y se ha añadido otro para la nueva clase.

level.py

Aunque hay muchos métodos de *pathfinding* para videojuegos [Millington and Funge, 2009, capítulo 4] [Cui and Shi, 2011b], el más usado es A^*, pero hay casos en los que no es recomendable usarlo. Una de estas situaciones podría ser si hubiera que calcular caminos diferentes para muchos agentes que necesitaran llegar al mismo objetivo. Para estos casos existe un método llamado *flow field pathfinding* o *goal-based vector field pathfinding*.

Aunque en esta ocasión haya que calcular el camino de un solo agente, este método puede ser muy útil, puesto que *Pac-Man* podría tener que esquivar a un fantasma después de calcular el camino, obligándole a hacer los cálculos otra vez. Siendo los túneles elementos estáticos, el método *flow field pathfinding* nos permite hacer los cálculos una sola vez y usarlos durante todo el nivel.

Lo primero que hace este método es calcular las distancia que hay desde el objetivo a cualquier punto del mapa. Para ello se usa un algoritmo de propagación *wavefront*, que es en esencia un recorrido en anchura, y se calcula una matriz con todas las distancias al objetivo. Una vez se tiene esta matriz, también llamada *heatmap*, se usa para conseguir una nueva matriz donde cada posición contendrá un vector con la dirección a seguir estando en ese punto para llegar al objetivo.

Para implementar este algoritmo se han creado dos métodos. El primero se llama *FlowfieldBFS* y calcula la primera matriz para el objetivo solicitado. Su pseudocódigo está en el apéndice A.8.

La función recorre el laberinto entero empezando desde la casilla objetivo e ignorando paredes. A cada casilla le da un valor igual al anterior más uno, consiguiendo así una matriz con todas las distancias al objetivo. A continuación se puede ver la matriz resultante al llamar a la función con la posición de la puerta superior del laberinto:

100	100	100	100	100	100	100	100	100	100	0	100	100	100	100	100	100	100	100	100	100
100	100	100	100	100	100	100	100	100	100	1	100	100	100	100	100	100	100	100	100	100
100	100	10	9	8	7	6	5	4	3	2	3	4	5	6	7	8	9	10	100	100
100	100	11	100	9	100	7	100	100	100	3	100	100	100	7	100	9	100	11	100	100
100	100	12	11	10	100	8	9	10	100	4	100	10	9	8	100	10	11	12	100	100
100	100	13	100	100	100	9	100	9	100	5	100	9	100	9	100	100	100	13	100	100
100	100	14	13	12	11	10	100	8	7	6	7	8	100	10	11	12	13	14	100	100
100	100	15	100	13	100	100	100	100	100	7	100	100	100	100	100	13	100	15	100	100
100	100	16	15	14	13	12	11	10	9	8	9	10	11	12	13	14	15	16	100	100
100	100	100	100	15	100	13	100	100	100	100	100	100	100	13	100	15	100	100	100	100
100	100	100	100	16	100	14	15	16	17	18	17	16	15	14	100	16	100	100	100	100
100	100	100	100	17	100	100	16	100	100	100	100	100	16	100	100	17	100	100	100	100
22	21	20	19	18	19	18	17	18	19	100	19	18	17	18	19	18	19	20	21	22
100	100	100	100	19	100	100	18	100	20	21	20	100	18	100	100	19	100	100	100	100
100	100	22	21	20	21	20	19	20	21	100	21	20	19	20	21	20	21	22	100	100
100	100	23	100	21	100	100	20	100	100	100	100	100	20	100	100	21	100	23	100	100
100	100	24	23	22	100	22	21	22	23	24	23	22	21	22	100	22	23	24	100	100
100	100	100	100	23	100	23	100	100	100	100	100	100	100	23	100	23	100	100	100	100
100	100	100	25	24	25	24	25	26	27	100	27	26	25	24	25	24	25	100	100	100
100	100	100	26	100	100	25	100	100	28	100	28	100	100	25	100	100	26	100	100	100
100	100	100	27	28	27	26	27	100	29	100	29	100	27	26	27	28	27	100	100	100
100	30	29	28	100	100	100	28	100	30	100	30	100	28	100	100	100	28	29	30	100
100	31	100	29	30	31	30	29	30	31	32	31	30	29	30	31	30	29	100	31	100
100	32	31	30	100	100	100	100	100	100	33	100	100	100	100	100	100	30	31	32	100
100	100	100	100	100	100	100	100	100	100	34	100	100	100	100	100	100	100	100	100	100

El segundo método para implementar el algoritmo se llama *FlowfieldPathfinding*. Este método calcula una única matriz de direcciones de manera que en cada posición esté indicada la dirección que debe seguir *Pac-Man* para llegar al teleportador más cercano desde esa posición. La función tiene el pseudocódigo del apéndice A.9.

Lo primero que hace este método es llamar a la función *FlowfieldPathfinding* por cada teleportador del nivel. Después, combinaremos todas estas matrices en una sola, cogiendo el numero más pequeño de cada posición, consiguiendo así un *heatmap* con varios objetivos. En la figura 3.10 se puede ver este *heatmap* superpuesto al laberinto. Las paredes y las posiciones inaccesibles tendrán un valor de 100.

Una vez tengamos esto, solo hay que generar una última matriz que corresponderá al *flow field*. En cada posición de esta matriz se guardará la dirección que hay que seguir para ir hacia un numero más pequeño que el actual en el *heatmap* y, por tanto, la dirección que acercará a *Pac-Man* a uno de los objetivos. En la figura 3.11 se puede ver el *flow field* superpuesto al laberinto.

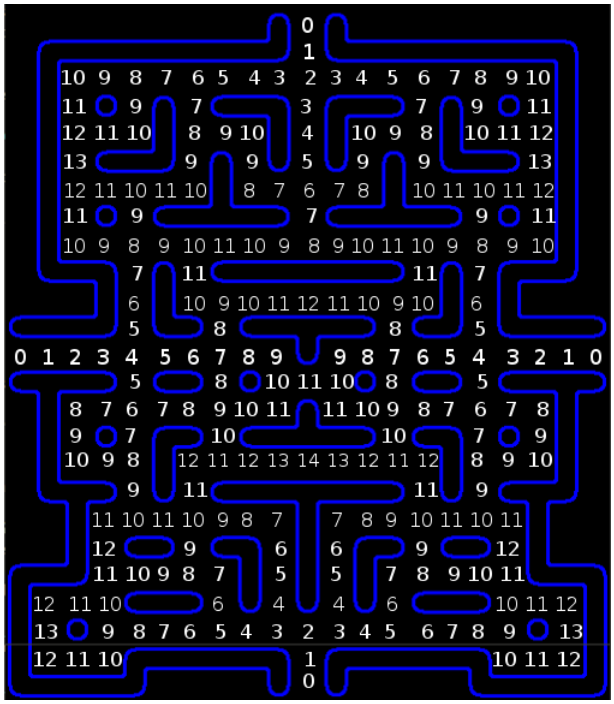

Figura 3.10: *Heatmap* de las cuatro puertas de teletransporte.

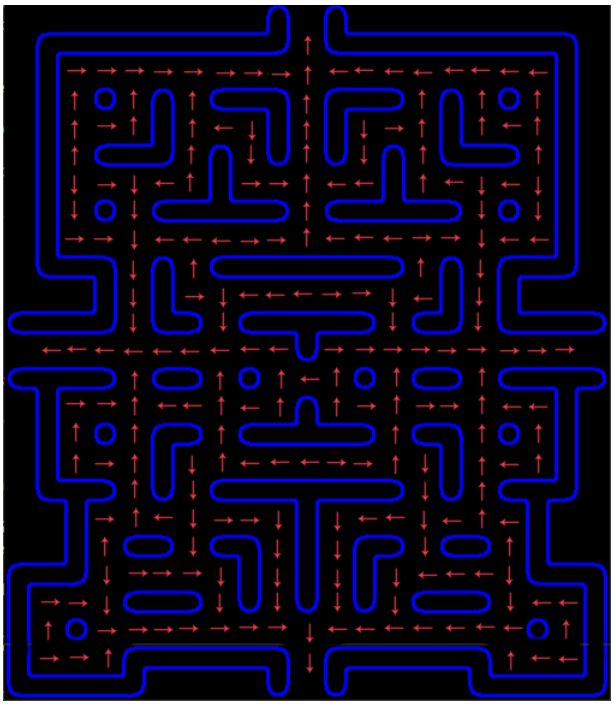

Figura 3.11: *Flow field* de las cuatro puertas de teletransporte.

game.py

En este fichero solo se ha modificado el método *StartNewGame*, haciendo que reinicie la lista de posiciones de los teletransportadores que se guarda en la clase *level* y, si se ha elegido la versión 2 de *Pac-Man* o una versión posterior, haciendo que llame a la función *FlowfieldPathfiding* y guarde el *flow field* en una variable de la clase *pacman*.

pacmanV2.py

Por último, igual que en la versión anterior, se ha añadido un nuevo fichero para la nueva versión del agente. Al tener esta versión como mínimo las mismas capacidades de la anterior, se ha usado el fichero del apartado *pacmanV1.py* como base. El único cambio se ha producido en el procedimiento *DecideSpeed*, para hacer que *Pac-Man* se dirija hacia una puerta teleportadora al sentir que es seguido de cerca por un fantasma. En el apéndice A.10 se puede ver el pseudocódigo de la nueva versión del procedimiento.

El nuevo árbol de decisión se puede ver en la figura 3.12.

Como se puede ver, el nuevo mecanismo de escape de *Pac-Man* se activa cuando la fuente del ruido está a la misma distancia que en la casilla anterior y esta distancia es menor que cinco. En este caso se asume que hay un fantasma siguiendo a *Pac-Man* y, por tanto, que este necesita hacerle perder su rastro. Para ello mirará qué dirección debe seguir para llegar al teletransportador más cercano. Solo seguirá esta dirección si no es contraria a su orientación actual, puesto que el fantasma que le sigue podría estar entre la puerta y *Pac-Man*, en cuyo caso iría hacia el fantasma. Si la dirección a tomar es contraria a su orientación, tomará una dirección aleatoria, exceptuando la contraria. El resto del árbol es idéntico al de la versión 1.

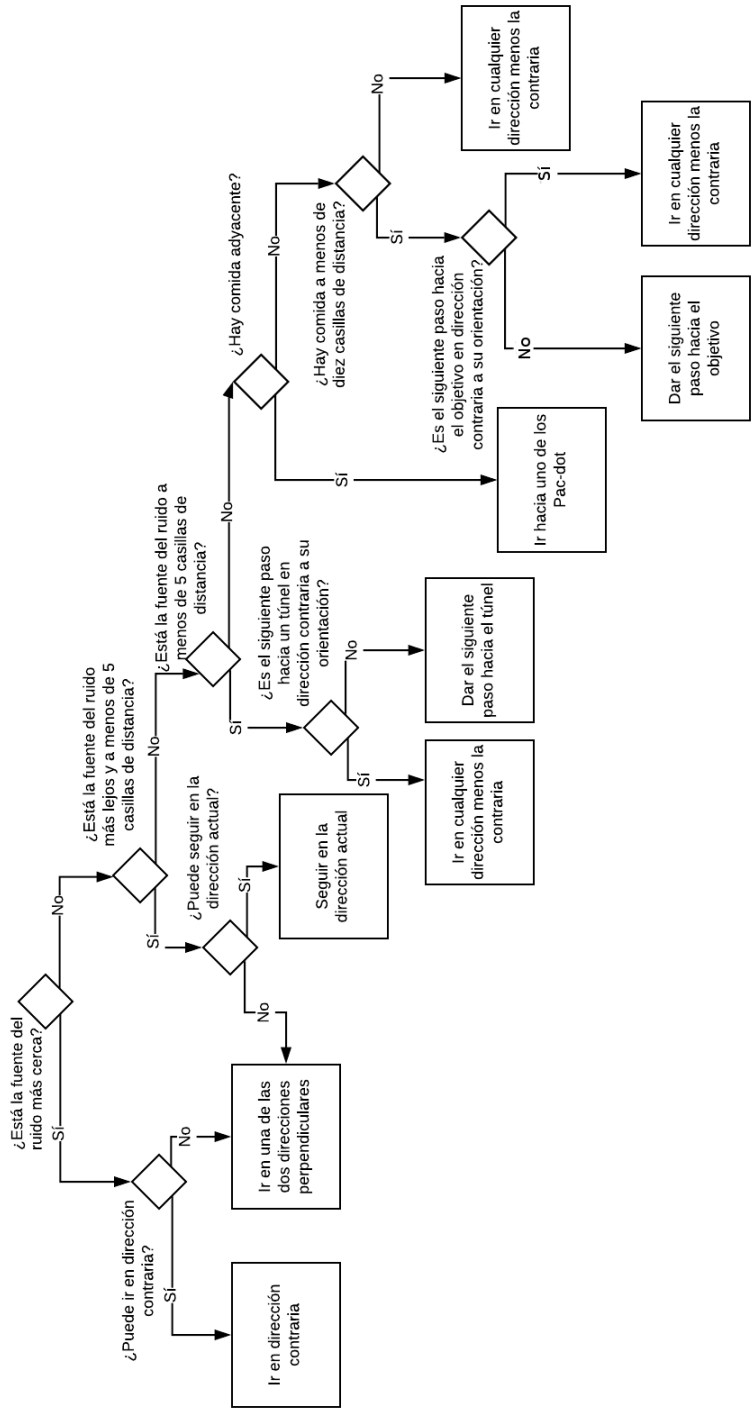

Figura 3.12: Árbol de decisión de la versión 2.

3.7. Versión 3

3.7.1. Descripción de la versión

En esta versión se le da uso por primera vez a las *Power Pellet* que hay por el laberinto. Cuando *Pac-Man* se come una de estas cápsulas, los fantasmas se vuelven temporalmente vulnerables, por lo que, si hay una colisión, *Pac-Man* se comerá al fantasma y no al contrario. Teniendo en cuenta que, al no haber puntuación, comerse a un fantasma no hace que el resultado de *Pac-Man* sea mejor, el agente no los perseguirá cuando estén vulnerables, sino que los ignorará para centrarse en la comida y así terminar el nivel antes.

3.7.2. Detalles de la implementación

Para esta versión se ha modificado un fichero y añadido otro.

level.py

Para que las *Power Pellet* tengan efecto, se ha modificado el procedimiento *CheckIfHitSomething*, añadiendo una condición que se activará si *Pac-Man* colisiona con una *Power Pellet*. Cuando esto ocurra, se pondrá en marcha el temporizador del poder y se pondrá a los fantasmas en estado de vulnerabilidad, siempre y cuando estemos usando la versión 3 de *Pac-Man* o una versión posterior.

pacmanV3.py

Este nuevo fichero es similar al fichero explicado en el apartado *pacmanV2.py*, pero se han modificado los métodos *Move* y *DecideSpeed*. En el primero se ha incluido la posibilidad de que *Pac-Man* colisione con un fantasma vulnerable, en cuyo caso se devolverá al fantasma a su posición inicial, sin que el choque perjudique a *Pac-Man*. Siendo esto así, podría darse el caso de que *Pac-Man* se encontrase con un fantasma en su misma posición inicial y, por tanto, trasladar al fantasma a esta posición solo provocaría que inevitablemente *Pac-Man* continuara colisionando hasta llegar el temporizador de vulnerabilidad del fantasma a cero. Para evitar que se de esta situación, se comprueba que la posición

actual de *Pac-Man* no coincida con la posición de origen del fantasma en cuestión y, si coincide, el fantasma será trasladado a la posición de origen de su compañero fantasma.

También se ha modificado este mismo método para que gestione el temporizador de vulnerabilidad de los fantasmas, reduciendo el tiempo en cada iteración y haciendo que los fantasmas vuelvan a su estado original al acabarse el mismo.

El procedimiento *DecideSpeed* se ha modificado de forma que el agente ignorará el sonido emitido por los fantasmas cuando estos estén en estado de vulnerabilidad. El pseudocódigo se encuentra en el apéndice A.11.

El árbol de decisión de la nueva versión se puede ver en la figura 3.13. Como se puede observar, se ha añadido una nueva condición a la que se le da prioridad. Para mirar si esta condición se cumple o no, se usa el temporizador de vulnerabilidad, asumiendo que si este tiene un valor igual o menor a cero, los fantasmas son invulnerables y *Pac-Man* debería evitarlos. Por lo demás, el comportamiento es idéntico al de la versión anterior.

3.8. Versión 4

3.8.1. Descripción de la versión

Uno de los problemas que surge a medida que *Pac-Man* se enfrenta a fantasmas más inteligentes, es que el agente se encuentra más a menudo en situaciones de riesgo y, por tanto, la mayoría de sus decisiones están enfocadas a huir y no a completar el nivel, aumentando el tiempo necesario para esto último. Para evitar esto, se ha creado esta versión donde *Pac-Man* tiene menos miedo a los fantasmas, pudiendo acercarse más a ellos y a los *Pac-dot* que haya cerca de los mismos.

3.8.2. Detalles de la implementación

Para esta versión se ha incluido un nuevo fichero con la nueva clase *pacman*.

pacmanV4.py

Este nuevo fichero es muy parecido al fichero explicado en el apartado *pacmanV3.py*, habiendo modificado únicamente el método *DecideSpeed*. Su pseudocódigo se encuentra

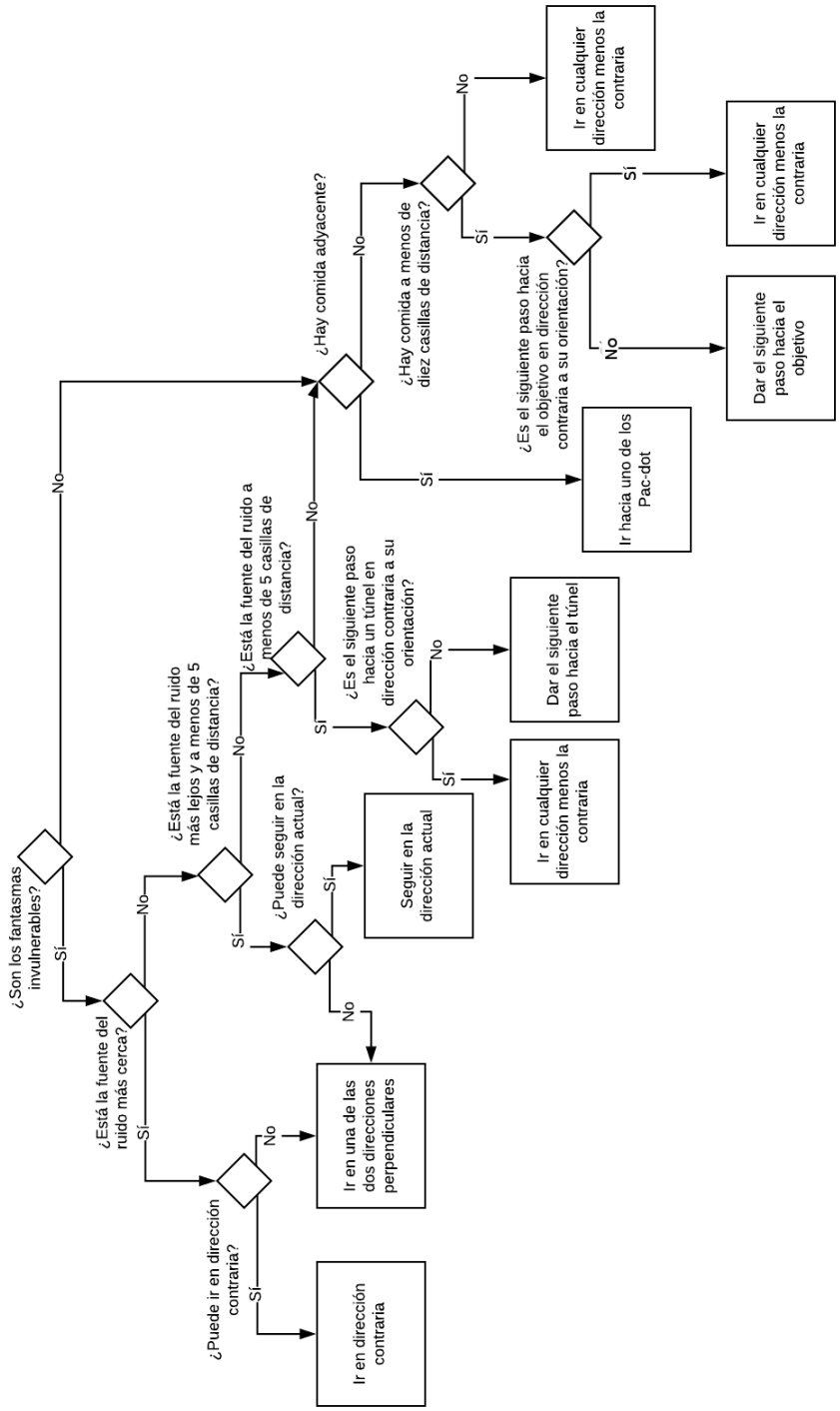

Figura 3.13: Árbol de decisión de la versión 3.

en el apéndice A.12.

El árbol de decisión correspondiente se puede ver en la figura 3.14. A simple vista se puede apreciar que esta versión es más simple en cuanto a posibles acciones que puede tomar *Pac-Man* y más restrictivo cuando se trata de huir de los fantasmas.

