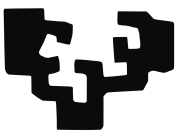
eman ta zabal zazu



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: Ghosts Vs. Pac-Man

---

Autor

*Ibai Baglietto Arakistain*


2018

# Resumen

En este documento trataremos la inteligencia artificial en los juegos, centrándonos en la inteligencia de los fantasmas del juego *Pac-Man*. Para llevar a cabo este trabajo hemos colaborado dos personas, Xabier Garmendia, que ha trabajo con la inteligencia del *Pac-Man*, y yo, Ibai Baglietto, que he trabajado con los fantasmas. El objetivo del proyecto sera definir diferentes inteligencias a los agentes para conseguir el comportamiento más inteligente posible, llevando a cabo una competición entre los dos agentes del juego. Para ver la eficacia de las inteligencias contaremos el número de muertes del *Pac-Man* y el tiempo necesario antes de terminar el nivel.

# Índice general

# Índice de figuras

# Indice de tablas

# 1. CAPÍTULO

# Introducción

## 1.1. Contexto

El mundo del videojuego siempre ha estado marcado por la inteligencia artificial. Ya desde el conocidísimo *Asteroids* (Fig. 1.1) nos enfrentarnos a dos platillos volantes con inteligencias diferentes. El más grande dispara aleatoriamente y el más pequeño dispara al jugador.

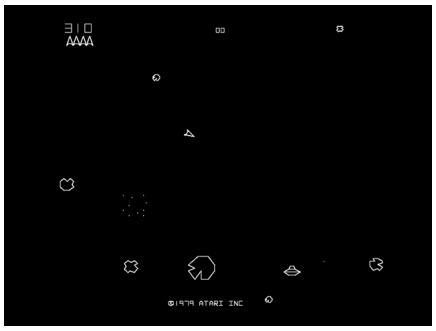



**Figura 1.1:** Captura del juego *Asteroids*.

En un mercado gobernado por las recreativas, la empresa japonesa Namco buscaba trabajadores y encontraron a Toru Iwatani, un joven de 22 años interesado en hacer *pinballs*. Como la empresa no se dedicaba a ello intento levar el *pinball* al videojuego sin mucho éxito con juegos como *Gee Bee* y *Cutie Q* (Fig. 1.2).

Toru Iwatani se dio cuenta de que el mercado del videojuego estaba muy centrado en

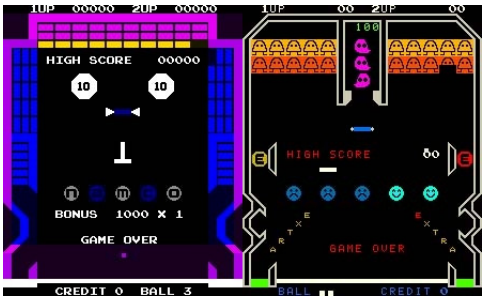



**Figura 1.2:** Captura de los juegos *Geebee*, izquierda, y CutieQ, derecha.

los disparos y en la guerras y decidió intentar hacer un juego para todo el publico. Para ello se basó en un cuento japonés en el que una criatura se come a los monstruos para proteger a los niños. Teniendo eso como base, decidió basar el juego entero en comer, usándolo hasta para el nombre del juego, que en Japón fue *Puck-Man* (Fig. 1.3 ), basado en la onomatopeya de comer japonesa, *paku-paku*. El nombre final en el resto del mundo fue cambiado a *Pac-Man* para que los vándalos no cambiaran la P de *Puck-Man* por F.


**Figura 1.3:** Recreativa del *Puck-Man*.

## 1.2. Inteligencia original de los fantasmas

Para la inteligencia del juego Toru Iwatani decidió que cada uno de los fantasmas tuviera personalidad propia y bien definida, solo teniendo en común que cada cierto tiempo dejaran de perseguir al jugador para irse a una esquina (Fig. 1.4).