Tiene dos diferencias en cuanto a la versión anterior. La primera es que se ha eliminado la acción que hacia que el agente siguiera alejándose de los fantasmas cuando detectaba la fuente del sonido más lejana que en la casilla anterior. Esta opción solo servía para que *Pac-Man* continuara alejándose del ruido hasta no detectar nada, algo no muy necesario si no hay fantasmas persiguiéndole. El segundo cambio es que en esta versión *Pac-Man* solo dará media vuelta si detecta que está cada vez más cerca de la fuente del ruido y esta está a menos de cinco casillas de distancia. Esto le da al agente la capacidad de acercarse más a los *Pac-dot* que puede haber cerca de los fantasmas.

3.9. Versión 5

3.9.1. Descripción de la versión

En ocasiones, cuando *Pac-Man* accede a un teletransportador mientras le sigue un fantasma, puede salir por el otro teletransportador y encontrarse con el segundo fantasma, decidiendo darse la vuelta y colisionando contra el fantasma que le seguía en un principio. En algunos casos, el fantasma con el que se encuentra *Pac-Man* después del teletransporte no está lo suficientemente cerca para atraparlo, por lo que conviene evitar que se gire y choqué contra el que le persigue. Esto se ha solucionado obligando al agente a continuar hacia delante hasta salir del túnel después de teleportarse.

3.9.2. Detalles de la implementación

Para esta versión se ha modificado el fichero *level* y se ha incluido un nuevo fichero con la nueva clase *pacman*.

level.py

El método *CheckIfHitSomething* es la que se encarga de teletransportar a *Pac-Man* cuando este entra en uno de los túneles. Se ha modificado este procedimiento para que active uno

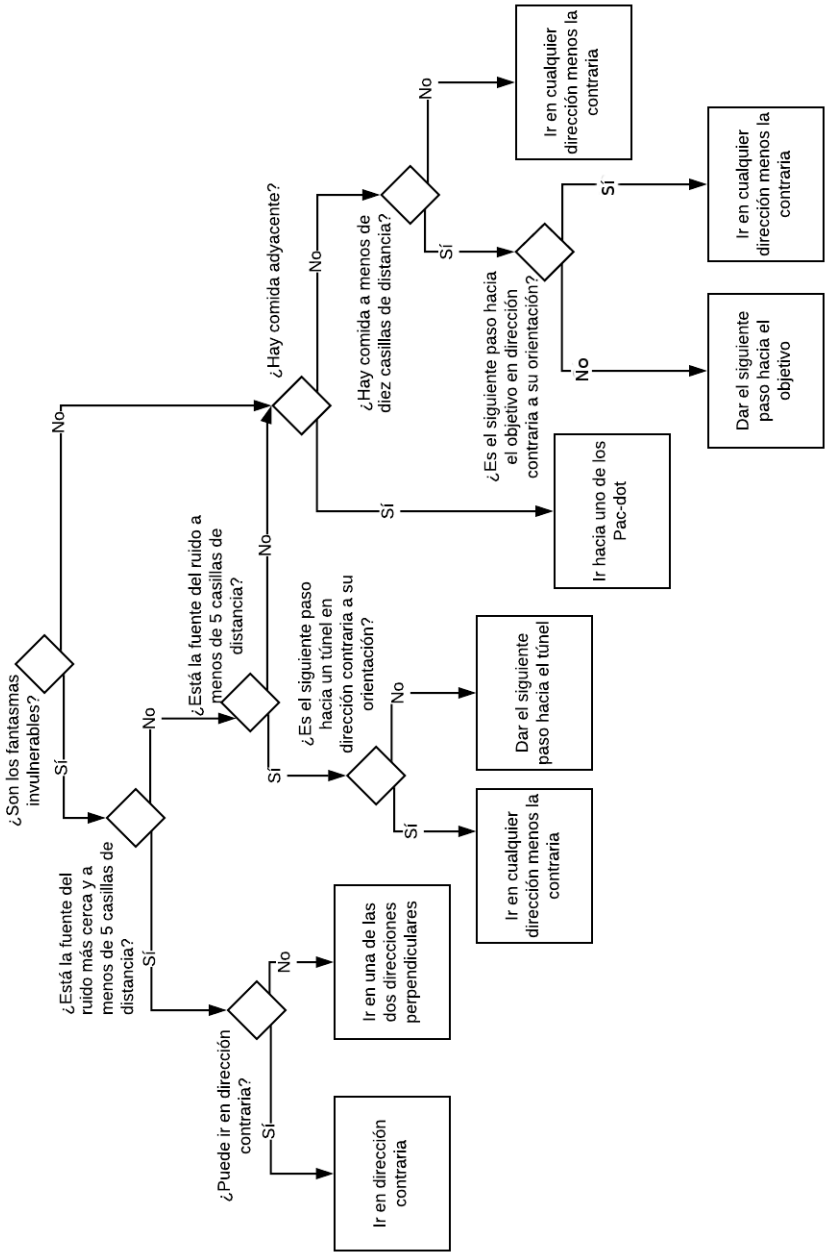

Figura 3.14: Árbol de decisión de la versión 4.

de los dos atributos de la clase *pacman* que indican si este ha cruzado un teleportador horizontal o vertical.

pacmanV5.py

La nueva clase se basa en la detallada en el apartado pacmanV4.py y, por tanto, el agente posee todas las capacidades de la versión anterior. Además de esto, se ha modificado el método *DecideSpeed* para que tenga en cuenta si *Pac-Man* ha pasado recientemente por un teleportador a la hora de tomar decisiones. En el apéndice A.13 se puede observar su pseudocódigo y en la figura 3.15 su árbol de decisión.

Se ha añadido una nueva condición que comprueba si *Pac-Man* se ha teletransportado recientemente. Si ese es el caso y el agente sigue en el túnel del teleportador, tendrá que continuar adelante. Una vez haya salido del túnel, *Pac-Man* procederá en una dirección aleatoria, excepto la dirección que le haría volver al teleportador. Se ha tomado la decisión de ir en una dirección aleatoria para evitar que el agente tome la decisión de volver al túnel, lo cual no le beneficiaría de ninguna forma.

3.10. Versión 6

3.10.1. Descripción de la versión

Esta versión es la primera del segundo modo de juego y, por tanto, la primera pensada para funcionar con dos agentes. El comportamiento de los agentes es exactamente igual al de la versión anterior.

3.10.2. Detalles de la implementación

Para esta versión se ha incluido un nuevo fichero con la nueva clase *pacman*.

pacmanV6.py

Esta versión pretende tener el mismo comportamiento que la versión anterior, por lo que la clase es casi idéntica a la clase del apartado pacmanV5.py. Solo se han hecho algunas

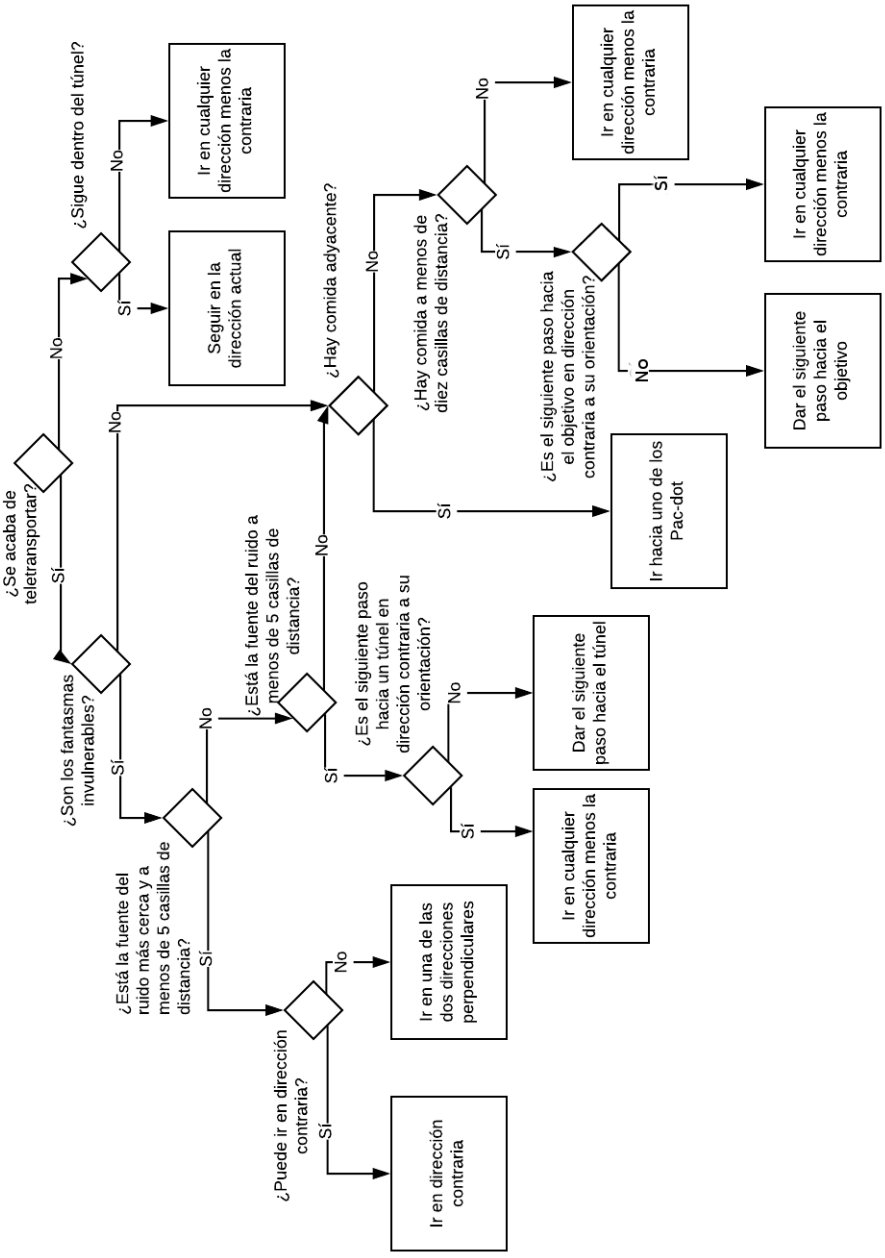

Figura 3.15: Árbol de decisión de la versión 5.

modificaciones a la clase para que se puedan crear varios agentes. Además, se ha modificado un atributo de la clase para que cada agente inicie el juego en una orientación distinta. Por último, se ha añadido un atributo que indica si el agente está vivo o muerto. Este atributo se usa en los métodos *Move*, *Odor*, *Visible* y *DecideSpeed* para saber si hace falta ejecutarlos o no.

Al ser el comportamiento el mismo de la versión anterior, el árbol de decisión también es igual al de la figura 3.15, aunque solo se recorrerá el árbol si el agente está vivo.

3.11. Versión 7

3.11.1. Descripción de la versión

En esta versión se ha intentado en cierta forma que los dos agentes se dividan el trabajo de comer los *Pac-dot*, con el objetivo de ahorrar tiempo. Para ello, se le ha asignado la parte superior del laberinto a un *Pac-Man* y la parte inferior al otro.

3.11.2. Detalles de la implementación

Para esta versión se ha incluido un nuevo fichero con la nueva clase *pacman*.

pacmanV7.py

El fichero añadido se basa en el detallado en la sección pacmanV6.py, con un ligero cambio en el procedimiento *DecideSpeed*, que se puede observar en el apéndice A.14. En la figura 3.16 se puede ver la representación del procedimiento mediante un árbol de decisión.

La única diferencia de este procedimiento respecto a su homónimo de la versión anterior se da cuando no hay ningún *Pac-dot* a la distancia requerida. En este caso, se comprueba que el agente este en su zona asignada (superior o inferior); si está en su zona, irá en una dirección aleatoria, si no, intentará ir en la dirección que le lleve a la misma.

Aunque teóricamente este comportamiento debería mejorar como mínimo el tiempo necesario para completar el nivel, en la práctica esto no tiene por qué cumplirse, debido al uso que hacen los agentes de los teleportadores. Estos mecanismos provocan que los agentes

se encuentren más veces de lo debido fuera de su zona asignada, obligándoles a recorrer todo el laberinto para llegar a la misma.

3.12. Versión 8

3.12.1. Descripción de la versión

En esta versión se ha dotado a los agentes con la capacidad de resucitarse el uno al otro. Es decir, cuando un *Pac-Man* vivo colisione con uno muerto, este último volverá a la vida. Esta versión de los agentes no tiene cambios en la inteligencia, por tanto, un agente solo resucitará al otro si se encuentran de manera fortuita.

3.12.2. Detalles de la implementación

Para esta versión se ha modificado la clase *level* y se ha añadido un fichero con la nueva clase *pacman*.

level.py

Se ha añadido un nuevo método llamado *RevivePacman* en este fichero. Su pseudocódigo está disponible en el apéndice A.15. Este procedimiento se ejecuta en cada iteración del programa y comprueba si un *Pac-Man* vivo ha colisionado con uno muerto. Si se da este caso, el agente muerto pasará a estar vivo y el contador que lleva la cuenta de los *Pac-Man* que están vivos aumentará en uno.

Para evitar que el agente recién resucitado usé información desactualizada para decidir su próximo movimiento, se le asigna como dirección a seguir la orientación actual del agente que lo ha rescatado.

pacmanV8.py

Esta versión no requiere ningún cambio en la inteligencia de los agentes, por lo que se ha tomado la decisión de copiar la clase *pacman* de una versión anterior. Más concretamente, se ha copiado el fichero de la versión 6 (pacmanV6.py). Se ha usado esta versión y no la versión 7 debido a que el comportamiento que hace que cada agente se centre en una zona

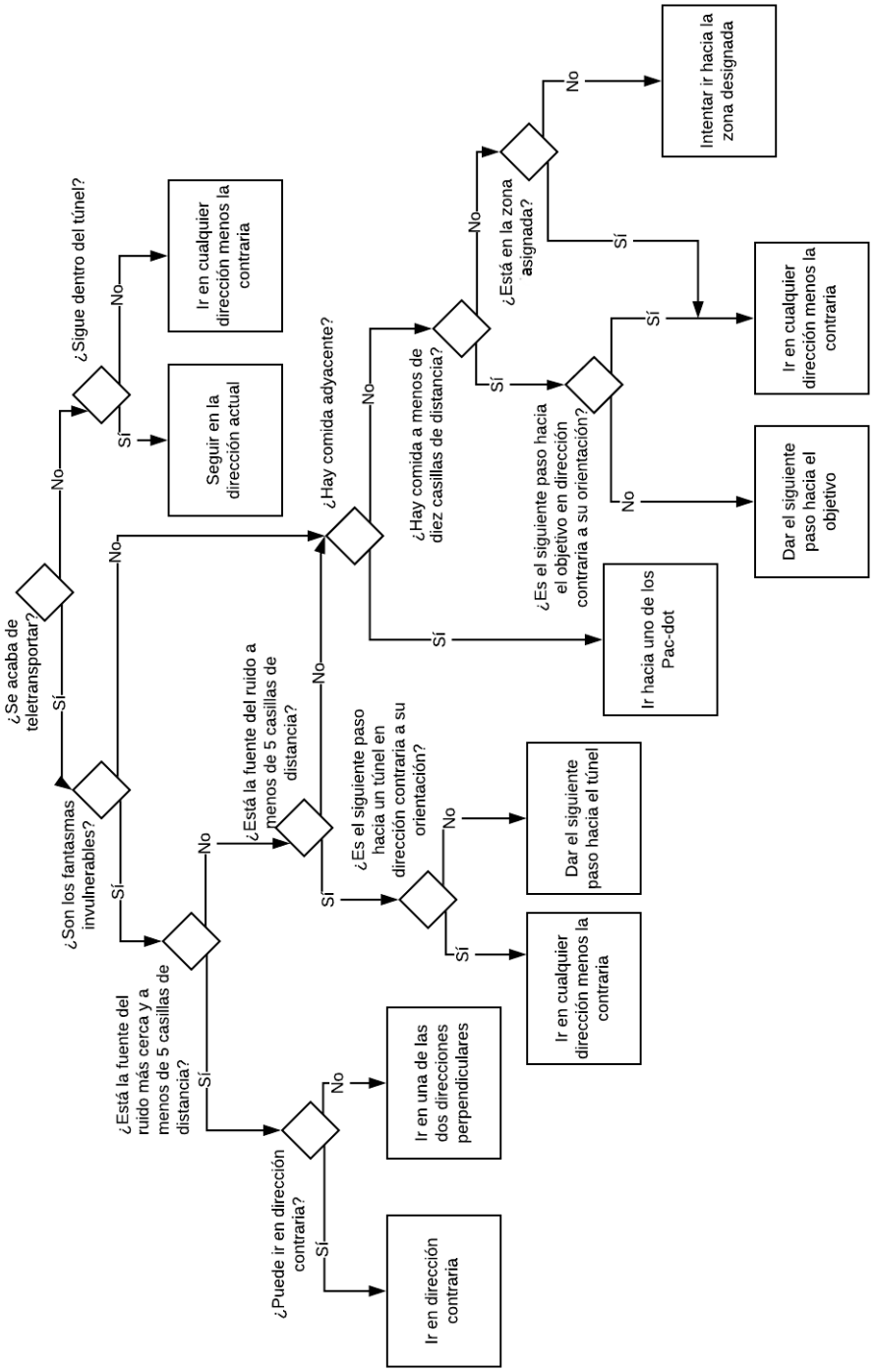

Figura 3.16: Árbol de decisión de la versión 7.

del laberinto no les beneficia a la hora de salvarse entre ellos, puesto que si un agente muere en su zona asignada, el otro no podría salvarlo a no ser que saliera de su zona.

3.13. Versión 9

3.13.1. Descripción de la versión

En esta versión los agentes se buscan de manera activa para intentar resucitarse el uno al otro. Un agente tratará de resucitar a otro siempre y cuando este último este muerto y ningún fantasma le impida acercarse a él. Los agentes le darán prioridad al rescate respecto a los *Pac-dot*.

3.13.2. Detalles de la implementación

Para esta versión se ha modificado el fichero *level.py* y se ha añadido un fichero con la nueva clase *pacman*.

level.py

Se han añadido tres nuevos métodos a la clase *level*. El primero de ellos se llama *GetNeighbors* y su pseudocódigo se encuentra en el apéndice A.16. Es una función auxiliar que, habiendo recibido una casilla del laberinto, devuelve en una lista las coordenadas de sus casillas adyacentes que no sean paredes.

El segundo método, *TranslatePath*, se puede ver en el apéndice A.17. Esta función recibe una lista de casillas que forman un camino a través del laberinto y devuelve una lista con las direcciones para seguir ese camino.

El último método añadido y el más importante es *AStarSearch*, cuyo pseudocódigo se puede ver en el apéndice A.18. Como su nombre indica, esta función lleva a cabo una búsqueda A^* [Cui and Shi, 2011a] para encontrar el camino más corto entre las casillas del laberinto indicadas. Para ello va recorriendo el laberinto ayudándose de la función auxiliar *GetNeighbors* y una cola de prioridades ordenada mediante el coste. El coste se calcula sumando el coste hasta el momento, que es equivalente al número de casillas recorridas, y el coste que se prevé que habrá desde una casilla hasta la casilla objetivo. Esta previsión se ha calculado usando la distancia Manhattan.

Una vez la función llega a la casilla objetivo, hay que desandar el camino para ir guardando en orden inverso las casillas por las que tendría que pasar el agente para realizar el camino. Para terminar, se usa la función *TranslatePath* para traducir esta sucesión de casillas a una lista de direcciones que los agentes puedan interpretar sin problemas.

pacmanV9.py

En esta versión los agentes necesitan por primera vez tener información el uno del otro. Para que uno de los *Pac-Man* intenté resucitar al otro necesita saber antes de nada si este último está muerto. Además, también necesita conocer la posición de su compañero, para así llegar hasta él. Estos dos datos se guardarán en la clase *pacman* como dos nuevos atributos y se actualizarán en cada iteración del programa tras moverse los agentes.

Por otra parte, se ha modificado la función *DecideSpeed* para añadir el nuevo mecanismo de rescate. El nuevo pseudocódigo se puede consultar en el apéndice A.19 y su representación mediante un árbol de decisión se ilustra en la figura 3.17.

Como se puede ver, si el agente no necesita escapar de ningún fantasma, comprueba si su compañero está muerto. Si ese es el caso, el procedimiento verifica que *Pac-Man* tenga el camino hasta su compañero calculado y si no lo tiene lo calcula llamando a la función *AStarSearch*, especificando como casilla de salida la posición actual de *Pac-Man* y como casilla objetivo la posición del compañero. Una vez calculado el camino, se guarda en un atributo de la instancia y se elige como próximo movimiento del agente el primero de esta lista, a no ser que sea en una dirección contraria a su orientación, en cuyo caso el agente se dirigirá en una dirección aleatoria que no sea la contraria.

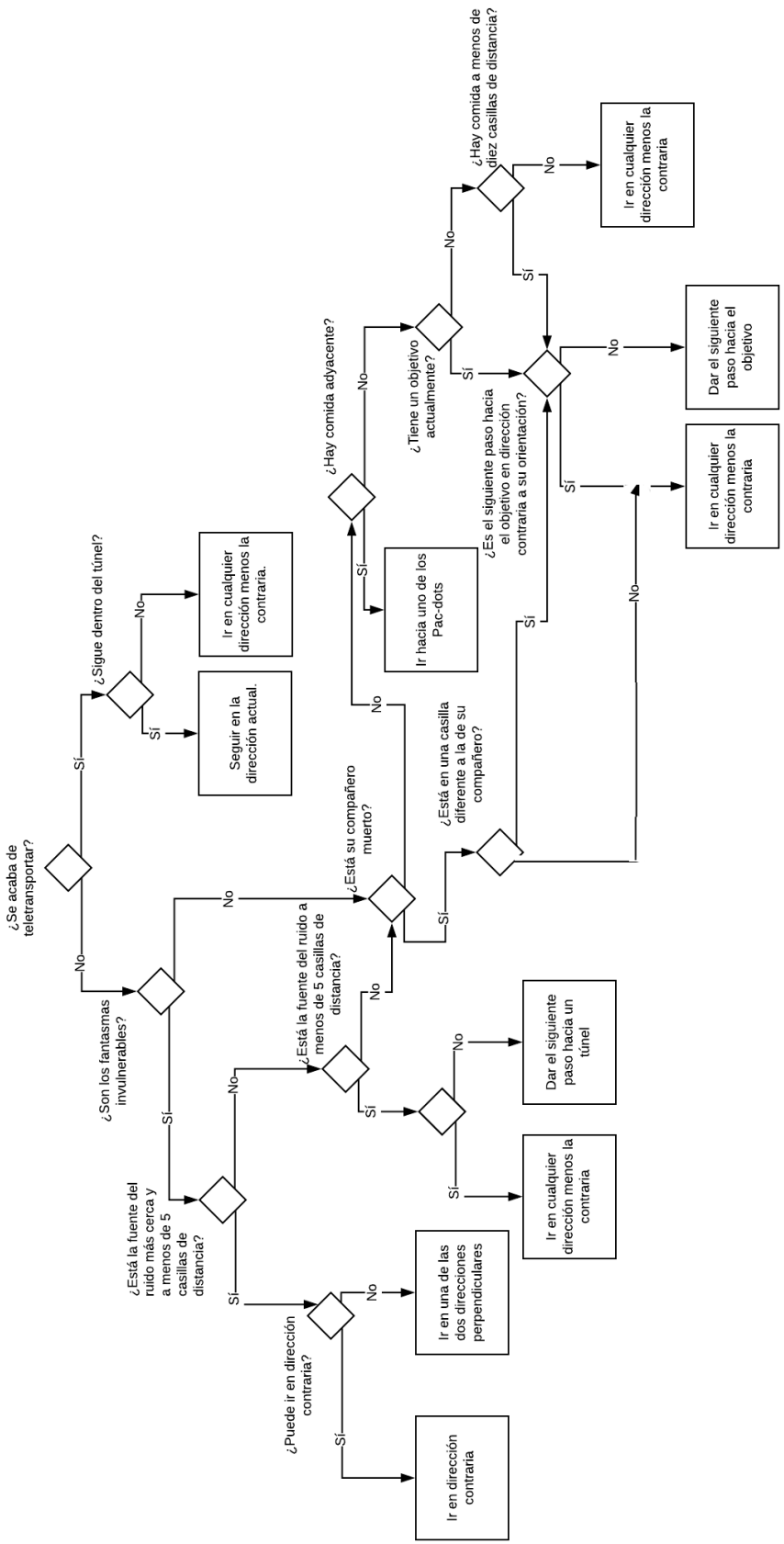

Figura 3.17: Árbol de decisión de la versión 9.

3.14. Versión 10

3.14.1. Descripción de la versión

Esta versión es similar a la anterior en que los agentes también tratan de resucitarse entre ellos. La diferencia es que esta vez los agentes le dan prioridad al objetivo del juego de comer los *Pac-dot* por encima de rescatar a su compañero.

3.14.2. Detalles de la implementación

Para esta versión se ha añadido un fichero con la nueva clase *pacman*.

pacmanV10.py

Este nuevo fichero es similar al de la versión anterior, pero con las prioridades alternadas en la función *DecideSpeed*. En el apéndice A.19 se puede ver el pseudocódigo modificado y en la figura 3.18 el árbol de decisión que lo representa.

Como se puede apreciar, el proceso es parecido al de la versión anterior, solo que esta vez se lleva a cabo después de fracasar buscando comida alrededor del agente. Es decir, un agente solo irá a rescatar a su compañero si en ese momento no puede avanzar hacia la compleción del juego.

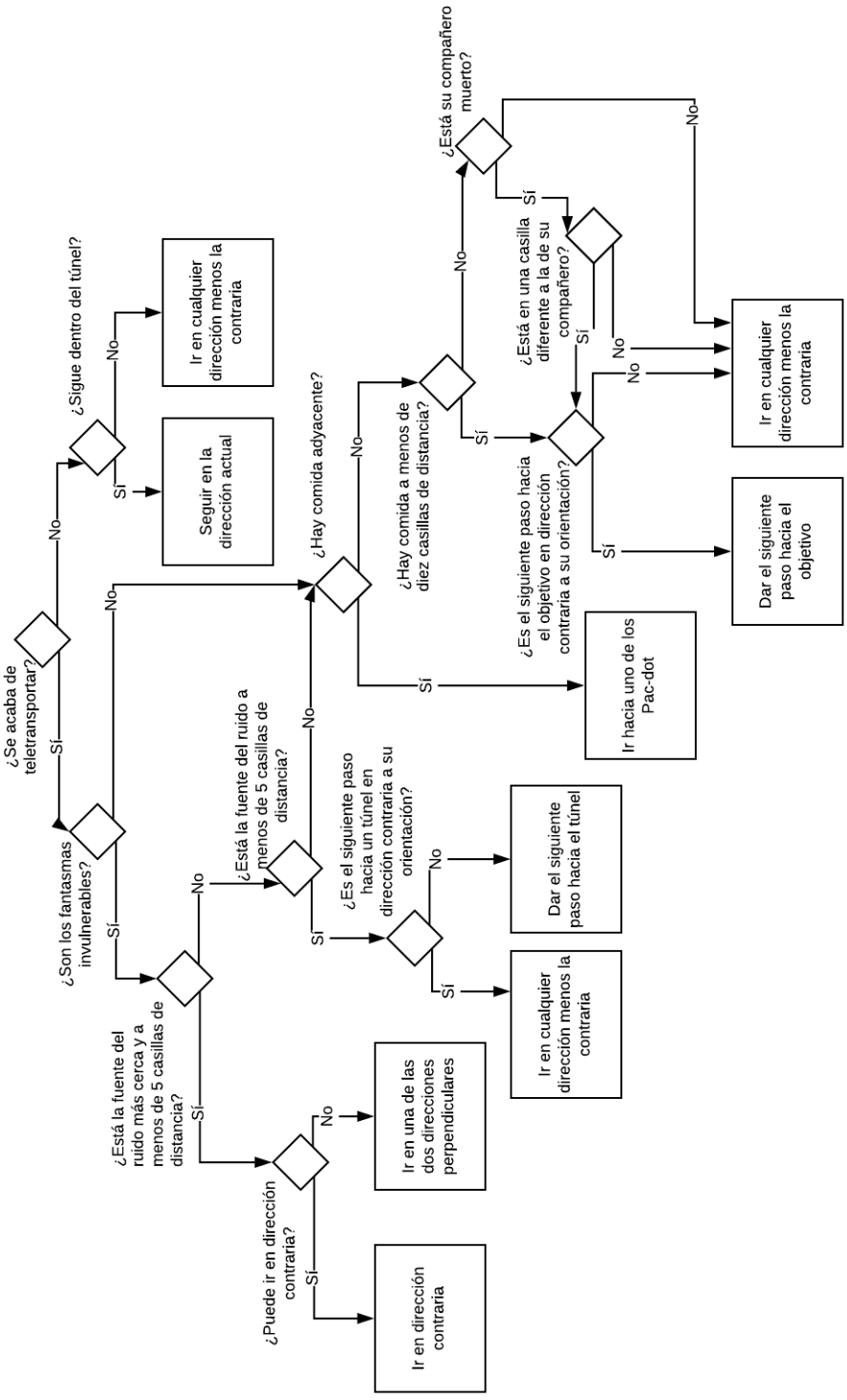

Figura 3.18: Árbol de decisión de la versión 10.

4. CAPÍTULO

Resultados y conclusiones

En este capítulo se muestran los resultados de las pruebas realizadas para evaluar la eficacia de cada una de las versiones de IA creadas. Se ha enfrentado cada versión a cada una de las inteligencias rivales del mismo modo de juego diez veces por nivel. Por tanto, hay veinte pruebas por cada combinación de una versión de *Pac-Man* con una de los fantasmas, siempre y cuando pertenezcan al mismo modo de juego.

La evaluación de la eficacia se mide mediante el número de vidas que pierde *Pac-Man* para completar el nivel y el tiempo transcurrido para ello. En algunos casos, todos los agentes se coordinan de tal forma que la partida continúa indefinidamente; esto ocurre cuando un fantasma protege la comida en una zona con pocos accesos, *Pac-Man* no se atreve a acercarse y el fantasma no consigue verle. En estos casos, se ha registrado el número de vidas perdidas hasta el momento y se ha fijado el tiempo como ∞.

En lo que respecta al tiempo, ejecutando el programa de forma normal algunas pruebas llegan a durar varios minutos, por lo que se ha acelerado este proceso eliminando el límite de ejecuciones por segundo del código para conseguir los siguientes resultados. Las pruebas de cada nivel se han realizado en ordenadores con capacidades de computo diferentes[1], por lo que los resultados solo serán comparables siempre y cuando pertenezcan mismo nivel.

[1] Nivel 1: procesador Intel(R) Core(TM) i5-4460 CPU @ 3.20GHz 3.20GHz y 8 GB de RAM. Nivel 2: procesador Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz 3.20GHz y 16 GB de RAM.

4.1. Versión 0

4.1.1. Resultados de las pruebas

Los resultados de las pruebas en el primer nivel se pueden ver en la tabla 4.1. En cuanto al segundo nivel, sus resultados se pueden observar en la tabla 4.2.

Tabla 4.1: Resultados de la versión 0 en el primer nivel, donde t es el tiempo en segundos y v el número de vidas perdidas.

	n.º de prueba																			
	1		**2**		**3**		**4**		**5**		**6**		**7**		**8**		**9**		**10**	
	t	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**
V0	105	3	111	5	96	6	104	5	101	7	95	5	103	1	86	2	91	2	97	5
V1	167	12	154	12	123	8	227	20	127	3	90	9	161	19	157	12	86	11	131	10
V2	235	13	276	11	333	24	111	2	232	11	132	4	263	12	237	10	279	15	109	9
V3	351	18	161	8	252	14	180	5	302	16	176	16	137	3	183	17	267	16	243	12
V4	351	18	161	8	252	14	180	5	302	16	176	16	137	3	183	17	267	16	243	12
V5	264	9	196	7	343	13	190	11	301	6	134	6	266	16	225	6	343	6	314	9
V6.1	165	9	320	24	252	10	139	4	89	4	352	11	285	12	152	8	222	7	289	7
V6.2	193	12	183	7	182	13	∞	4	398	19	292	12	199	14	316	21	∞	16	239	17
V7	221	13	134	7	140	6	170	6	209	17	369	8	161	6	265	7	175	9	189	9
V8	78	8	165	12	146	13	112	8	80	5	87	6	185	21	113	10	73	3	66	5
V9	182	13	160	14	298	17	123	7	97	2	204	7	241	20	185	13	101	9	169	11

Tabla 4.2: Resultados de la versión 0 en el segundo nivel, donde t es el tiempo en segundos y v el número de vidas perdidas.

	n.º de prueba																			
	1		**2**		**3**		**4**		**5**		**6**		**7**		**8**		**9**		**10**	
	t	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**
V0	54	5	101	4	79	1	85	1	87	4	68	4	92	3	86	5	66	3	84	5
V1	69	10	90	9	69	6	72	4	83	6	98	8	66	5	72	6	68	4	89	7
V2	287	21	91	2	135	11	256	22	98	4	309	17	144	14	122	11	283	23	136	7
V3	∞	8	∞	2	∞	0	∞	4	∞	0	∞	0	∞	0	367	19	∞	2	∞	1
V4	∞	8	∞	2	∞	0	∞	4	∞	0	∞	0	∞	0	367	19	∞	2	∞	1
V5	103	2	174	4	∞	0	∞	1	∞	0	∞	0	∞	2	∞	1	∞	0	73	7
V6.1	∞	0	∞	0	111	1	275	13	∞	1	125	4	∞	0	∞	2	∞	2	248	10
V6.2	∞	4	∞	3	∞	1	∞	0	∞	0	100	8	∞	1	∞	0	∞	2	∞	3
V7	∞	2	∞	5	∞	0	∞	0	∞	1	∞	0	∞	7	∞	1	177	9	∞	1
V8	69	10	70	8	78	7	119	14	61	2	82	8	64	6	81	9	90	11	99	10
V9	∞	2	118	17	∞	1	170	20	76	10	∞	4	226	10	110	18	∞	10	59	7

La figura 4.1 ilustra la eficacia de esta versión de *Pac-Man* en el primer nivel contra cada

una de las versiones de los fantasmas de este modo de juego. En la figura 4.2 se muestra lo mismo para el segundo nivel. Debido a la heterogeneidad de los datos, para hacer estos gráficos se han usado las medianas de las diez pruebas.

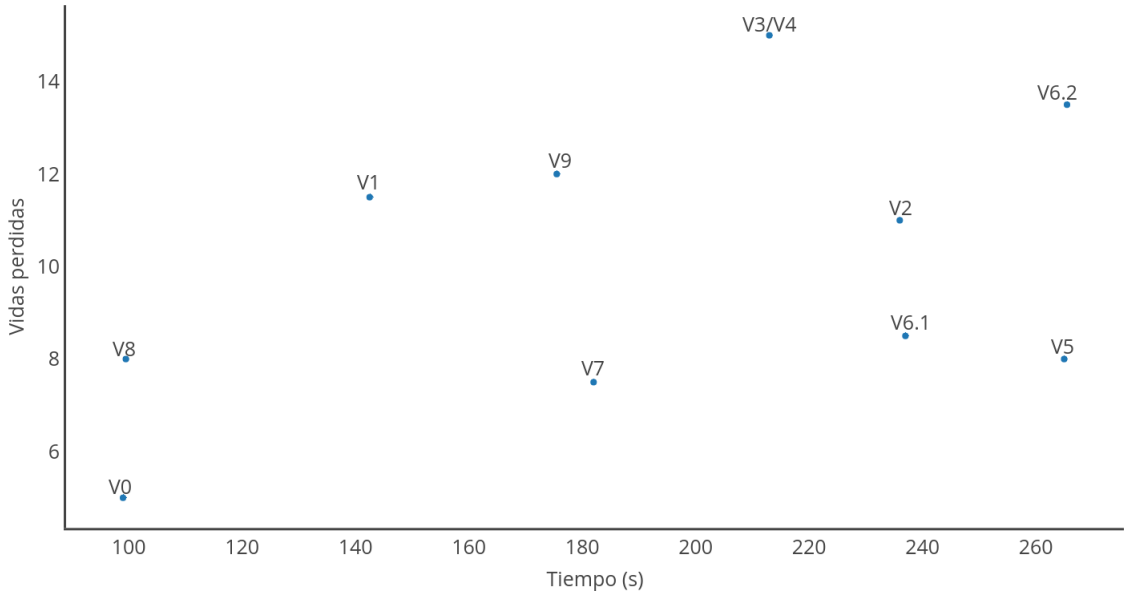

Figura 4.1: Representación gráfica de la eficacia de la versión 0 contra las diferentes versiones de los fantasmas en el primer nivel.

4.1.2. Conclusiones

Como se puede apreciar en las tablas, esta primera versión del agente no es nada constante, debido a su aleatoriedad y la falta de capacidad de localizar los *Pac-dot*. En consecuencia, *Pac-Man* se puede quedar bloqueado en diferentes zonas del laberinto, resultando en una partida incompleta. Esto ocurre especialmente en el segundo nivel, motivo por el que no se han podido representar de manera gráfica muchas de las eficacias en este nivel.

En las figuras se puede ver que esta versión funciona mejor contra las primeras versiones de los fantasmas, hasta que estos consiguen la capacidad de proteger los *Pac-dot*. Al no tener *Pac-Man* la capacidad de localizar la comida, la protección de los fantasmas a menudo tiene como consecuencia que el agente no termine el nivel. También se puede observar que la eficacia del agente aumenta contra las dos últimas versiones, consiguiendo contra la versión 8 uno de los mejores resultados en ambos niveles.

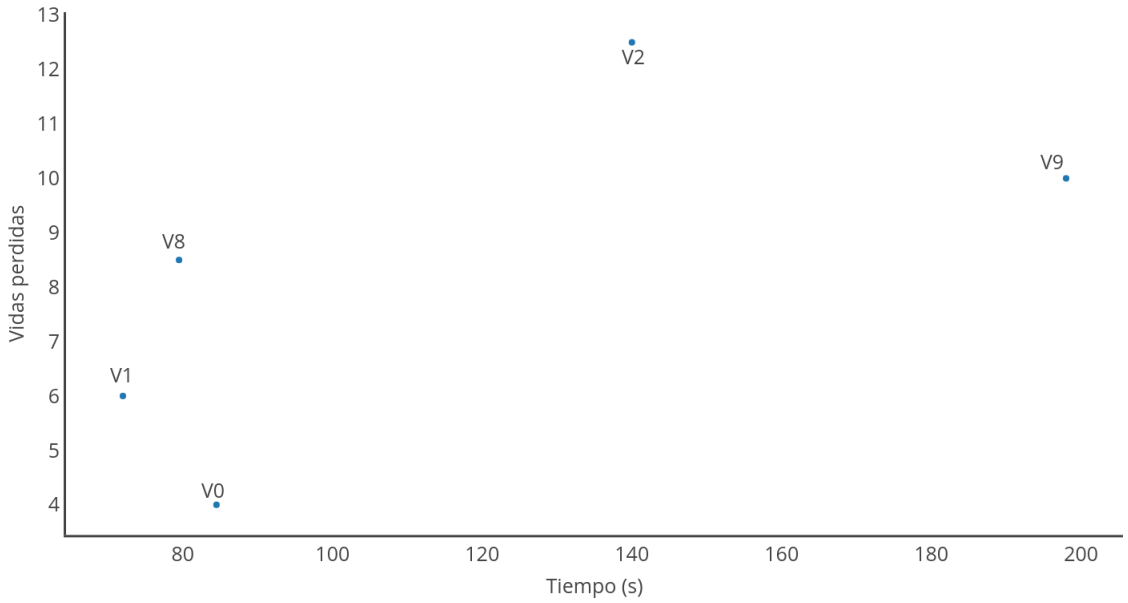

Figura 4.2: Representación gráfica de la eficacia de la versión 0 contra las diferentes versiones de los fantasmas en el segundo nivel.

4.2. Versión 1

4.2.1. Resultados de las pruebas

Los resultados de esta versión se pueden ver en las tablas 4.3, y 4.4.

Las representaciones gráficas se pueden ver en las figuras 4.3 y 4.4.

4.2.2. Conclusiones

Se puede ver en las tablas que tanto los resultados del tiempo como los de las vidas perdidas mejoran en esta versión. Esto ocurre debido a que la versión 1 del agente sabe que su objetivo es comer los *Pac-dot* repartidos por el laberinto y puede localizarlos a cierta distancia. Del mismo modo que la versión anterior, consigue buenos resultados contra las primeras tres versiones de los fantasmas, como se puede ver en los gráficos. La eficacia también es buena contra algunas otra versiones del fantasma que centran su atención en los teletransportadores, puesto que esta versión no hace uso de estos mecanismos demasiado a menudo.