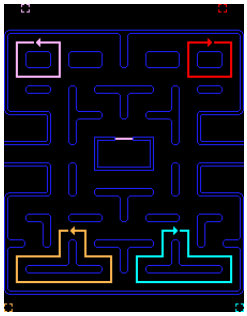


**Figura 1.4:** Recreativa del *scatter*.

El primer fantasma del que vamos a hablar es el rojo, se refiere a el como 追いかけ, *oikake*, el perseguidor. Su inteligencia es la más simple de todas, pero es el que suele atrapar al jugador. Su comportamiento se basa en perseguir al jugador todo el rato menos cuando se tiene que ir a su esquina (Fig. 1.5). Además de esto, cuando el jugador consiga comer un número de puntos determinado su velocidad aumentara un 5 %, y ya cuando falten unos pocos puntos aumentara su velocidad otro 5 %. Cuando se activan estos aumentos de velocidad también se aplica un cambio importante al comportamiento del fantasma, el cual dejara de ir a la esquina cuando lo hagan los demás y seguirá persiguiendo al *Pac-Man*.

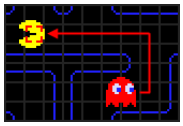


**Figura 1.5:** Comportamiento del fantasma rojo.

El segundo fantasma que vamos a analizar es el rosa, se refiere a el como 待ち伏せ, *machibuse*, el emboscador. Como bien dice su nombre, el objetivo de este fantasma es emboscar al *Pac-Man*. Su inteligencia se basa en ir a la casilla que se encuentra cuatro casillas delante del *Pac-Man* (Fig. 1.6). Por esto, cuando el fantasma esta cara a cara con el *Pac-Man* su objetivo estará detrás suyo, lo que hará que coja el primer desvío que encuentre, alejándose del *Pac-Man*.

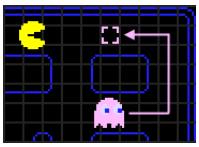


**Figura 1.6:** Comportamiento del fantasma rosa.

En tercer lugar hablaremos del fantasma azul, se refiere a el como 気紛れ, *kimagure*, el caprichoso. Su inteligencia es la mas compleja de todas, ya que es la única que no usa solo la posicion del *Pac-Man*. Para calcular a que casilla tiene que ir usa las posiciones del fantasma rojo y del *Pac-Man*. Para decidir la casilla objetivo primero calcula la casilla que esta dos casillas por delante del *Pac-Man* y después calcula una línea recta entre el fantasma rojo y esa casilla. Finalmente la casilla a la que se dirigirá el fantasma azul será la que esté en la misma línea recta antes calculada pero al doble de distancia (Fig. 1.7).

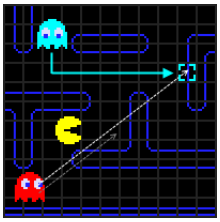


**Figura 1.7:** Comportamiento del fantasma azul.

En último lugar, pero no por eso menos importante, hablaremos del fantasma naranja, se refiere a el como お惚け, *otoboke*, el que finge ignorancia. Su comportamiento variará segun a la distancia que esté del *Pac-Man*, si está mas lejos de ocho casillas siguiéndolo como el fantasma rojo y si esta más cerca desplazándose hacia su esquina. Por esto, si el jugador conoce este funcionamiento puede hacer que de vueltas en el mismo sitio todo el rato, como se ve en la imagen 1.8, están marcadas con una *X* las casillas en las que el fantasma cambia de comportamiento.

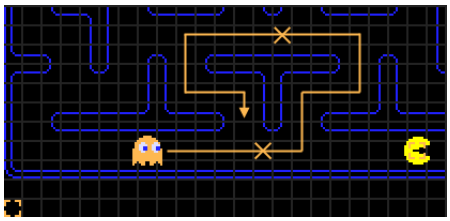


**Figura 1.8:** Comportamiento del fantasma naranja.

## 1.3. Orientación del proyecto

Para la realización de este proyecto, he colaborado con un compañero del grado, Xabier Garmendia, 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 del *Pac-Man*. 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.

## 1.4. Inteligencia artificial

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 los fantasmas el movimiento y la toma de decisiones irán de la mano, ya que la única decisión que pueden tomar es hacia donde moverse. En el caso de la estrategia esta variará dependiendo de la inteligencia artificial aplicada, pudiendo ser desde acercarse hacia la presa hasta mantenerse en una zona con comida para que la presa no pueda comérsela.