Tabla 4.3: Resultados de la versión 1 en el primer nivel, donde t es el tiempo en segundos y v el número de vidas perdidas.

	n.º de prueba																			
	1		**2**		**3**		**4**		**5**		**6**		**7**		**8**		**9**		**10**	
	t	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**
V0	31	1	26	1	24	1	34	1	28	2	25	1	35	3	35	3	35	3	23	1
V1	29	3	32	2	28	2	26	1	30	3	32	1	25	1	46	4	27	3	29	3
V2	53	7	34	4	35	4	34	0	28	2	28	0	47	2	46	3	33	2	38	3
V3	36	2	53	4	51	7	43	2	41	3	56	7	49	4	88	11	31	3	83	4
V4	36	2	53	4	51	7	43	2	41	3	56	7	49	4	88	11	31	3	83	4
V5	80	0	39	1	35	3	33	3	30	2	50	1	28	1	29	3	28	1	48	1
V6.1	62	4	58	6	80	1	42	3	48	2	33	1	26	1	23	0	26	0	29	1
V6.2	36	4	52	10	39	3	94	7	∞	5	43	2	40	3	42	4	39	4	57	4
V7	52	3	27	3	54	2	22	0	55	5	29	3	103	5	38	2	54	6	42	1
V8	26	2	22	2	28	2	46	3	30	3	27	3	25	1	28	2	29	2	46	5
V9	39	4	71	6	101	9	43	6	48	6	43	4	32	2	43	3	26	2	35	5

Tabla 4.4: Resultados de la versión 1 en el segundo nivel, donde t es el tiempo en segundos y v el número de vidas perdidas.

	n.º de prueba																			
	1		**2**		**3**		**4**		**5**		**6**		**7**		**8**		**9**		**10**	
	t	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**
V0	19	0	16	0	27	1	25	3	21	0	20	1	25	2	33	1	29	1	32	2
V1	27	4	28	2	21	1	29	2	24	3	45	6	27	3	23	0	29	3	28	4
V2	123	7	95	6	43	1	35	1	33	2	27	1	33	1	22	1	85	4	79	4
V3	41	1	36	12	∞	2	80	2	∞	0	∞	5	206	4	39	1	∞	0	33	6
V4	41	1	36	12	∞	2	80	2	∞	0	∞	5	206	4	39	1	∞	0	33	6
V5	∞	1	22	1	22	0	147	3	38	4	36	0	32	1	34	1	21	0	∞	0
V6.1	20	0	∞	2	∞	0	32	5	50	0	∞	0	19	0	25	1	∞	0	∞	0
V6.2	85	7	25	7	59	3	∞	0	34	6	∞	0	∞	0	61	8	63	4	∞	0
V7	24	4	38	4	∞	0	30	7	50	4	28	0	29	2	∞	0	55	6	60	3
V8	21	3	19	2	25	2	21	3	27	3	21	3	33	5	39	5	21	3	21	3
V9	26	3	128	4	∞	0	∞	3	31	3	27	4	∞	0	48	10	18	0	∞	3

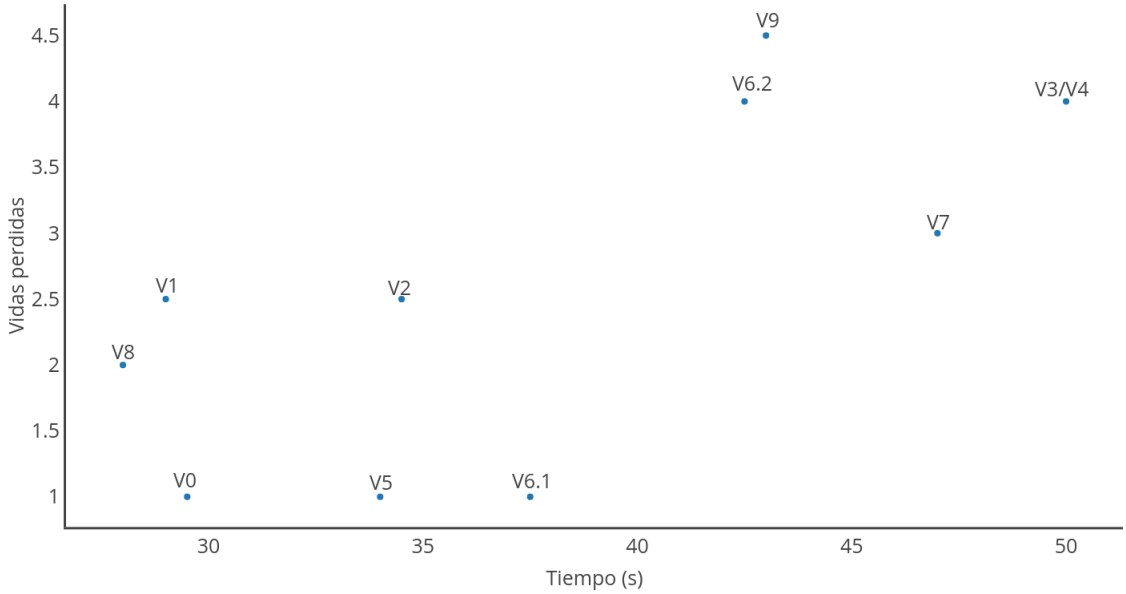

Figura 4.3: Representación gráfica de la eficacia de la versión 1 contra las diferentes versiones de los fantasmas en el primer nivel.

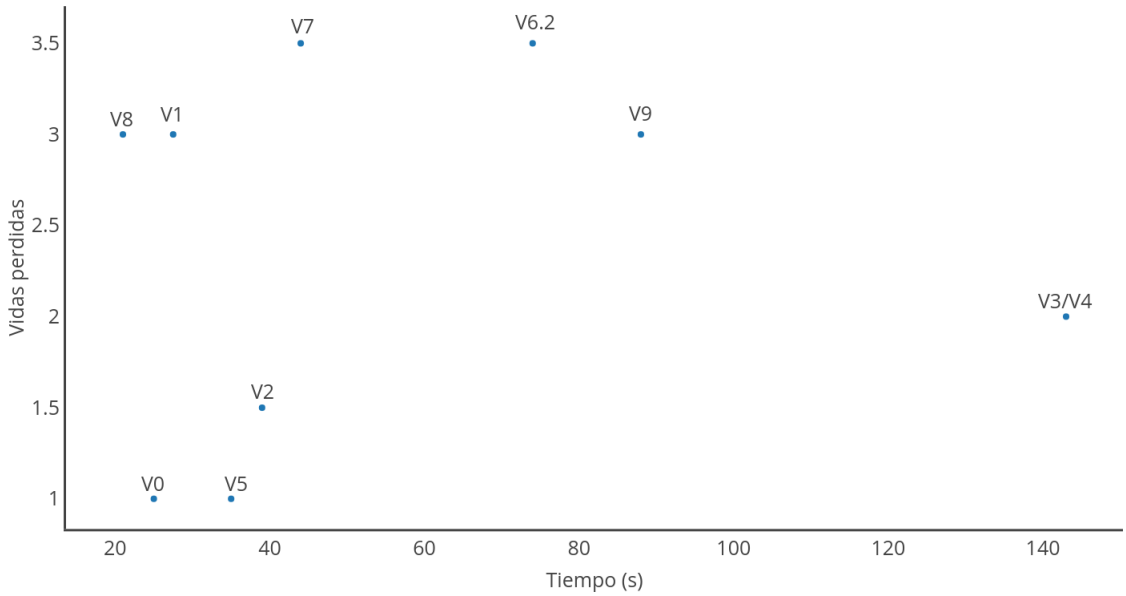

Figura 4.4: Representación gráfica de la eficacia de la versión 1 contra las diferentes versiones de los fantasmas en el segundo nivel.

4.3. Versión 2

4.3.1. Resultados de las pruebas

Los resultados de la versión 2 en el primer nivel se pueden ver en la tabla 4.5; los del segundo nivel se muestran en la tabla 4.6.

Tabla 4.5: Resultados de la versión 2 en el primer nivel, donde t es el tiempo en segundos y v el número de vidas perdidas.

	n.º de prueba																			
	1		**2**		**3**		**4**		**5**		**6**		**7**		**8**		**9**		**10**	
	t	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**
V0	29	2	32	0	32	0	32	0	32	0	56	5	36	0	39	4	43	2	41	1
V1	34	0	52	2	41	2	35	2	39	2	39	1	31	1	28	0	75	7	25	1
V2	29	1	78	5	36	1	32	1	36	0	56	3	57	4	54	4	43	4	65	1
V3	40	2	42	5	44	4	68	2	96	6	83	1	68	3	85	2	121	2	34	1
V4	40	2	42	5	44	4	68	2	96	6	83	1	68	3	85	2	121	2	34	1
V5	60	5	∞	2	44	2	40	4	49	5	56	5	35	2	41	1	91	5	57	3
V6.1	66	2	∞	0	107	3	126	3	72	6	49	1	31	1	46	4	40	3	41	4
V6.2	77	1	133	6	46	3	51	2	36	5	35	3	70	4	77	5	52	2	64	0
V7	45	4	51	3	44	4	67	5	31	2	60	5	106	7	67	2	52	4	100	2
V8	50	6	28	3	34	2	34	3	34	4	55	3	73	6	27	2	36	4	28	4
V9	50	4	36	2	52	5	64	5	∞	0	47	7	45	7	36	2	112	7	51	2

Tabla 4.6: Resultados de la versión 2 en el segundo nivel, donde t es el tiempo en segundos y v el número de vidas perdidas.

	n.º de prueba																			
	1		**2**		**3**		**4**		**5**		**6**		**7**		**8**		**9**		**10**	
	t	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**
V0	22	0	25	1	36	3	24	1	24	1	24	0	19	0	24	3	26	2	28	2
V1	27	2	31	2	20	0	27	2	21	0	25	0	27	0	29	0	22	1	35	1
V2	44	4	∞	0	62	3	33	3	62	1	67	2	65	4	72	4	56	0	63	4
V3	∞	0	∞	0	35	0	∞	0	50	1	90	3	∞	0	∞	0	∞	1	∞	1
V4	∞	0	∞	0	35	0	∞	0	50	1	90	3	∞	0	∞	0	∞	1	∞	1
V5	∞	1	∞	0	∞	0	∞	0	∞	1	67	2	39	1	∞	1	39	0	∞	1
V6.1	∞	0	51	7	84	2	32	4	∞	0	∞	1	93	6	∞	0	237	12	83	4
V6.2	∞	3	122	4	41	1	68	1	∞	4	122	5	∞	0	28	1	∞	0	67	0
V7	∞	1	118	3	160	3	59	2	61	1	74	4	37	2	83	1	∞	0	47	2
V8	21	2	26	2	33	3	26	2	35	4	21	2	52	3	27	5	20	1	46	4
V9	47	4	48	2	68	9	∞	0	84	7	∞	5	∞	5	49	7	∞	1	128	2

Se puede ver la eficacia de la versión 2 de manera gráfica en las figuras 4.5 y 4.6.

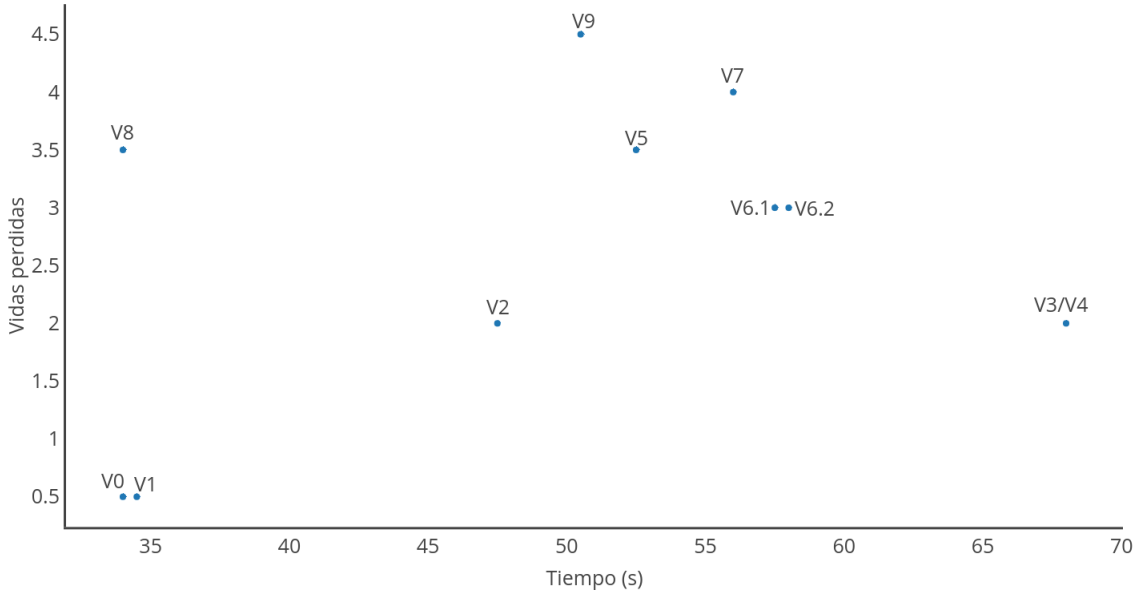

Figura 4.5: Representación gráfica de la eficacia de la versión 2 contra las diferentes versiones de los fantasmas en el primer nivel.

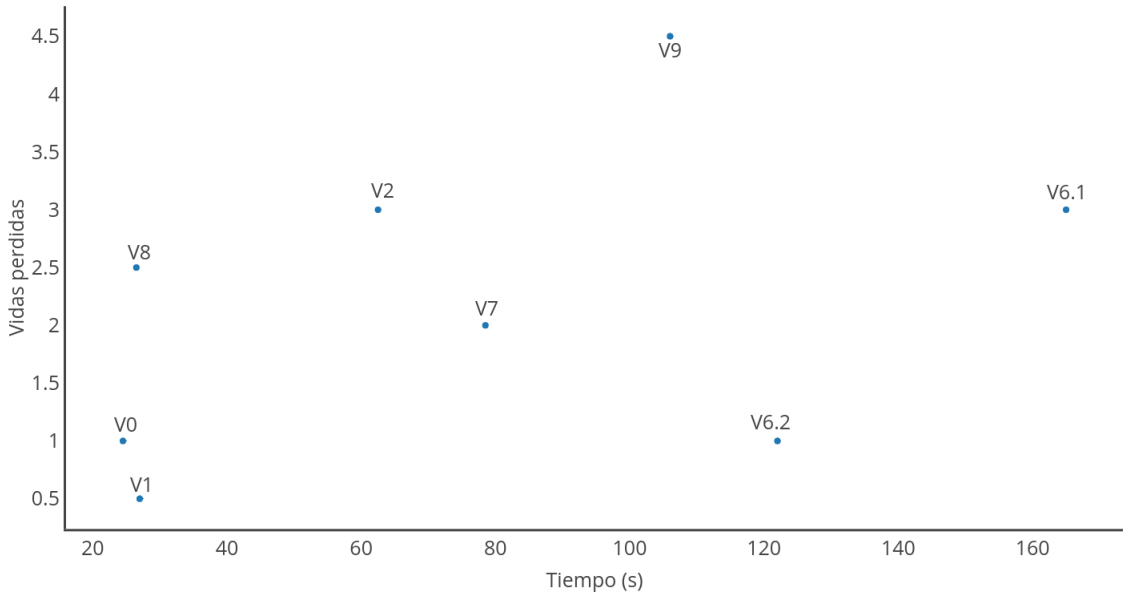

Figura 4.6: Representación gráfica de la eficacia de la versión 2 contra las diferentes versiones de los fantasmas en el segundo nivel.

4.3.2. Conclusiones

Esta versión sigue siendo eficaz contra las primeras versiones de los fantasmas, pero pierde eficacia contra las versiones que comienzan a bloquear teletransportes, puesto que este agente hace uso de estos mecanismos cada vez que le persigue un fantasma. Esto termina consumiendo demasiadas vidas a *Pac-Man*. Además, la versión 2 sigue sin terminar siempre el segundo nivel contra fantasmas que protegen mucho la comida.

4.4. Versión 3

4.4.1. Resultados de las pruebas

Las tablas 4.7 y 4.8 contienen los resultados de la versión 3 en el primer y segundo nivel respectivamente.

Tabla 4.7: Resultados de la versión 3 en el primer nivel, donde t es el tiempo en segundos y v el número de vidas perdidas.

	n.º de prueba																			
	1		**2**		**3**		**4**		**5**		**6**		**7**		**8**		**9**		**10**	
	t	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**
V0	29	3	27	0	57	2	24	0	41	2	32	0	54	2	34	1	37	1	24	0
V1	43	3	29	1	39	0	44	3	32	0	26	1	36	2	26	0	27	0	27	0
V2	28	1	47	1	78	4	63	2	35	2	39	2	66	4	113	9	44	4	65	2
V3	96	4	36	1	45	4	34	3	32	0	48	2	38	1	35	5	35	2	117	2
V4	64	7	36	2	39	0	35	0	48	2	52	3	33	1	32	4	44	0	48	2
V5	37	1	38	5	68	5	57	5	38	7	45	7	65	9	87	6	79	2	52	5
V6.1	43	2	33	1	34	1	39	5	84	3	45	4	50	3	81	3	38	5	44	7
V6.2	62	3	84	7	28	1	37	1	189	7	33	2	49	1	43	1	50	4	56	2
V7	61	4	64	4	52	2	69	1	46	2	60	3	49	2	40	2	88	2	30	3
V8	25	2	35	5	32	1	28	3	41	3	38	5	35	2	46	3	28	2	59	6
V9	135	6	59	3	41	4	76	6	38	3	55	2	32	1	34	2	46	4	58	3

Las figuras 4.7 y 4.8 ilustran la eficacia de la versión 3 en el primer y segundo nivel respectivamente.

Tabla 4.8: Resultados de la versión 3 en el primer nivel, donde t es el tiempo en segundos y v el número de vidas perdidas.

	n.º de prueba																			
	1		**2**		**3**		**4**		**5**		**6**		**7**		**8**		**9**		**10**	
	t	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**
V0	26	0	52	4	23	0	24	0	26	2	22	1	25	2	34	0	26	4	25	1
V1	21	0	30	3	23	1	28	1	31	0	29	1	27	2	23	2	22	1	25	3
V2	48	1	24	3	23	2	46	1	70	6	49	0	37	1	52	3	28	1	54	5
V3	54	1	29	1	∞	0	46	1	36	2	∞	1	27	3	182	2	78	2	34	2
V4	∞	0	33	1	36	5	∞	0	∞	0	30	1	36	1	27	2	39	3	∞	1
V5	35	4	∞	0	39	2	63	3	70	3	29	5	30	1	∞	0	∞	0	∞	0
V6.1	∞	2	43	1	30	2	45	3	36	3	105	2	43	4	∞	1	38	2	21	1
V6.2	∞	4	∞	1	∞	2	60	1	25	2	37	1	27	2	26	2	∞	0	71	4
V7	37	3	∞	0	∞	0	31	0	56	2	∞	0	∞	0	∞	1	42	2	∞	1
V8	28	2	28	2	25	2	18	1	30	4	28	5	24	1	34	2	21	2	27	2
V9	69	0	22	3	22	0	74	2	138	10	32	2	∞	3	23	2	∞	3	∞	1

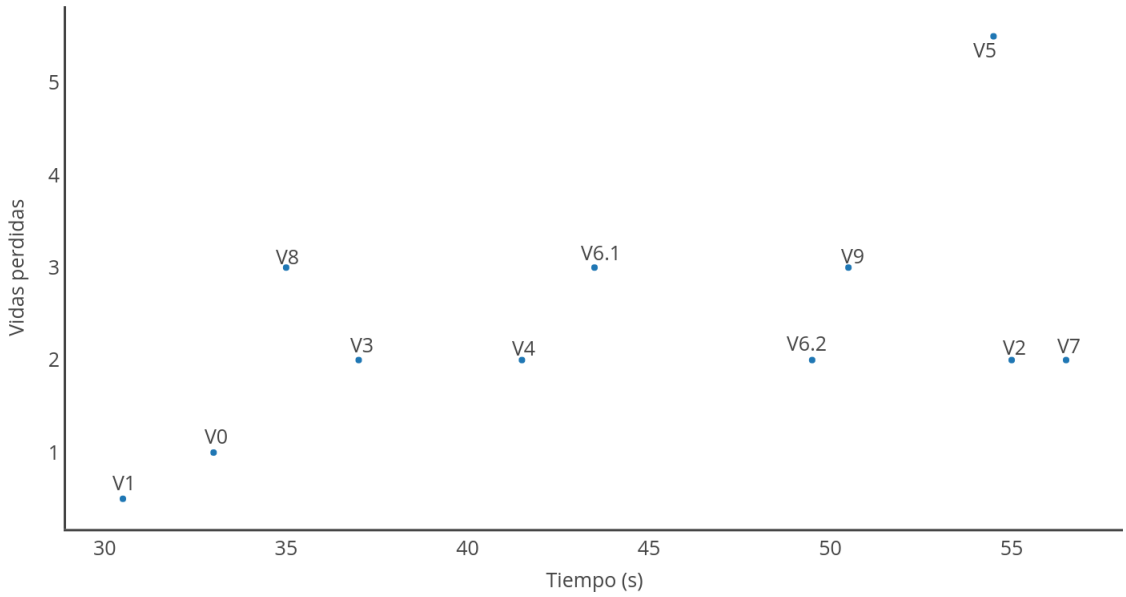

Figura 4.7: Representación gráfica de la eficacia de la versión 3 contra las diferentes versiones de los fantasmas en el primer nivel.

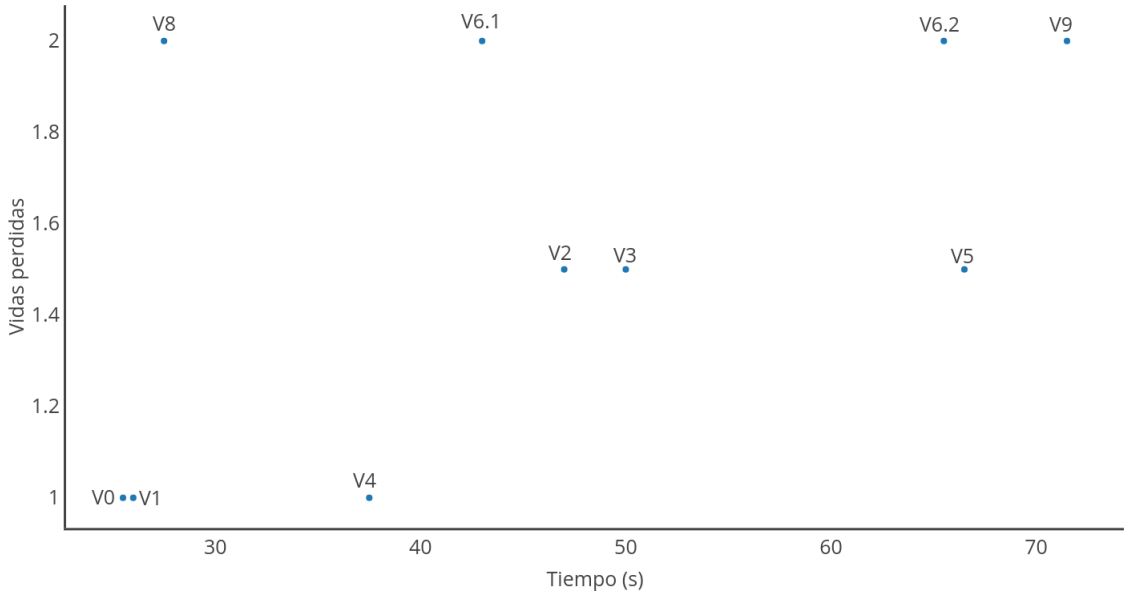

Figura 4.8: Representación gráfica de la eficacia de la versión 3 contra las diferentes versiones de los fantasmas en el segundo nivel.

4.4.2. Conclusiones

Lo primero que se ve en los resultados de esta versión es que consigue completar el juego de forma más habitual. Esto se debe al uso que hace este agente de las *Power Pellet* y a que su localización coincide mayoritariamente con zonas en las que los fantasmas se quedan protegiendo la comida indefinidamente. Al usar el poder de las *Power Pellet*, *Pac-Man* ignora a los fantasmas y consigue entrar en estas zonas anteriormente inaccesibles.

Como se aprecia en los gráficos, los resultados son parecidos a los de la versión anterior cuando el agente se enfrenta a las primeras versiones de los fantasmas y mejoran de manera significativa en versiones más avanzadas. Aunque esta mejora la provoca el uso de las *Power Pellet*, depende mucho del azar, puesto *Pac-Man* no siempre consume estas capsulas en el momento oportuno y, por tanto, a veces no le ayudan a avanzar en su objetivo.

4.5. Versión 4

4.5.1. Resultados de las pruebas

Los resultados de las pruebas de esta versión se muestran en las tablas 4.9 y 4.10.

Tabla 4.9: Resultados de la versión 4 en el primer nivel, donde t es el tiempo en segundos y v el número de vidas perdidas.

	n.º de prueba																			
	1		**2**		**3**		**4**		**5**		**6**		**7**		**8**		**9**		**10**	
	t	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**
V0	28	2	28	0	29	0	38	3	37	4	24	0	26	0	30	1	44	2	30	3
V1	27	2	29	0	32	1	29	1	46	2	29	1	31	2	29	1	33	0	28	1
V2	32	2	48	1	39	3	30	2	33	2	34	2	67	2	38	2	39	0	36	1
V3	25	0	32	1	26	1	32	3	29	1	27	2	32	2	43	4	34	2	29	1
V4	29	1	27	1	29	1	27	0	32	1	44	2	32	2	32	1	30	1	49	4
V5	36	4	65	8	36	5	56	7	50	3	43	5	45	4	30	2	25	0	34	1
V6.1	31	3	29	3	85	10	35	2	55	4	30	1	89	6	30	3	88	13	100	14
V6.2	29	1	39	2	34	1	43	5	28	1	27	1	26	1	37	1	40	0	30	2
V7	46	5	35	1	26	2	30	3	31	5	31	3	∞	1	24	1	31	2	50	12
V8	26	1	28	3	26	1	39	4	32	2	32	6	26	1	24	3	34	3	54	5
V9	34	0	35	6	43	3	36	3	29	2	33	2	38	3	34	3	33	2	32	2

Tabla 4.10: Resultados de la versión 4 en el segundo nivel, donde t es el tiempo en segundos y v el número de vidas perdidas.

	n.º de prueba																			
	1		**2**		**3**		**4**		**5**		**6**		**7**		**8**		**9**		**10**	
	t	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**
V0	23	2	24	0	19	0	19	0	23	1	22	2	28	1	21	1	29	4	27	2
V1	27	1	23	1	25	2	21	0	22	1	22	2	26	1	27	3	23	2	26	3
V2	27	1	20	0	24	0	20	1	30	2	43	1	25	1	23	0	31	1	40	6
V3	21	1	32	0	∞	0	20	1	23	2	27	1	23	2	23	2	30	2	21	2
V4	28	3	59	3	40	4	31	2	26	1	61	7	26	1	25	3	31	1	26	1
V5	27	2	24	3	24	3	∞	0	26	4	∞	1	∞	2	43	3	∞	2	37	1
V6.1	19	1	55	5	43	3	∞	0	∞	2	39	4	24	3	39	5	29	1	27	2
V6.2	27	2	58	3	21	0	34	3	26	0	45	4	21	0	19	0	28	1	35	2
V7	24	1	28	2	∞	0	26	1	22	2	49	4	25	1	26	1	46	1	28	4
V8	30	3	23	1	24	1	19	1	18	1	31	2	23	1	22	2	22	0	26	4
V9	27	2	28	5	22	2	26	1	22	1	28	4	27	0	34	9	34	1	22	1

Como siempre, se puede ver la eficacia de esta versión contra fantasmas del mismo modo de juego en las siguientes figuras: 4.9 y 4.10.

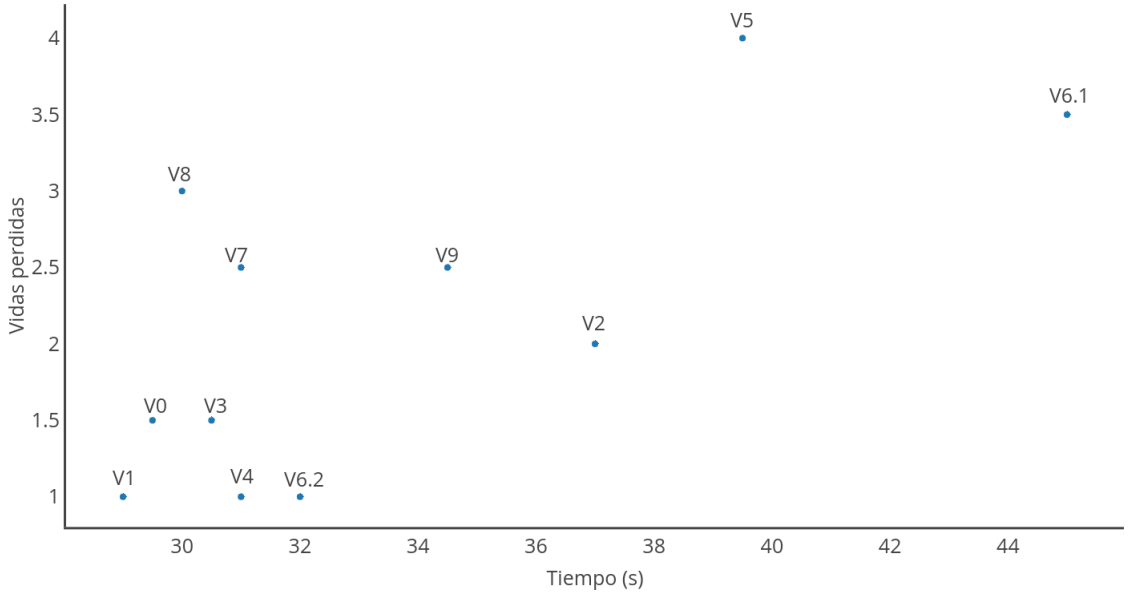

Figura 4.9: Representación gráfica de la eficacia de la versión 4 contra las diferentes versiones de los fantasmas en el primer nivel.

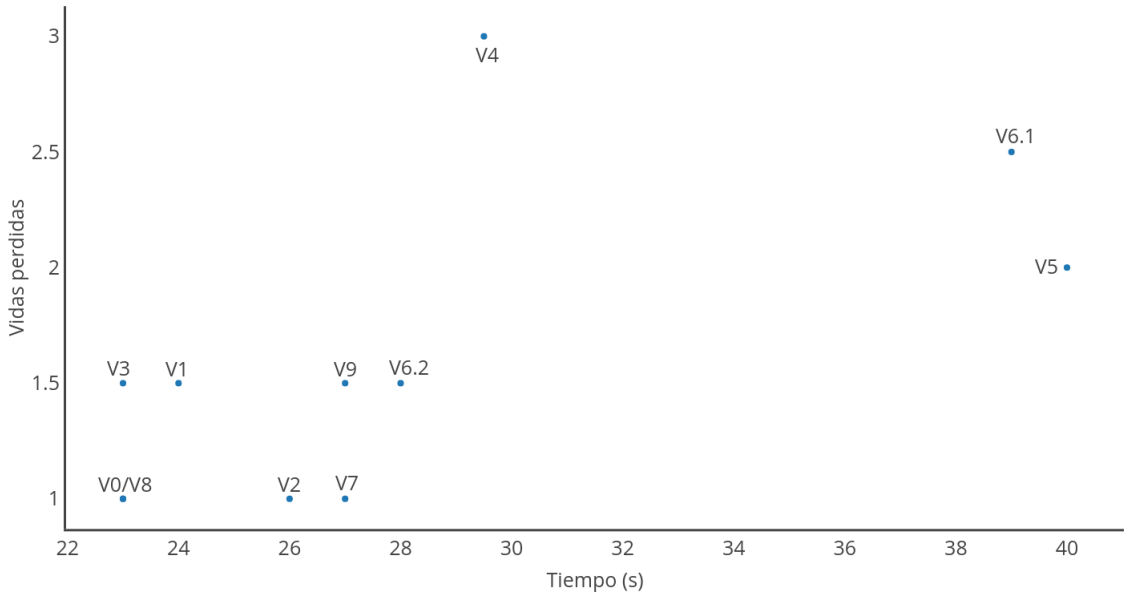

Figura 4.10: Representación gráfica de la eficacia de la versión 4 contra las diferentes versiones de los fantasmas en el segundo nivel.

4.5.2. Conclusiones

A diferencia de las anteriores, la versión 4 completa el juego casi siempre, incluso jugando en el segundo nivel. Esto se debe a que esta versión de *Pac-Man* tiene más tolerancia a los fantasmas, pudiendo acercarse más a ellos y accediendo a zonas a las que antes no se atrevía a aproximarse por miedo a sus cazadores.

En las representaciones gráficas se puede apreciar que esta versión es la que mejor rinde hasta ahora, necesitando la mayoría de veces menos de 40 segundos para completar el primer nivel y menos de 30 para el segundo. En lo que respecta a las vidas, también se consiguen resultados muy satisfactorios, perdiendo la mayoría de veces menos de 3 vidas.

4.6. Versión 5

4.6.1. Resultados de las pruebas

Los resultados de la versión 5 se pueden ver en las tablas 4.11 y 4.12.

Tabla 4.11: Resultados de la versión 5 en el primer nivel, donde t es el tiempo en segundos y v el número de vidas perdidas.

	n.º de prueba																			
	1		**2**		**3**		**4**		**5**		**6**		**7**		**8**		**9**		**10**	
	t	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**
V0	28	2	25	2	35	3	27	2	29	1	31	1	31	1	50	3	35	0	33	3
V1	43	4	30	2	37	2	36	2	35	1	33	1	36	2	30	1	23	0	22	1
V2	37	2	50	3	40	1	52	2	51	5	39	0	28	0	34	2	55	3	41	1
V3	38	0	37	1	53	3	∞	1	41	1	51	2	38	1	30	0	28	0	32	1
V4	34	1	39	2	41	3	50	0	28	0	37	4	29	2	36	3	∞	2	31	1
V5	58	3	24	1	51	3	54	5	49	5	32	2	37	2	26	2	45	4	31	2
V6.1	44	4	48	6	44	4	55	4	34	1	103	8	33	3	35	3	34	4	44	7
V6.2	31	0	32	0	27	1	35	3	28	2	83	7	45	1	43	4	35	3	31	1
V7	36	3	30	2	55	5	39	4	31	1	32	2	47	4	35	3	40	3	40	1
V8	52	4	32	2	31	2	34	0	48	3	35	2	25	2	28	2	34	4	34	0
V9	45	4	29	1	48	7	36	3	37	3	51	4	27	2	25	2	∞	3	28	2

Tabla 4.12: Resultados de la versión 5 en el segundo nivel, donde t es el tiempo en segundos y v el número de vidas perdidas.

	n.º de prueba																			
	1		**2**		**3**		**4**		**5**		**6**		**7**		**8**		**9**		**10**	
	t	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**
V0	31	3	23	1	22	2	26	1	31	2	23	3	29	2	21	0	24	1	23	1
V1	20	0	24	0	27	2	23	0	30	1	31	4	32	3	20	1	23	1	31	1
V2	21	1	23	2	42	0	29	1	28	2	30	2	29	1	27	1	27	1	20	0
V3	29	2	32	1	28	1	30	1	31	3	29	2	41	2	∞	1	27	3	35	5
V4	∞	1	∞	0	29	0	30	1	38	3	26	2	40	3	∞	0	38	2	23	2
V5	∞	0	23	2	37	3	23	1	37	3	61	4	34	1	49	5	∞	0	41	1
V6.1	23	3	∞	0	23	0	28	2	23	2	35	1	35	6	22	0	25	2	69	9
V6.2	27	5	26	2	29	0	22	2	23	0	38	3	51	2	29	1	28	2	40	5
V7	∞	0	35	2	46	5	∞	0	47	4	30	3	58	7	37	4	26	2	25	2
V8	30	2	19	1	27	2	22	1	24	0	23	2	25	2	22	2	21	2	34	1
V9	29	4	∞	0	20	0	25	2	22	1	36	3	25	2	48	6	28	1	27	1

En las figuras 4.11 y 4.12 se pueden ver los gráficos correspondientes a las pruebas.

4.6.2. Conclusiones

En general los resultados de esta versión son similares a los de la versión 4. Esto se debe en gran medida a que el cambio introducido en la versión 5 solo evita la muerte de *Pac-Man* en situaciones muy concretas y, por tanto, no se llega a sacarle provecho.

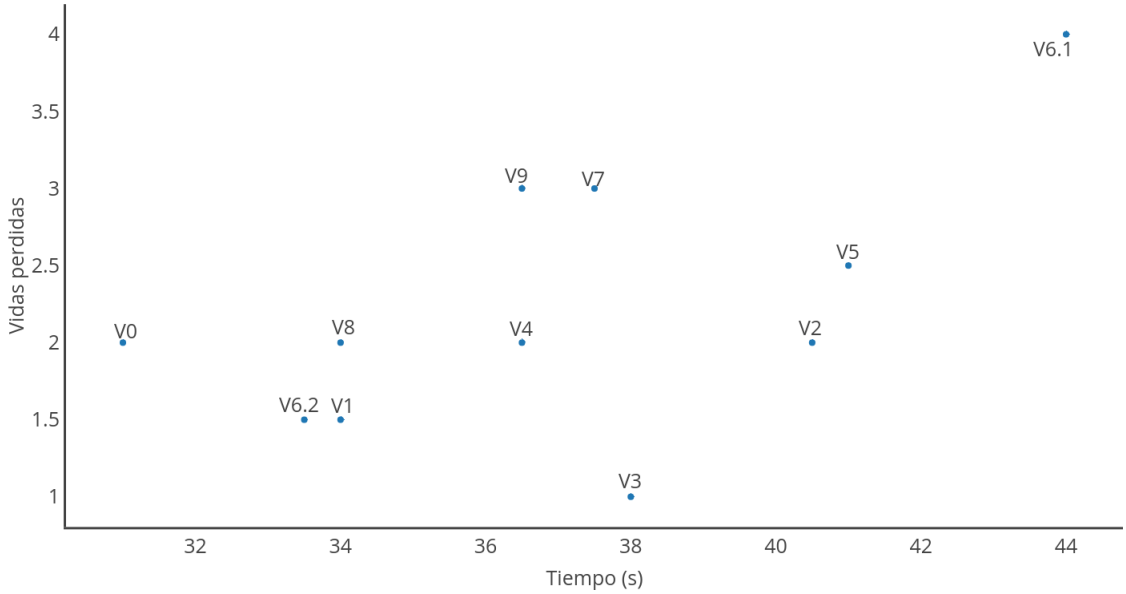

Figura 4.11: Representación gráfica de la eficacia de la versión 5 contra las diferentes versiones de los fantasmas en el primer nivel.

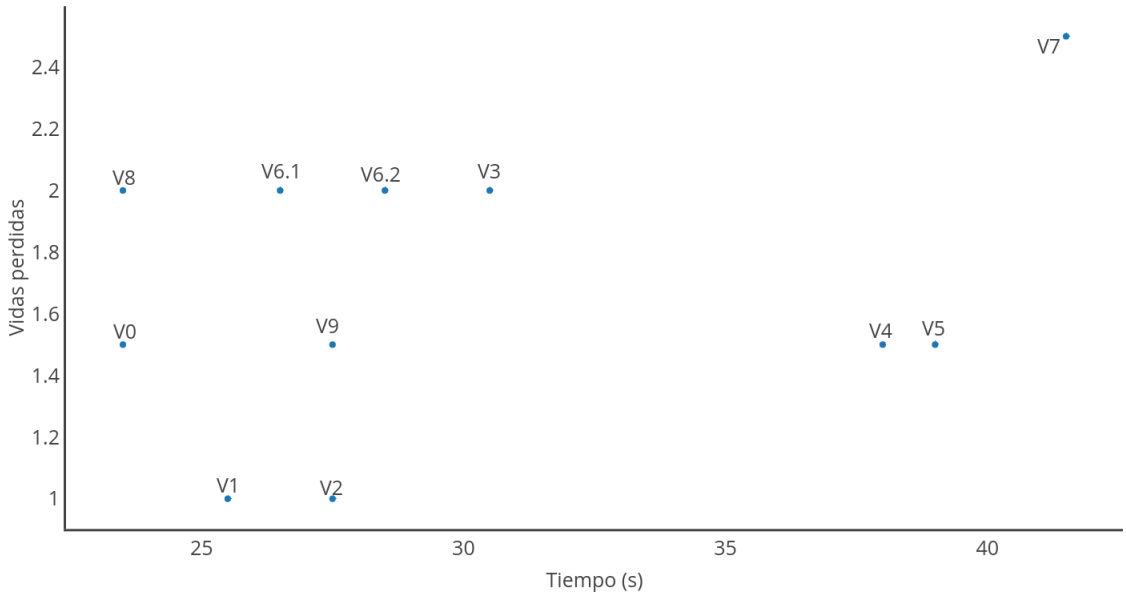

Figura 4.12: Representación gráfica de la eficacia de la versión 5 contra las diferentes versiones de los fantasmas en el segundo nivel.

4.7. Primer modo de juego

Habiendo realizado las pruebas de todas las versiones del primer modo de juego, se puede ilustrar de manera gráfica la eficacia que tienen de media las inteligencias de *Pac-Man* contra cualquier inteligencia de los fantasmas. La figura 4.13 corresponde al primer laberinto y la figura 4.14 al segundo.

Como se puede ver a simple vista, las versiones 4 y 5 son las que mejor funcionan, tanto en tiempo como en vidas perdidas, siendo los resultados de la versión 4 ligeramente mejores. La siguiente versión con el mejor tiempo es la primera en ambos niveles, mientras que la versión 3 tarda más, pero pierde la misma cantidad de vidas que las dos mejores versiones. Por último, están las versiones 2 y 0, que son peores en todos los sentidos, sobre todo teniendo en cuenta que la versión 0 no suele completar el segundo nivel.

En conclusión, los comportamientos que se han añadido hasta llegar a la versión 4 han sido beneficiosas, resultando innecesaria la modificación realizada para la versión 5 en la mayoría de las situaciones.

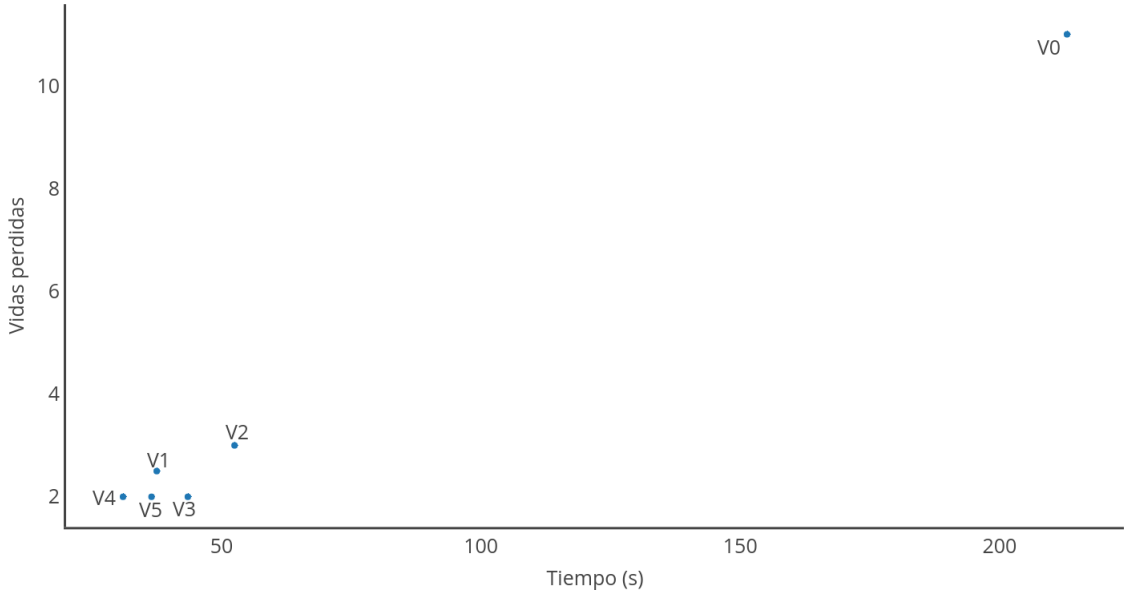

Figura 4.13: Representación gráfica de la eficacia de las versiones de *Pac-Man* del primer modo de juego en el nivel 1.