# 2. CAPÍTULO

# Documento de los objetivos del proyecto

## 2.1. Objetivos del proyecto

El proyecto se divide en dos apartados: programación y análisis. En la primera parte nos basaremos en el *Pac-Man* hecho para Python 2.7 por David Reilly. Teniendo en cuenta que Xabier Garmendia hará la inteligencia del *Pac-Man* también utilizaremos su código para probar la inteligencia de nuestros fantasmas. Los objetivos principales de esta parte son los siguientes:

- Trabajar con la comunicación entre fantasmas.
- Hacer que el comportamiento de los fantasmas sea lo más realista posible, por ejemplo usando ejemplos de la naturaleza.
- Usar el máximo número de sentidos para conseguir nuestro objetivo.
- Atrapar a la presa el máximo número de veces.

En la segunda parte analizaremos los datos conseguidos con las inteligencias ya creadas, centrándonos en los fantasmas. Estos son los principales objetivos:

- Analizar el comportamiento de los fantasmas sin que trabajen en equipo.
- Observar cómo actúan los cazadores según las diferentes versiones de la presa.

- Analizar si el comportamiento de los fantasmas es realista según los datos que reciben.

- Realizar diferentes pruebas con cada versión de los fantasmas, por ejemplo ver cuantas veces son capaces de atrapar a la presa antes de que coma toda la comida del mapa o ver cuantas veces lo pueden atrapar en un tiempo determinado.

## 2.2. 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 2.1 se puede ver una imagen del juego.

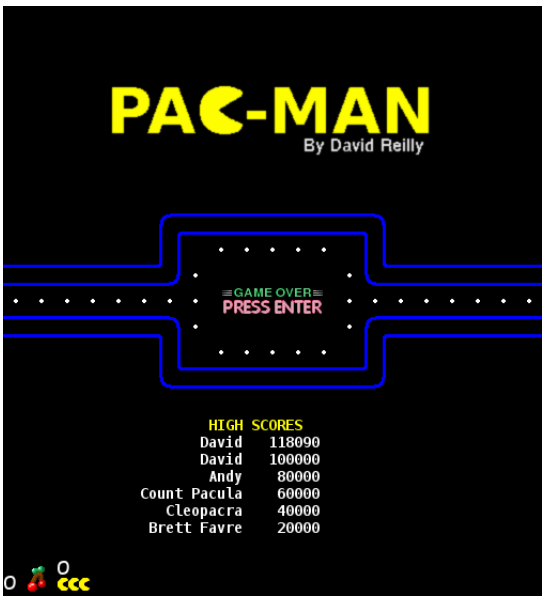



**Figura 2.1:** Pacman, por David Reilly

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*.

## 2.3. Planificación

Vamos a dividir el proyecto en 5 partes bien diferenciadas: apredizaje, diseño e implementación de la versión modificada, diseño e implementación de las versiones de la IA, diseño y realización de las pruebas y redacción de la memoria. En el diagrama de la figura 2.2 podemos ver el número de días que le dedicaremos a cada parte. Tenemos que tener en cuenta que desde enero hasta finales de abril también se tendrá que trabajar con las asignaturas del curso, por lo que no podremos ofrecerle tanto tiempo al proyecto como en mayo y junio.

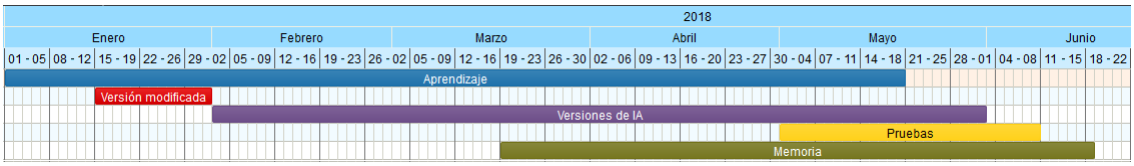



**Figura 2.2:** Diagrama Gantt con la planificación del proyecto

Ahora analizaremos cada parte del proyecto, viendo el trabajo que supone cada uno:

### 2.3.1. Aprendizaje

El aprendizaje será parte fundamental del proyecto, ya que tendremos que informarnos en diferentes lugares para intentar conseguir el mejor resultado posible. Esta parte del proyecto sera también la mas longeva, ya que tendremos que empezar con el aprendizaje antes de empezar con todo lo demás y continuaremos con ello hasta mediados de mayo.