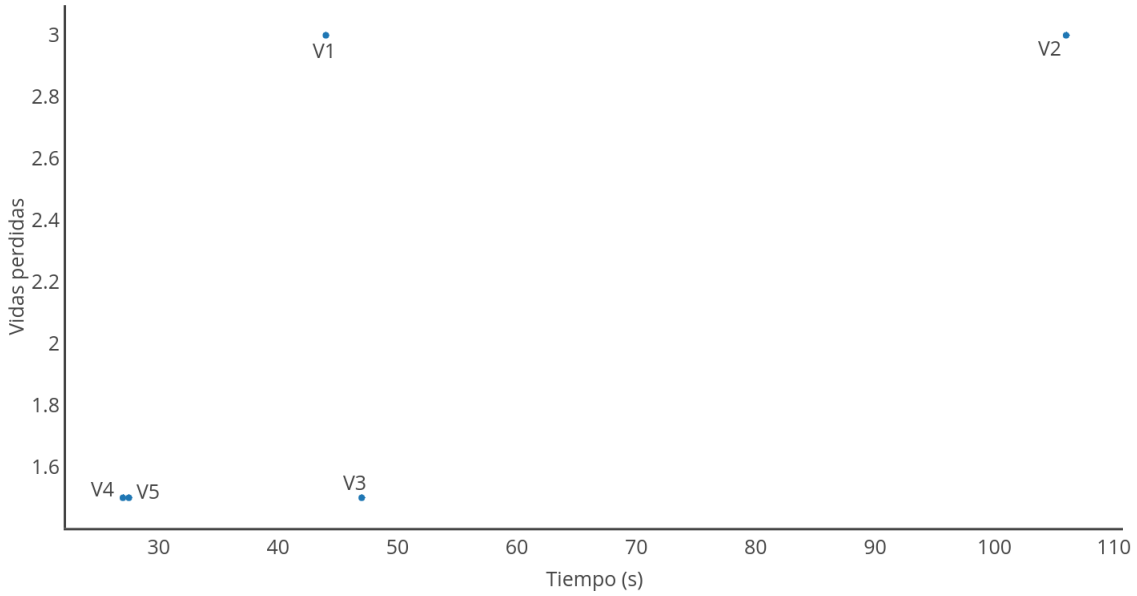

Figura 4.14: Representación gráfica de la eficacia de todas las versiones de *Pac-Man* del primer modo de juego en el nivel 2.

4.8. Versión 6

4.8.1. Resultados de las pruebas

Esta es la primera versión del segundo modo de juego y, por tanto, a partir de este momento en las tablas de resultados solo habrá versiones de los fantasmas de este segundo modo. La tabla 4.13 contiene los resultados del nivel 1 y la tabla 4.14 los del nivel 2.

Tabla 4.13: Resultados de la versión 6 en el primer nivel, donde t es el tiempo en segundos y v el número de vidas perdidas.

	n.º de prueba																			
	1		**2**		**3**		**4**		**5**		**6**		**7**		**8**		**9**		**10**	
	t	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**
V10	29	1	22	1	30	3	19	1	35	2	17	0	25	3	23	1	37	3	21	1
V11	25	2	40	4	31	0	26	2	44	4	27	2	56	5	40	4	23	1	43	6
V12	18	1	45	4	21	1	42	1	42	3	19	2	26	2	31	2	30	3	38	2
V13	31	3	17	1	24	2	38	4	19	2	33	1	27	1	24	2	18	1	30	2

En las figuras 4.15 y 4.16 se pueden ver los gráficos correspondientes a las pruebas.

Tabla 4.14: Resultados de la versión 6 en el segundo nivel, donde t es el tiempo en segundos y v el número de vidas perdidas.

	n.º de prueba																			
	1		**2**		**3**		**4**		**5**		**6**		**7**		**8**		**9**		**10**	
	t	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**
V10	31	5	13	0	24	2	18	1	18	1	16	1	42	5	21	2	16	1	18	0
V11	13	0	14	1	22	1	20	1	18	2	20	0	28	4	21	2	19	1	28	3
V12	16	1	26	4	15	1	29	2	20	2	20	1	38	3	14	0	20	2	47	3
V13	28	2	18	1	25	2	14	1	26	2	14	2	22	3	21	3	23	1	20	2

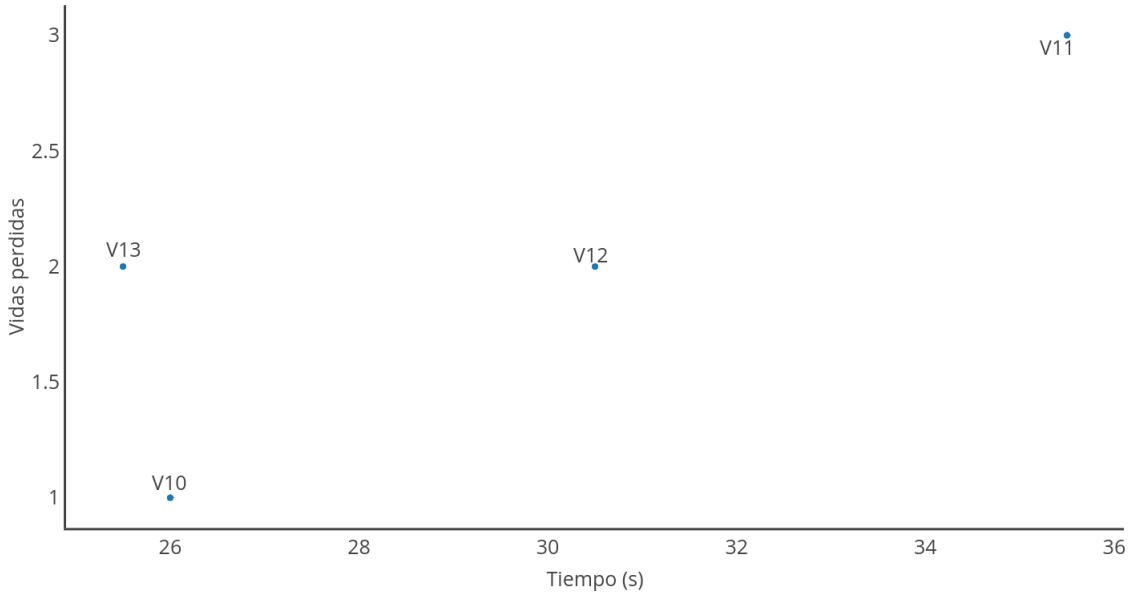

Figura 4.15: Representación gráfica de la eficacia de la versión 6 contra las diferentes versiones de los fantasmas en el primer nivel.

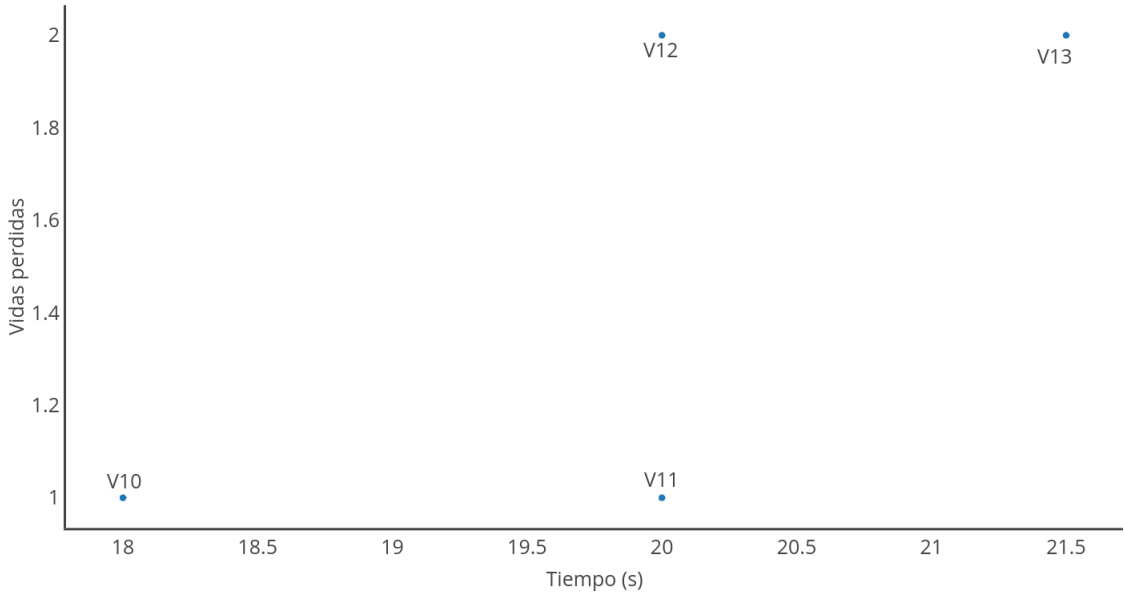

Figura 4.16: Representación gráfica de la eficacia de la versión 6 contra las diferentes versiones de los fantasmas en el segundo nivel.

4.8.2. Conclusiones

Como se puede ver en las imágenes, esta versión funciona mejor contra la versión 10 de los fantasmas, que también es la primera de este modo de juego. Los resultados contra las demás versiones varían en cada nivel, aunque las vidas perdidas se mantienen siempre iguales o menores a dos, excepto contra la versión 11 en el primer nivel.

4.9. Versión 7

4.9.1. Resultados de las pruebas

En las tablas 4.15 y 4.16 se pueden ver los resultados obtenidos con la versión 7.

Las representaciones gráficas de las pruebas se ilustran en las figuras 4.17 y 4.18.

Tabla 4.15: Resultados de la versión 7 en el primer nivel, donde t es el tiempo en segundos y v el número de vidas perdidas.

	n.º de prueba																			
	1		**2**		**3**		**4**		**5**		**6**		**7**		**8**		**9**		**10**	
	t	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**
V10	19	1	18	1	24	1	20	1	45	4	24	2	18	0	23	1	25	1	21	1
V11	26	2	30	2	34	2	31	2	22	1	23	3	25	3	22	1	19	0	19	0
V12	24	3	50	6	25	1	29	4	34	1	17	0	22	1	75	5	24	3	55	5
V13	37	4	38	2	24	2	56	3	35	1	19	1	63	9	33	2	21	2	22	1

Tabla 4.16: Resultados de la versión 7 en el segundo nivel, donde t es el tiempo en segundos y v el número de vidas perdidas.

	n.º de prueba																			
	1		**2**		**3**		**4**		**5**		**6**		**7**		**8**		**9**		**10**	
	t	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**
V10	19	1	15	1	23	4	15	1	12	0	23	2	15	1	24	1	21	2	25	2
V11	12	1	24	2	18	2	32	3	37	2	30	3	13	0	35	5	30	1	19	1
V12	19	1	16	1	25	1	23	2	44	4	18	1	16	1	21	2	20	1	17	1
V13	19	1	29	1	23	2	16	1	24	1	29	1	20	1	17	1	21	3	27	1

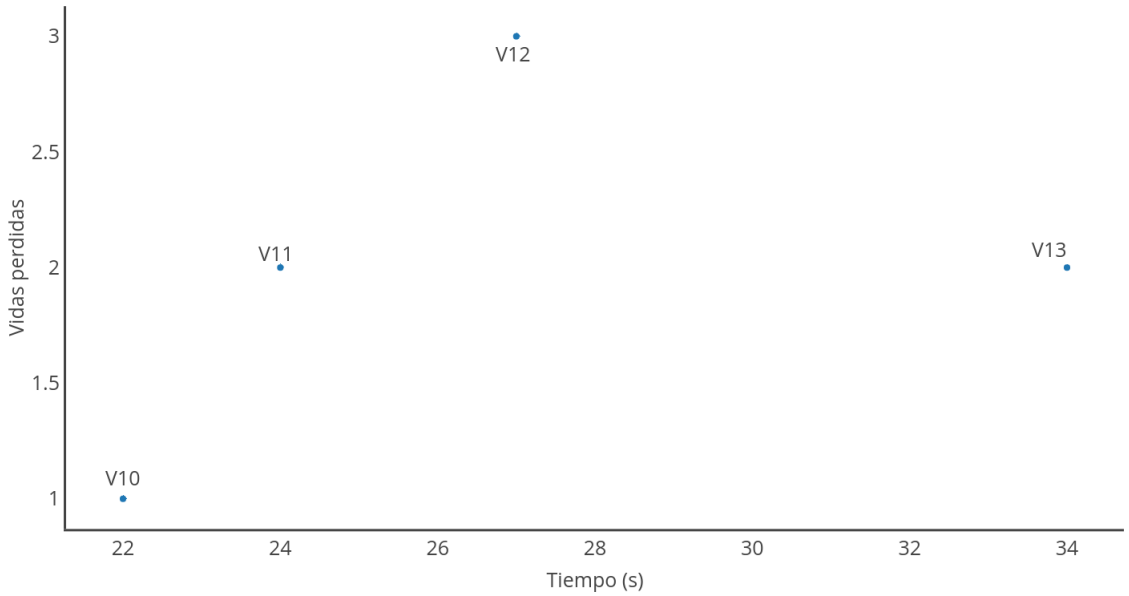

Figura 4.17: Representación gráfica de la eficacia de la versión 7 contra las diferentes versiones de los fantasmas en el primer nivel.

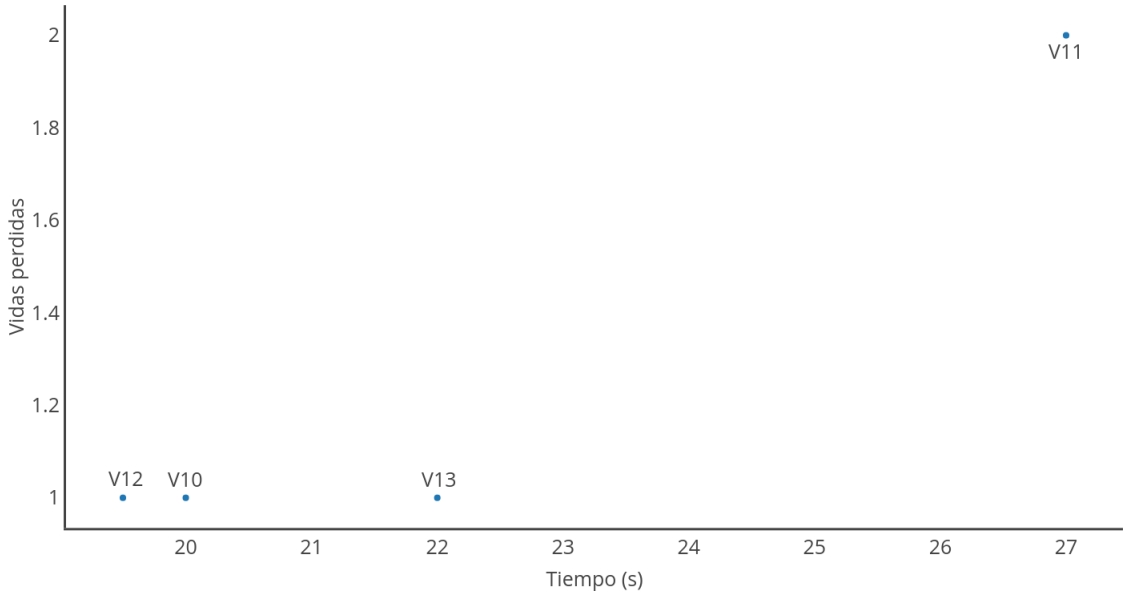

Figura 4.18: Representación gráfica de la eficacia de la versión 7 contra las diferentes versiones de los fantasmas en el segundo nivel.

4.9.2. Conclusiones

Esta versión mejora los resultados del tiempo de la anterior versión en el primer nivel, pero los empeora en el segundo, siendo una versión bastante irregular. Esto se debe en gran medida al uso que hacen los agentes de los teletransportadores combinado con la nueva capacidad de repartirse el laberinto entre los dos agentes. Al tener cada uno de los dos agentes una zona del mapa asignada, intentarán acercarse a esta zona cuando no haya otra acción prioritaria como escapar o ir hacia los *Pac-dot*. El problema viene cuando un agente en su zona designada entra al teletransportador y aparece en el otro extremo del laberinto, teniendo que cruzar el mapa para volver a su zona y, por consiguiente, perdiendo tiempo en el proceso.

Por otra parte, en los gráficos se aprecia que esta versión va empeorando los resultados en el orden de las versiones de los fantasmas. Así, los mejores resultados se obtienen contra la versión 10 y los peores contra la versión 13.

4.10. Versión 8

4.10.1. Resultados de las pruebas

Los resultados de las pruebas de esta versión se encuentran en las tablas 4.17 y 4.18.

Tabla 4.17: Resultados de la versión 8 en el primer nivel, donde t es el tiempo en segundos y v el número de vidas perdidas.

	n.º de prueba																			
	1		**2**		**3**		**4**		**5**		**6**		**7**		**8**		**9**		**10**	
	t	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**
V10	23	0	24	1	21	1	18	1	21	1	40	4	26	1	41	6	28	2	27	2
V11	21	2	23	2	17	0	21	2	33	2	43	4	24	3	43	3	24	1	19	1
V12	24	3	27	1	27	1	26	2	25	2	27	1	31	2	24	1	27	2	30	2
V13	36	2	26	2	19	2	18	1	15	1	23	2	30	3	28	2	54	3	25	1

Tabla 4.18: Resultados de la versión 8 en el segundo nivel, donde t es el tiempo en segundos y v el número de vidas perdidas.

	n.º de prueba																			
	1		**2**		**3**		**4**		**5**		**6**		**7**		**8**		**9**		**10**	
	t	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**
V10	24	1	25	2	14	1	20	2	25	1	16	1	25	1	19	1	16	1	20	2
V11	14	0	20	1	24	1	14	0	13	0	22	1	18	1	13	1	23	2	18	2
V12	15	0	20	0	15	2	23	2	29	3	22	3	31	0	25	1	18	0	25	2
V13	23	3	24	3	24	3	19	2	17	1	18	0	34	2	27	2	16	0	20	2

Los gráficos con las eficacias de la versión 8 en cada nivel se pueden ver en las figura 4.19 y 4.20.

4.10.2. Conclusiones

Esta versión mejora el tiempo de la versión 6, que es en la que se basa, en el primer nivel. Por otra parte, en el segundo nivel aumenta el tiempo y reduce la cantidad de vidas perdidas. Aun así, los resultados no son demasiado significativos porque esta versión tiene la misma inteligencia que la versión en la que se basa, con la única diferencia de que ahora los agentes poseen la habilidad de resucitarse el uno al otro.

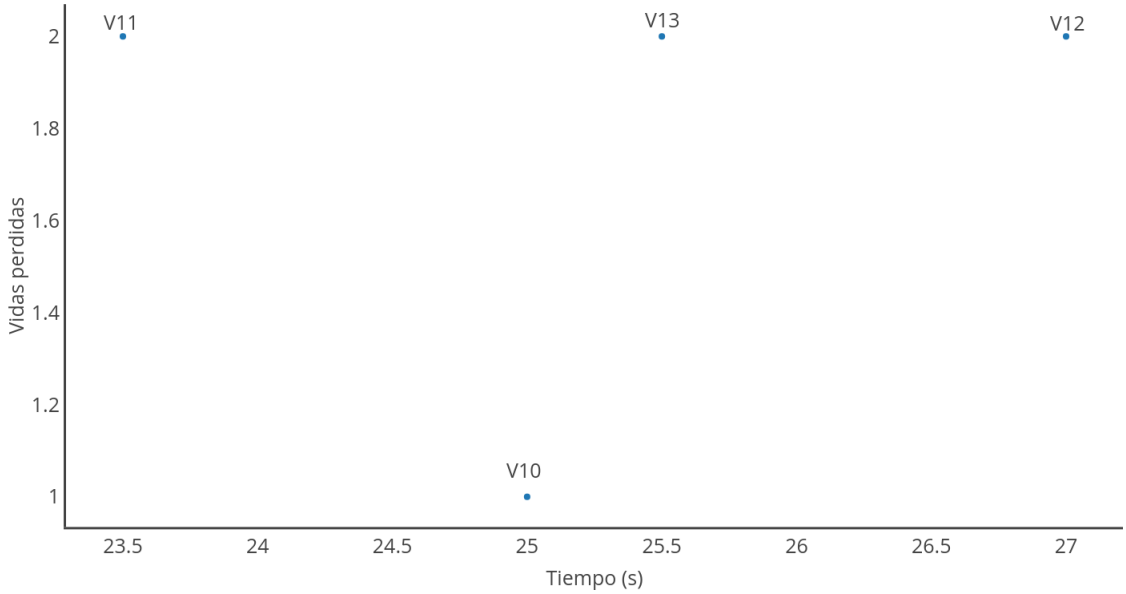

Figura 4.19: Representación gráfica de la eficacia de la versión 8 contra las diferentes versiones de los fantasmas en el primer nivel.

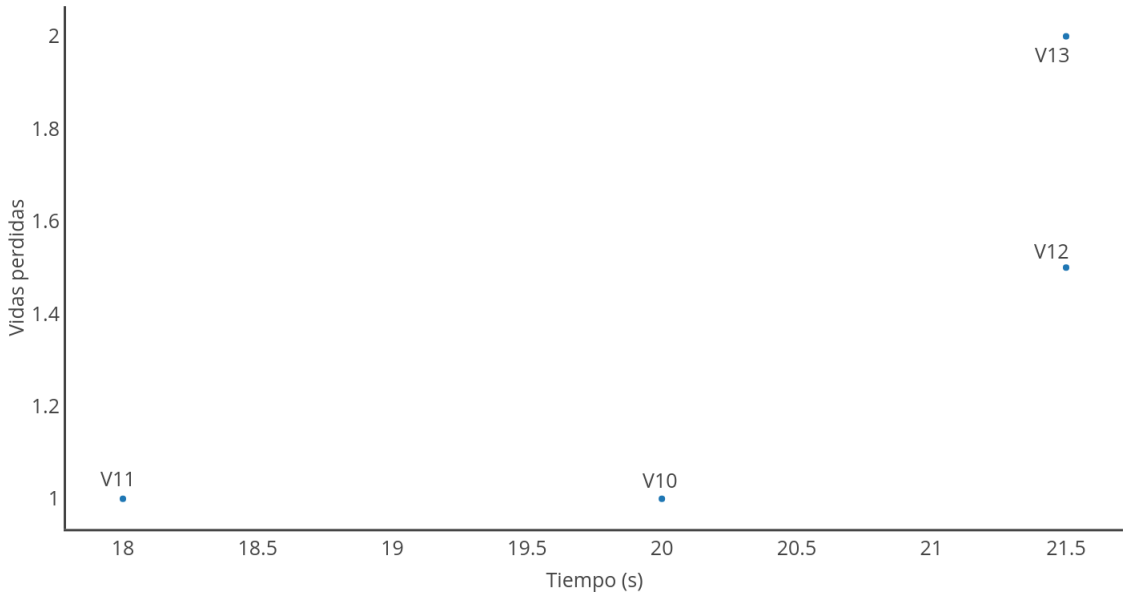

Figura 4.20: Representación gráfica de la eficacia de la versión 8 contra las diferentes versiones de los fantasmas en el segundo nivel.

4.11. Versión 9

4.11.1. Resultados de las pruebas

Las tablas 4.19 y 4.20 contienen los resultados de esta versión y las figuras 4.21 y 4.22 ilustran las medianas de estos resultados.

Tabla 4.19: Resultados de la versión 9 en el primer nivel, donde t es el tiempo en segundos y v el número de vidas perdidas.

	n.º de prueba																			
	1		**2**		**3**		**4**		**5**		**6**		**7**		**8**		**9**		**10**	
	t	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**
V10	22	0	23	0	21	1	33	2	21	2	30	1	36	4	24	1	29	3	24	0
V11	17	1	28	1	18	1	33	3	28	2	31	2	38	1	27	2	28	2	36	3
V12	17	0	21	2	44	5	35	2	42	2	34	2	27	2	25	2	19	1	45	2
V13	22	2	28	1	20	2	24	2	19	1	25	1	36	3	24	1	19	0	45	3

Tabla 4.20: Resultados de la versión 9 en el segundo nivel, donde t es el tiempo en segundos y v el número de vidas perdidas.

	n.º de prueba																			
	1		**2**		**3**		**4**		**5**		**6**		**7**		**8**		**9**		**10**	
	t	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**
V10	31	1	20	1	20	0	17	0	26	1	19	0	25	1	26	0	35	2	28	2
V11	20	1	24	3	24	1	18	0	19	0	14	1	15	1	18	0	22	3	16	0
V12	24	2	40	3	15	1	55	7	22	1	15	0	24	3	24	3	19	1	33	1
V13	23	0	25	2	25	1	15	1	13	1	16	1	13	0	13	0	20	1	10	0

4.11.2. Conclusiones

Como se puede ver en los resultados, esta versión reduce el número de vidas perdidas de la versión previa, aunque empeora el resultado del tiempo. Esto es razonable si se tiene en cuenta que esta solo pretende reducir el número de vidas perdidas, dándole prioridad a resucitar al agente compañero por delante de comer los *Pac-dot*.

Centrándonos en las vidas perdidas, en el primer nivel los agentes tienen más éxito contra la versión 10 de los fantasmas, empeorando contra las versiones siguientes, que tienen la capacidad de vigilar a los agentes muertos para que no se puedan resucitar. En cuanto

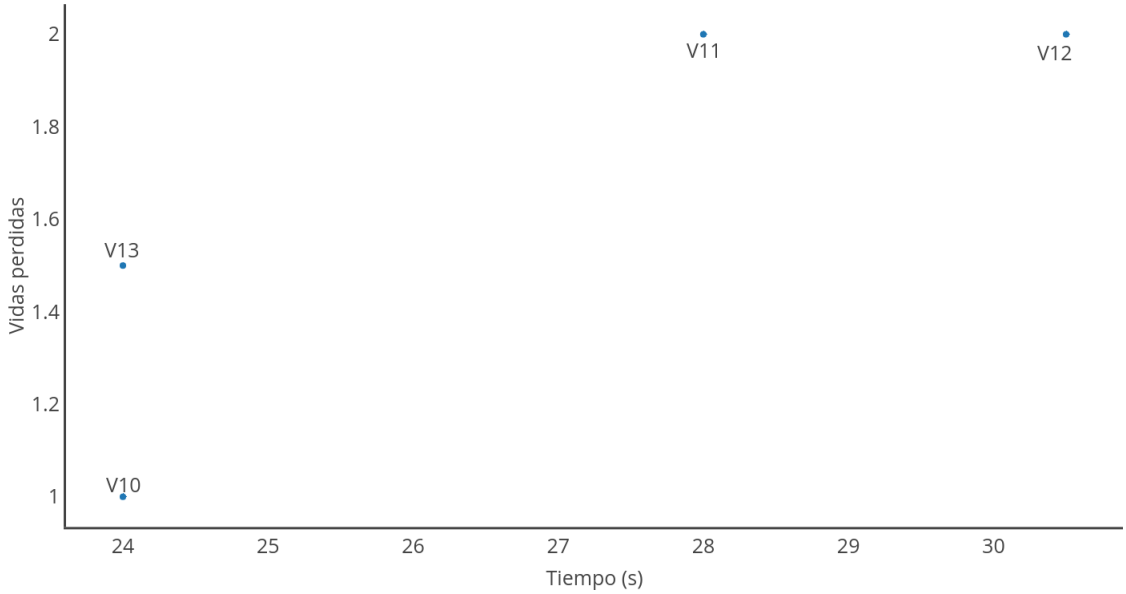

Figura 4.21: Representación gráfica de la eficacia de la versión 9 contra las diferentes versiones de los fantasmas en el primer nivel.

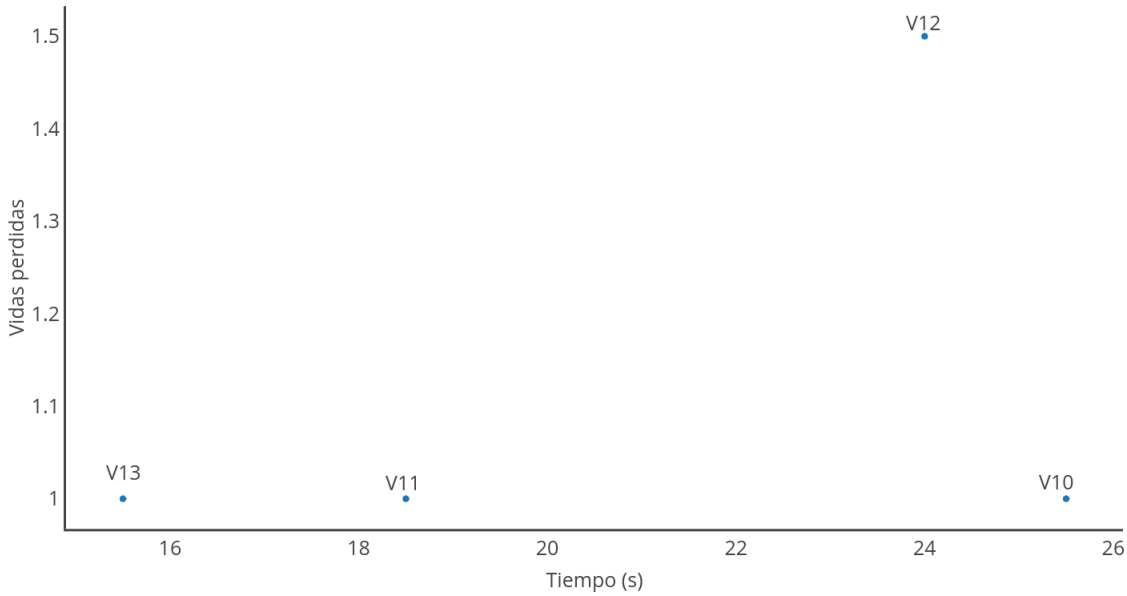

Figura 4.22: Representación gráfica de la eficacia de la versión 9 contra las diferentes versiones de los fantasmas en el segundo nivel.

al segundo nivel, esta versión obtiene buenos resultados contra todas las versiones de los fantasmas, aunque sean ligeramente peores contra la versión 12.

4.12. Versión 10

4.12.1. Resultados de las pruebas

Para terminar con las pruebas, los resultados de esta última versión se pueden encontrar en las tablas 4.21 y 4.22.

Tabla 4.21: Resultados de la versión 10 en el primer nivel, donde t es el tiempo en segundos y v el número de vidas perdidas.

	n.º de prueba																			
	1		**2**		**3**		**4**		**5**		**6**		**7**		**8**		**9**		**10**	
	t	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**
V10	34	1	21	1	17	0	22	1	16	1	23	2	24	1	33	3	27	2	20	2
V11	20	1	17	1	22	1	21	1	20	1	25	1	31	2	22	2	20	1	29	3
V12	30	3	22	1	24	0	36	3	24	1	24	2	33	2	34	2	20	1	22	2
V13	22	1	28	1	19	2	29	2	36	2	23	2	28	3	42	2	66	5	24	0

Tabla 4.22: Resultados de la versión 10 en el segundo nivel, donde t es el tiempo en segundos y v el número de vidas perdidas.

	n.º de prueba																			
	1		**2**		**3**		**4**		**5**		**6**		**7**		**8**		**9**		**10**	
	t	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**	**t**	**v**
V10	23	1	22	1	22	1	16	0	22	0	19	0	17	2	17	2	18	0	21	2
V11	24	0	18	1	22	1	17	1	27	1	15	1	17	1	20	0	17	2	20	2
V12	17	1	19	0	21	2	13	0	21	2	21	1	18	2	21	1	15	0	18	1
V13	29	1	18	1	17	2	21	1	21	1	18	1	15	1	15	0	23	0	19	1

Los gráficos que representan la eficacia de la última versión de *Pac-Man* se pueden ver en las figuras 4.23 y 4.24.

4.12.2. Conclusiones

Como se puede apreciar en los resultados, esta versión es superior a todas las anteriores en cuanto a eficiencia, tanto en tiempo como en vidas perdidas. Además, en los gráficos

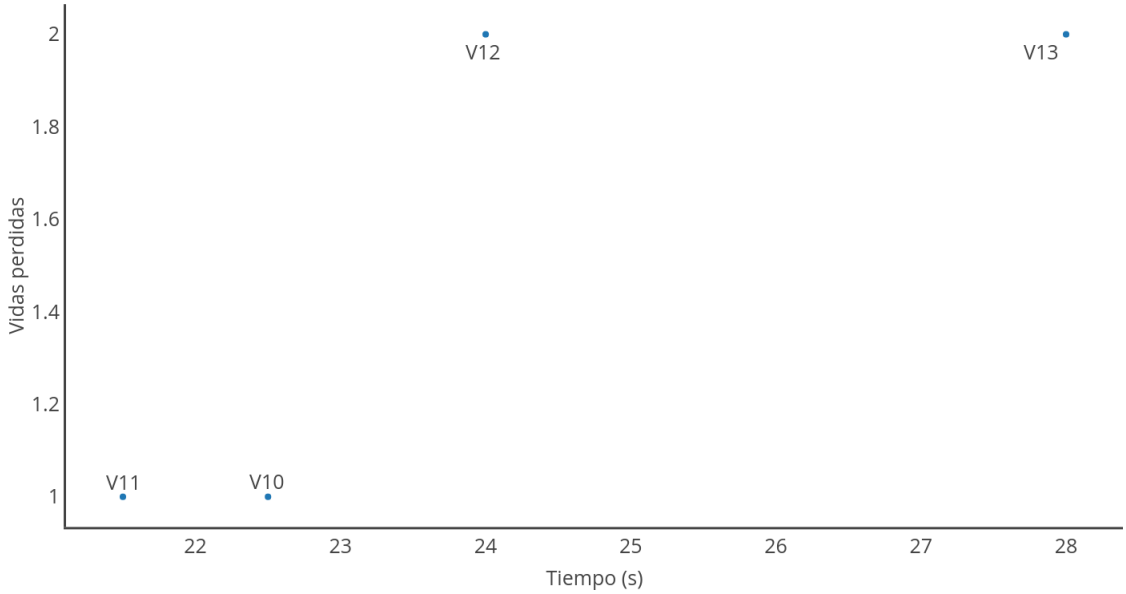

Figura 4.23: Representación gráfica de la eficacia de la versión 10 contra las diferentes versiones de los fantasmas en el primer nivel.

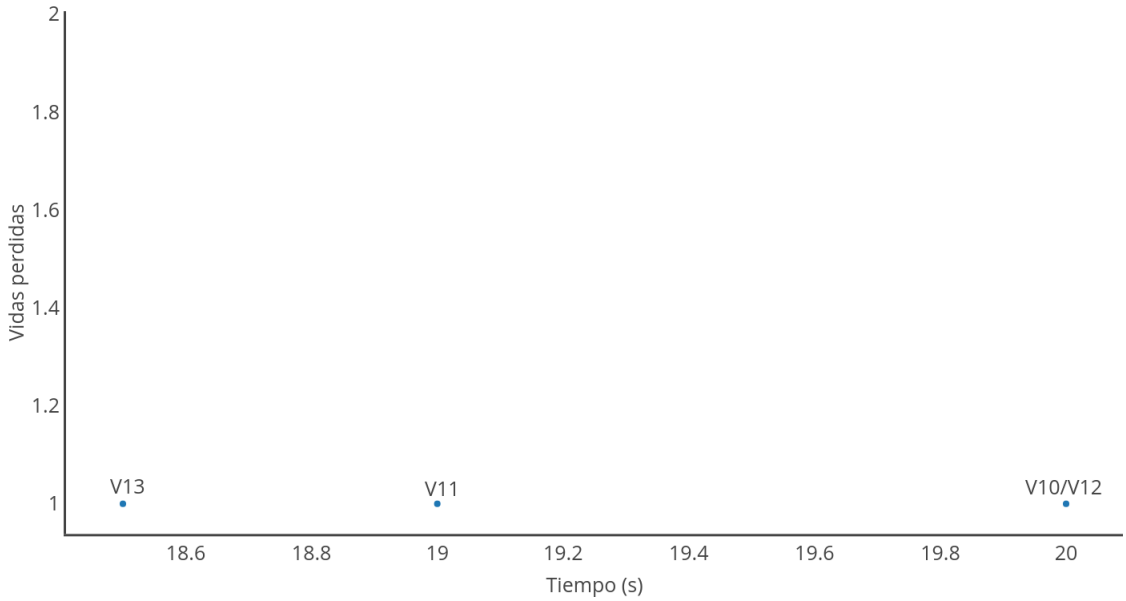

Figura 4.24: Representación gráfica de la eficacia de la versión 10 contra las diferentes versiones de los fantasmas en el segundo nivel.

se puede observar que obtiene buenos resultados contra todas las versiones, sobre todo en el segundo nivel, donde la perdida de vidas es mínima y la diferencia de tiempo entre el mejor y el peor resultado apenas alcanza los dos segundos. Por tanto, se puede concluir que darle prioridad a comer los *Pac-dot* y, por tanto, a completar el juego, antes que a la resurrección de su compañero beneficia a *Pac-Man* enormemente.

4.13. Segundo modo de juego

Después de realizar todas las pruebas del segundo modo de juego, se puede ilustrar de manera gráfica la eficacia que tienen de media las versiones de *Pac-Man* contra cualquier versión de los fantasmas. La figura 4.25 corresponde al primer laberinto y la figura 4.26 al segundo.

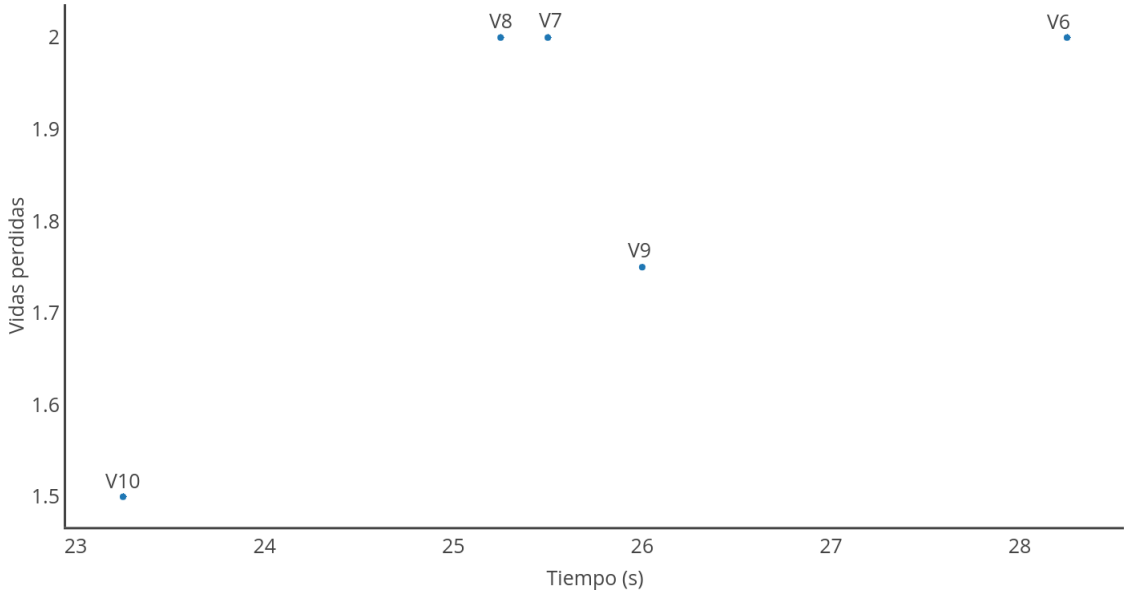

Figura 4.25: Representación gráfica de la eficacia de las versiones de *Pac-Man* del segundo modo de juego en el nivel 1.

Se puede observar que la versión 10 es la mejor en ambos niveles sin ninguna duda, puesto que obtiene los mejores resultados tanto de tiempo como de vidas perdidas. En el primer nivel le sigue la versión 9 en cuanto a vidas perdidas, que es una versión que precisamente busca perder menos vidas, pero le superan a este en tiempo las versiones 8 y 7. Por último, está la versión 6 con los peores resultados del primer nivel.