### 2.3.2. Diseño e implementación de la versión modificada

Como ya se ha mencionado antes, para hacer este proyecto nos vamos a basar en una versión del *Pac-Man* de David Reilly. Esta versión esta pensada para que juegue una persona contra la máquina y tiene una tabla de puntuaciones máximas incorporada. Además, el código está agrupado en pocos archivos, haciendo el código más difícil de entender a primera vista.

Para facilitar la comprensión, decidimos que era mejor dividir el código en cinco archivos: main, level, game, ghost y pacman. En cada uno de estos archivos se trabajará con partes especificas del programa, como bien indica cada nombre.

### 2.3.3. Diseño e implementación de las versiones de la IA

Una vez acomodado el código a cinco archivos se empezará con la programación de la nueva inteligencia. Los archivos ghost y pacman antes nombrados tendrán varias versiones, guardando en cada archivo una versión de la inteligencia creada. También se crearán dos niveles nuevos, intentando que sean los más diferentes posibles entre ellas. Para acceder a estas versiones se creará un selector de modo de juego, un selector de versión de *Pac-Man* y fantasma y un selector de nivel.

### 2.3.4. Diseño y realización de las pruebas

Cuando una gran parte de las versiones de la inteligencia artificial estén terminadas se empezará con las pruebas a modo de competición dónde probaremos todas las versiones, buscando la que mejor funcione contra todas las demás. Para medir la calidad de cada versión se contarán las muertes del *Pac-Man* y el tiempo que necesita para completar el nivel.

### 2.3.5. Redacción de la memoria

En esta última parte analizaremos todo lo hecho hasta el momento, reuniéndolo en un documento y documentando todo el proceso de creación del proyecto.

## 2.4. Análisis de riesgos

Hay dos peligros principales en la realización de este proyecto:

1. Uso del tiempo durante el curso: se han planificado bastantes horas de trabajo durante el curso sin saber exactamente la carga de trabajo que darán las asignaturas. Por ello se ha tenido en cuenta que el tiempo disponible será menor que cuando se acabe el curso.

2. Pérdida del trabajo: si el trabajo se guarda en un único sitio es posible perderlo todo. Para que esto no pase el código se guardará en dos ordenadores y en una carpeta de Google Drive y el documento se guardará en un ordenador y en un pendrive.

## 2.5. Análisis de factibilidad

Teniendo en cuenta que los mayores peligros están cubiertos vemos que este proyecto es factible. Además, se ha tenido en cuenta que pueden surgir imprevistos al hacer la planificación, por lo que se han planificado horas de más para cada apartado.

# 3. CAPÍTULO

# Desarrollo del proyecto

Se han implementado dos modos de juego para los que se han creado varias versiones. En el primer modo habrá dos cazadores y una presa y en el segundo tres cazadores y dos presas. Primero se explicarán las partes comunes de todas las versiones, después se explicará el funcionamiento e cada versión para el primer modo de juego y por último se explicarán las versiones del segundo modo de juego.

## 3.1. Partes comunes de todas las versiones

Para desarrollar todas las versiones se han seguido las mismas bases, que iremos analizando una a una.

### 3.1.1. Inicio

Al ejecutar el programa se le pedirá al usuario que elija las versiones de los cazadores y la presa y luego se abrirá la ventana de inicio (Fig. 3.1), en la que se deberá pulsar el botón *enter* para comenzar el juego.

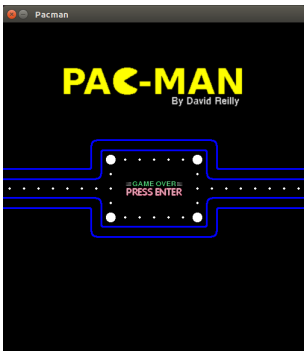



**Figura 3.1:** Ventana de inicio.

### 3.1.2. Mapas

Los mapas utilizados son laberintos de 21x25 casillas. Por todo el mapa habrá comida, que es el objetivo de la presa. Para facilitarle el trabajo podrá usar 4 teletransportes que le llevarán a la otra parte del mapa, si se mete por la derecha saldrá a la izquierda y viceversa y si se mete por arriba saldrá abajo y viceversa. La posición inicial de la presa y los cazadores también será siempre la misma, la que se puede ver en la figura 3.2.

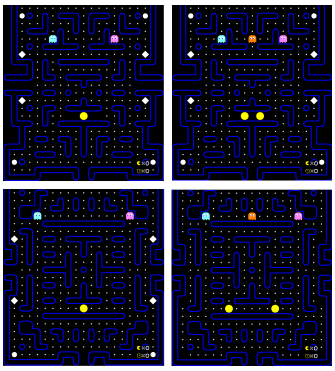


**Figura 3.2:** Los mapas en las posiciones iniciales en las dos versiones.

### 3.1.3. Rastros

Los cazadores y la presa dejarán diferentes rastros para que sus respectivos contrincantes puedan saber por donde se están moviendo.

### Olor

La presa dejará un rastro de olor en las últimas 10 casillas por las que haya pasado, dejando marcada la posición hacia la que se ha desplazado. Podemos ver la representación gráfica del olor en verde en la figura 3.3.

**Figura 3.3:** Rastro de olor de la presa.

He contribuido a la implementación de la presa con la rutina de generación de olor, ya que será utilizado por los cazadores para poder cazarla. En el código A.1 se puede ver cómo se ha hecho.

Como vemos en el pseudocódigo, guardaremos en la lista con las casillas que tienen olor las últimas 10 casillas por las que haya pasado la presa. Cada vez que la presa cambie de casilla se mirará el estado de la lista y pasaremos todos las posiciones de la lista hacia atrás para hacerle un hueco al principio a la nueva casilla. En las posiciones de la lista se guardará la casilla y la orientación en la que se ha movido la presa, para que el cazador pueda seguirla.

### Ruido

La cazador hará ruido al moverse, lo que alertará a la presa de su proximidad. Podemos ver la representación gráfica del ruido en rojo en la figura 3.4

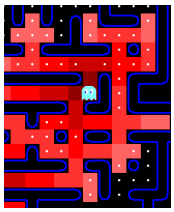


**Figura 3.4:** Ruido del cazador.

#### Visión

La presa será visible para los cazadores siempre que no haya paredes entre los dos. No se representará visualmente, ya que a simple vista sabemos en que casillas debería ser visible. Como con el olor, esta parte también ha sido implementada por mi, ya que esta función será usada por los cazadores. Se puede ver la implementación en el código A.2.

Como podemos apreciar, la implementación es bastante simple, lo único que se hará será guardar todas la casillas desde las que se pueda ver a la presa en una lista, mirando en todas las direcciones y guardando todas las casillas hasta que encontremos una pared.

### 3.1.4. Posiciones finales

El juego puede terminar de dos formas diferentes: la presa puede ser cazada (Fig. 3.5, izquierda), haciendo que los cazadores y la presa vuelvan a las posiciones iniciales pero sin hacer que la comida que ya se ha comido reaparezca, y la presa puede comer toda la comida del mapa (Fig. 3.5, derecha), lo que hará que se vuelva a abrir la ventana de inicio (Fig. 3.1), dándole la opción de volver a ejecutar el programa al usuario.

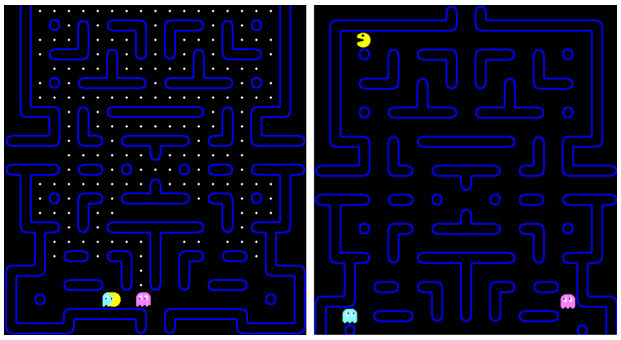


**Figura 3.5:** Derecha: la presa ha comido toda la comida. Izquierda: la presa ha sido cazada.

### 3.1.5. Interfaz

Se ha añadido una interfaz para que el usuario sepa cuantas veces han comido los cazadores a la presa y cuanto tiempo lleva la partida en marcha, como podemos ver en la figura 3.6.