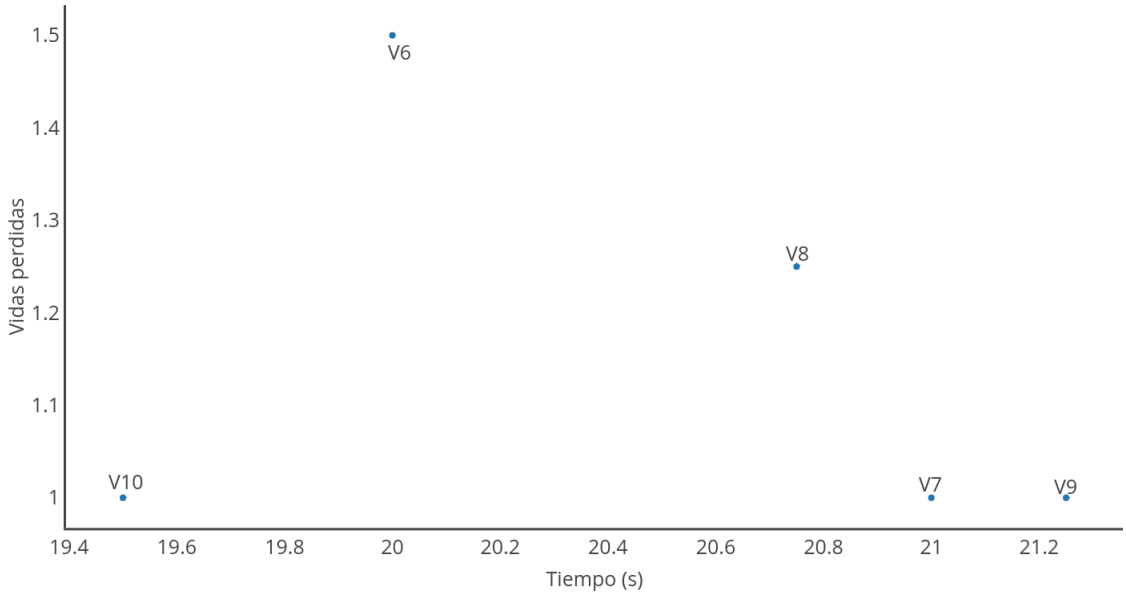

Figura 4.26: Representación gráfica de la eficacia de todas las versiones de *Pac-Man* del segundo modo de juego en el nivel 2.

En cuanto al segundo nivel, las versiones 7 y 9 pierden las mismas vidas que la versión 10, aunque para ello usan más tiempo que cualquier otra versión. Por otra parte, la versión 6 es la siguiente a la 10 en cuanto a tiempo, pero tiene los peores resultados en vidas perdidas. Para finalizar, en un punto intermedio se encuentra la versión 8, que no destaca ni en tiempo ni en vidas perdidas.

Para concluir, es evidente que los cambios realizados a los agentes han aumentado su eficiencia hasta llegar a la versión 10, aunque algunas versiones intermedias no hayan conseguido los resultados esperados.

4.14. Conclusiones generales

Para finalizar se puede concluir que se han cumplido los objetivos del proyecto, puesto que se han implementado diferentes inteligencias en Python para un agente, aprendiendo técnicas usadas actualmente en los videojuegos en el proceso. También se han analizado estas inteligencias diseñando unas pruebas adecuadas y usando técnicas estadísticas. Por tanto, se cree que este trabajo ha resultado exitoso.

Anexos

A. ANEXO

Pseudocódigo

A.1. *UpdateSound*

```
UpdateSound(limite, nivel):
    fantasma.sonido = [[100 for i in range(nivel.anchura)] for j in range(nivel.altura)]
    visitados = set()
    queue = [(filaActual, columnaActual, 0)]
    while NoEsVacio(queue):
        nodo = queue.pop()
        fila = nodo[0]
        columna = nodo[1]
        distancia = nodo[2]
        if distancia < limite:
            if (fila,columna) not in visitados:
                visitados.add((fila,columna))
                if not nivel.IsWall((fila,columna))
                    fantasma.sonido[fila][columna] = distancia
                    queue.append((fila+1,columna,distancia+1))
                    queue.append((fila-1,columna,distancia+1))
                    queue.append((fila,columna+1,distancia+1))
                    queue.append((fila,columna-1,distancia+1))
```

A.2. *GetTileSound*

```
GetTileSound(fila, columna, fantasmas):
    if NoEsVacio(fantasmas.sonido):
```

```
        valor = min(fantasmas.sonido[fila][columna]
    else:
        valor = 100
    return valor
```

A.3. *DrawSound*

```
DrawSound(fantasmas, pantalla, imagenes, juego):
    for fila in range(alturaNivel):
        for columna in range(anchuraNivel):
            sonido = GetTileSound(fila, columna, fantasmas)
            if sonido != 100:
                Dibujar(imagenes[sonido], columna, fila)
```

A.4. *IsNewTile*

```
IsNewTile():
    if (filaActual != filaAnterior) or (columnaActual != columnaAnterior):
        filaAnterior = filaActual
        columnaAnterior = columnaActual
        return True
    else:
        return False
```

A.5. *DecideSpeed*, versión 0

```
DecideSpeed(nivel, fantasmas):
    if PacmanEstaAlineado():
        if IsNewTile():
            sonido = nivel.GetTileSound(fila, columna, fantasmas)
            if sonido < sonidoAnterior:
                if PuedeVolverAtras():
                    IrEnDireccionContraria()
                else:
                    IrEnOtraDireccion()
            else if sonido > sonidoAnterior():
                if PuedeIrRecto():
                    SeguirAdelante()
```

```
                else:
                    IrEnOtraDireccion()
            else:
                IrEnCualquierDireccion()
```

A.6. *FindNearestFood*

```
FindNearestFood(nivel):
    visitados = set()
    queue = [(filaActual, columnaActual, 0, [])]
    while NoEsVacio(queue):
        nodo = queue.pop()
        fila = nodo[0]
        columna = nodo[1]
        distancia = nodo[2]
        camino = nodo[3]
        if distancia <= 10:
            if (fila,columna) not in visitados:
                visitados.add((fila,columna))
                if not nivel.IsWall((fila,columna))
                    if nivel.GetMapTile((fila,columna)) == 2:
                        return camino
                    queue.append((fila+1,columna,distancia+1, camino+'D'))
                    queue.append((fila-1,columna,distancia+1, camino+'U'))
                    queue.append((fila,columna+1,distancia+1, camino+'R'))
                    queue.append((fila,columna-1,distancia+1, camino+'L'))
        else:
            return NULL
```

A.7. *DecideSpeed*, versión 1

```
DecideSpeed(nivel, fantasmas):
    if PacmanEstaAlineado():
        if IsNewTile():
            sonido = nivel.GetTileSound(fila, columna, fantasmas)
            if sonido < sonidoAnterior:
                ReiniciarCamino()
                if PuedeVolverAtras():
                    IrEnDireccionContraria()
                else:
                    IrEnOtraDireccion()
            else if sonido > sonidoAnterior() and sonido < 5:
                ReiniciarCamino()
```

```
                if PuedeIrRecto():
                    SeguirAdelante()
                else:
                    IrEnOtraDireccion()
            else:
                if ComidaAdyacente():
                    ReiniciarCamino()
                    IrHaciaComida()
                else:
                    if EsVacio(camino):
                        camino = FindNearestFood(nivel)
                    if NoEsVacio(camino):
                        d = camino.pop()
                        if NoEsDireccionContraria(d):
                            IrEnDireccion(d)
                        else:
                            ReiniciarCamino()
                            IrEnCualquierDireccion()
                    else:
                        IrEnCualquierDireccion()
```

A.8. *FlowfieldBFS*

```
FlowfieldBFS((fila,columna)):
    heatmap = [[100 for i in range(nivel.anchura)] for j in range(nivel.altura)]
    visitados = set()
    queue = [(fila, columna, 0)]
    while NoEsVacio(queue):
        nodo = queue.pop()
        fila = nodo[0]
        columna = nodo[1]
        distancia = nodo[2]
        if (fila,columna) not in visitados:
            visitados.add((fila,columna))
            if not nivel.IsWall((fila,columna)):
                heatmap[fila][columna] = distancia
                queue.append((fila+1,columna,distancia+1))
                queue.append((fila-1,columna,distancia+1))
                queue.append((fila,columna+1,distancia+1))
                queue.append((fila,columna-1,distancia+1))
    return heatmap
```

A.9. *FlowfieldPathfinding*

```
FlowfieldPathfinding():
    for p in puertas:
        heatmaps.append(FlowfieldBFS(p))
    combinacion = [[0 for i in range(nivel.anchura)] for j in range(nivel.altura)]
    for i in range(nivel.altura):
        for j in range(nivel.anchura):
            combinacion[i][j] = min(heatmaps[i][j])
    flowfield = [['0' for i in range(nivel.anchura)] for j in range(nivel.altura)]
    for i in range(nivel.altura-1):
        for j in range(nivel.anchura-1):
            if combo[i][j] != 0:
                flowfield[i][j] = DireccionHaciaNumPeq()
    return flowfield
```

A.10. *DecideSpeed*, versión 2

```
DecideSpeed(nivel, fantasmas):
    if PacmanEstaAlineado():
        if IsNewTile():
            sonido = nivel.GetTileSound(fila, columna, fantasmas)
            if sonido < sonidoAnterior:
                ReiniciarCamino()
                if PuedeVolverAtras():
                    IrEnDireccionContraria()
                else:
                    IrEnOtraDireccion()
            else if sonido > sonidoAnterior() and sonido < 5:
                ReiniciarCamino()
                if PuedeIrRecto():
                    SeguirAdelante()
                else:
                    IrEnOtraDireccion()
            else:
                if sonido < 5:
                    ReiniciarCamino()
                    ff = flowfield[fila][columna]
                    if NoEsDireccionContraria(ff):
                        IrEnDireccion(ff)
                    else:
                        IrEnCualquierDireccion()
                else:
                    if ComidaAdyacente():
                        ReiniciarCamino()
                        IrHaciaComida()
                    else:
                        if EsVacio(camino):
```

```
                            camino = FindNearestFood(nivel)
                        if NoEsVacio(camino):
                            d = camino.pop()
                            if NoEsDireccionContraria(d):
                                IrEnDireccion(d)
                            else:
                                ReiniciarCamino()
                                IrEnCualquierDireccion()
                        else:
                            IrEnCualquierDireccion()
```

A.11. *DecideSpeed*, versión 3

```
DecideSpeed(nivel, fantasmas):
    if PacmanEstaAlineado():
        if IsNewTile():
            sonido = nivel.GetTileSound(fila, columna, fantasmas)
            if sonido < sonidoAnterior and temporizador <= 0:
                ReiniciarCamino()
                if PuedeVolverAtras():
                    IrEnDireccionContraria()
                else:
                    IrEnOtraDireccion()
            else if sonido > sonidoAnterior() and sonido < 5 and temporizador <= 0:
                ReiniciarCamino()
                if PuedeIrRecto():
                    SeguirAdelante()
                else:
                    IrEnOtraDireccion()
            else:
                if sonido < 5 and temporizador <= 0:
                    ReiniciarCamino()
                    ff = flowfield[fila][columna]
                    if NoEsDireccionContraria(ff):
                        IrEnDireccion(ff)
                    else:
                        IrEnCualquierDireccion()
                else:
                    if ComidaAdyacente():
                        ReiniciarCamino()
                        IrHaciaComida()
                    else:
                        if EsVacio(camino):
                            camino = FindNearestFood(nivel)
                        if NoEsVacio(camino):
                            d = camino.pop()
                            if NoEsDireccionContraria(d):
```

```
                                IrEnDireccion(d)
                            else:
                                ReiniciarCamino()
                                IrEnCualquierDireccion()
                        else:
                            IrEnCualquierDireccion()
```

A.12. *DecideSpeed*, versión 4

```
DecideSpeed(nivel, fantasmas):
    if PacmanEstaAlineado():
        if IsNewTile():
            sonido = nivel.GetTileSound(fila, columna, fantasmas)
            if sonido < sonidoAnterior and temporizador <= 0 and sonido < 5:
                ReiniciarCamino()
                if PuedeVolverAtras():
                    IrEnDireccionContraria()
                else:
                    IrEnOtraDireccion()
            else:
                if sonido < 5 and temporizador <= 0:
                    ReiniciarCamino()
                    ff = flowfield[fila][columna]
                    if NoEsDireccionContraria(ff):
                        IrEnDireccion(ff)
                    else:
                        IrEnCualquierDireccion()
                else:
                    if ComidaAdyacente():
                        ReiniciarCamino()
                        IrHaciaComida()
                    else:
                        if EsVacio(camino):
                            camino = FindNearestFood(nivel)
                        if NoEsVacio(camino):
                            d = camino.pop()
                            if NoEsDireccionContraria(d):
                                IrEnDireccion(d)
                            else:
                                ReiniciarCamino()
                                IrEnCualquierDireccion()
                        else:
                            IrEnCualquierDireccion()
```

A.13. *DecideSpeed*, versión 5

```
DecideSpeed(nivel, fantasmas):
    if PacmanEstaAlineado():
        if IsNewTile():
            sonido = nivel.GetTileSound(fila, columna, fantasmas)
            if teleportado:
                if SigueEnTunel():
                    SeguirAdelante()
                else:
                    teleportado = False
                    IrEnCualquierDireccion()
            else:
                if sonido < sonidoAnterior and temporizador <= 0 and sonido < 5:
                    ReiniciarCamino()
                    if PuedeVolverAtras():
                        IrEnDireccionContraria()
                    else:
                        IrEnOtraDireccion()
                else:
                    if sonido < 5 and temporizador <= 0:
                        ReiniciarCamino()
                        ff = flowfield[fila][columna]
                        if NoEsDireccionContraria(ff):
                            IrEnDireccion(ff)
                        else:
                            IrEnCualquierDireccion()
                    else:
                        if ComidaAdyacente():
                            ReiniciarCamino()
                            IrHaciaComida()
                        else:
                            if EsVacio(camino):
                                camino = FindNearestFood(nivel)
                            if NoEsVacio(camino):
                                d = camino.pop()
                                if NoEsDireccionContraria(d):
                                    IrEnDireccion(d)
                                else:
                                    ReiniciarCamino()
                                    IrEnCualquierDireccion()
                            else:
                                IrEnCualquierDireccion()
```

A.14. *DecideSpeed*, versión 7

```
DecideSpeed(nivel, fantasmas):
    if PacmanEstaAlineado():
        if IsNewTile():
            sonido = nivel.GetTileSound(fila, columna, fantasmas)
            if teleportado:
                if SigueEnTunel():
                    SeguirAdelante()
                else:
                    teleportado = False
                    IrEnCualquierDireccion()
            else:
                if sonido < sonidoAnterior and temporizador <= 0 and sonido < 5:
                    ReiniciarCamino()
                    if PuedeVolverAtras():
                        IrEnDireccionContraria()
                    else:
                        IrEnOtraDireccion()
                else:
                    if sonido < 5 and temporizador <= 0:
                        ReiniciarCamino()
                        ff = flowfield[fila][columna]
                        if NoEsDireccionContraria(ff):
                            IrEnDireccion(ff)
                        else:
                            IrEnCualquierDireccion()
                    else:
                        if ComidaAdyacente():
                            ReiniciarCamino()
                            IrHaciaComida()
                        else:
                            if EsVacio(camino):
                                camino = FindNearestFood(nivel)
                            if NoEsVacio(camino):
                                d = camino.pop()
                                if NoEsDireccionContraria(d):
                                    IrEnDireccion(d)
                                else:
                                    ReiniciarCamino()
                                    IrEnCualquierDireccion()
                            else:
                                if EstaEnZonaDesignada():
                                    IrEnCualquierDireccion()
                                else:
                                    IntentarIrHaciaZonaDesignada()
```

A.15. *RevivePacman*

```
RevivePacman(jugadores):
    if jugadores[0].muerto and not jugadores[1].muerto:
        jugadores[0].muerto = False
        contadorPacman++
        jugadores[0].IrEnDirección(jugadores[1].orientacion)
    elif jugadores[1].muerto and not jugadores[0].muerto:
        jugadores[1].muerto = False
        contadorPacman++
        jugadores[1].IrEnDirección(jugadores[0].orientacion)
```

A.16. *GetNeighbors*

```
GetNeighbors((fila,columna)):
    if not IsWall((row-1,col)):
        vecinos.append((row-1,col))
    elif not IsWall((row+1,col)):
        vecinos.append((row+1,col))
    elif not IsWall((row,col-1)):
        vecinos.append((row,col-1))
    elif not IsWall((row,col+1)):
        vecinos.append((row,col+1))
    return vecinos
```

A.17. *TranslatePath*

```
TranslatePath(camino):
    for i in range(legth(path)-1):
        if path[i][0] < path[i+1][0]:
            direcciones.append('D')
        elif path[i][0] > path[i+1][0]:
            direcciones.append('U')
        elif path[i][1] < path[i+1][1]:
            direcciones.append('R')
        elif path[i][1] > path[i+1][1]:
            direcciones.append('L')
    return direcciones
```

A.18. *AStarSearch*

```
AStarSearch(start, goal):
    if start == goal:
        return NULL
    queue = PriorityQueue()
    queue.put(start, 0)
    anterior[start] = NULL
    coste[start] = 0
    while NoEsVacio(queue):
        actual = queue.get()
        if actual == goal:
            break
        vecinos = GetNeighbors(actual)
        for v in vecinos:
            nuevoCoste = coste[actual] + 1
            if v not in coste or nuevoCoste < coste[v]:
                coste[v] = nuevoCoste
                heuristico = DistanciaManhattan(v, goal)
                prioridad = nuevoCoste + heuristico
                queue.put(v, prioridad)
                anterior[v] = actual
        camino.append(goal)
        p = anterior[goal]
        while p != start:
            camino.InsertarAlComienzo(p)
            p = anterior[p]
        camino.InsertarAlComienzo(start)
        direcciones = TranslatePath(camino)
        return direcciones
```

A.19. *DecideSpeed*, versión 9

```
DecideSpeed(nivel, fantasmas):
    if PacmanEstaAlineado():
        if IsNewTile():
            sonido = nivel.GetTileSound(fila, columna, fantasmas)
            if teleportado:
                if SigueEnTunel():
                    SeguirAdelante()
                else:
                    teleportado = False
                    IrEnCualquierDireccion()
            else:
                if sonido < sonidoAnterior and temporizador <= 0 and sonido < 5:
```

```
                    ReiniciarCamino()
                    ReiniciarCaminoCompañero()
                    if PuedeVolverAtras():
                        IrEnDireccionContraria()
                    else:
                        IrEnOtraDireccion()
                else:
                    if sonido < 5 and temporizador <= 0:
                        ReiniciarCamino()
                        ReiniciarCaminoCompañero()
                        ff = flowfield[fila][columna]
                        if NoEsDireccionContraria(ff):
                            IrEnDireccion(ff)
                        else:
                            IrEnCualquierDireccion()
                    else:
                        if compañeroMuerto:
                            ReiniciarCamino()
                            if EsVacio(caminoCompañero):
                                caminoCompañero = nivel.AStarSearch((fila,columna, posicionCompañ
    ero)
                            if NoEsVacio(caminoCompañero):
                                d = caminoCompañero.pop()
                                if NoEsDireccionContraria(d):
                                    IrEnDireccion(d)
                                else:
                                    ReiniciarCaminoCompañero()
                                    IrEnCualquierDireccion()
                            else:
                                IrEnCualquierDireccion()
                        else:
                            if ComidaAdyacente():
                                ReiniciarCamino()
                                IrHaciaComida()
                            else:
                                if EsVacio(camino):
                                    camino = FindNearestFood(nivel)
                                if NoEsVacio(camino):
                                    d = camino.pop()
                                    if NoEsDireccionContraria(d):
                                        IrEnDireccion(d)
                                    else:
                                        ReiniciarCamino()
                                        IrEnCualquierDireccion()
                                else:
                                    IrEnCualquierDireccion()
```

A.20. *DecideSpeed*, versión 10

```
DecideSpeed(nivel, fantasmas):
    if PacmanEstaAlineado():
        if IsNewTile():
            sonido = nivel.GetTileSound(fila, columna, fantasmas)
            if teleportado:
                if SigueEnTunel():
                    SeguirAdelante()
                else:
                    teleportado = False
                    IrEnCualquierDireccion()
            else:
                if sonido < sonidoAnterior and temporizador <= 0 and sonido < 5:
                    ReiniciarCamino()
                    if PuedeVolverAtras():
                        IrEnDireccionContraria()
                    else:
                        IrEnOtraDireccion()
                else:
                    if sonido < 5 and temporizador <= 0:
                        ReiniciarCamino()
                        ff = flowfield[fila][columna]
                        if NoEsDireccionContraria(ff):
                            IrEnDireccion(ff)
                        else:
                            IrEnCualquierDireccion()
                    else:
                        if ComidaAdyacente():
                            ReiniciarCamino()
                            IrHaciaComida()
                        else:
                            if EsVacio(camino):
                                camino = FindNearestFood(nivel)
                            if NoEsVacio(camino):
                                d = camino.pop()
                                if NoEsDireccionContraria(d):
                                    IrEnDireccion(d)
                                else:
                                    ReiniciarCamino()
                                    IrEnCualquierDireccion()
                            else:
                                if compañeroMuerto:
                                    if EsVacio(caminoCompañero):
                                        caminoCompañero = nivel.AStarSearch((fila,columna,
    posicionCompañero)
                                    if NoEsVacio(caminoCompañero):
                                        d = caminoCompañero.pop()
                                        if NoEsDireccionContraria(d):
                                            IrEnDireccion(d)
```

```
48                                          else:
49                                              ReiniciarCaminoCompañero()
50                                              IrEnCualquierDireccion()
51                                      else:
52                                          IrEnCualquierDireccion()
53                                  else:
54                                      IrEnCualquierDireccion()
```

Bibliografía

[Cui and Shi, 2011a] Cui, X. and Shi, H. (2011a). A*-based pathfinding in modern computer games. *IJAIA, Vol.11, No.1*.

[Cui and Shi, 2011b] Cui, X. and Shi, H. (2011b). Direction oriented pathfinding in video games. *IJAIA, Vol.2, No.4*.

[Millington and Funge, 2009] Millington, I. and Funge, J. (2009). *Artificial Intelligence for Games*. CRC Press.