**Figura 3.6:** Interfaz del juego.

### 3.1.6. Selector

Se ha creado un selector para que el usuario pueda decidir que mapa quiere usar, que modo de juego quiere ejecutar y que versiones del *Pac-Man* y de los fantasmas quiere que compitan, como se puede ver en la figura 3.7.


**Figura 3.7:** El selector con todas las opciones disponibles.

### 3.1.7. Medida de calidad

Con todas las versiones que se lleven a cabo se medirá la calidad respecto a las versiones de la presa. Para esto se analizará el tiempo y las vidas que necesite la presa para conseguir comerse toda la comida del mapa. Cada vez que los cazadores atrapen a la presa se reiniciarán las posiciones de los agentes, pero la comida no reaparecerá.

Para llevar a cabo las pruebas se van a hacer dos tablas por cada versión de los cazadores. Se hará una tabla para cada mapa, contando en cada una el número de muertes necesarias y el número de segundos necesarios para completar el mapa. También se dibujará la tabla de forma gráfica, usando las medianas de tiempo y muertes de cada versión.

Las ejecuciones se han hecho a la máxima velocidad permitida por el ordenador, quitando en el código la parte dónde se obliga que sean 60 ejecuciones por segundo. Como solo queremos comparar las versiones cuando están en el mismo nivel, cada nivel se ha ejecutado en un ordenador diferente, agilizando el trabajo [1].

[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.

## 3.2. Funcionamiento básico de las versiones de la presa del primer modo

Dado que los cazadores tienen que atrapar a la presa el máximo número de veces, vamos a analizar brevemente el funcionamiento de todas las versiones. Analizando el funcionamiento y los mayores peligros para los cazadores.

### 3.2.1. Versión 1.0

#### Funcionamiento

Al ser la primera versión su funcionamiento es bastante simple, se limita a escapar. Cuando detecta el ruido de los fantasmas intentará huir en la otra dirección como podemos ver en la figura 3.8.

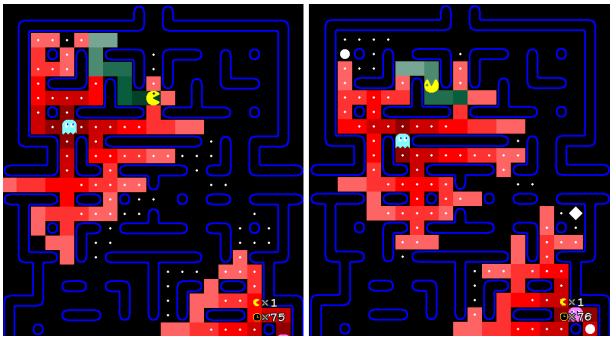


**Figura 3.8:** La presa detecta el ruido del cazador y se da la vuelta.

#### Peligros

Al tratarse de una versión tan simple no conlleva grandes peligros para los cazadores, excepto por su gran aleatoriedad. Cuando no escucha ningún ruido se mueve de forma aleatoria por el mapa, lo que hace que vaya comiendose toda la comida del mapa o que se quede dando vueltas en una esquina, sin que los cazadores puedan predecir sus movimientos.

### 3.2.2. Versión 1.1

#### Funcionamiento

En esta segunda versión la presa empezará a preocuparse por la comida, intentando comerla siempre que no este amenazada por los cazadores. La presa solo podra detectar la comida que esté a diez casillas de distancia como máximo, acercándose a la comida que antes encuentre.

#### Peligros

Como la presa va hacia la comida la partida terminará rápido si los cazadores no hacen nada para impedirlo. En el caso de que los cazadores no protejan la comida puede que la presa se la coma toda sin tener un solo encuentro con los cazadores.

### 3.2.3. Versión 1.2

#### Funcionamiento

En esta versión la presa empezará a aprovechar los teletransportes para intentar que los cazadores le pierdan el rastro. Cuando se vea amenazado se dirigirá al teletransporte más cercano, como podemos ver en la figura 3.9.

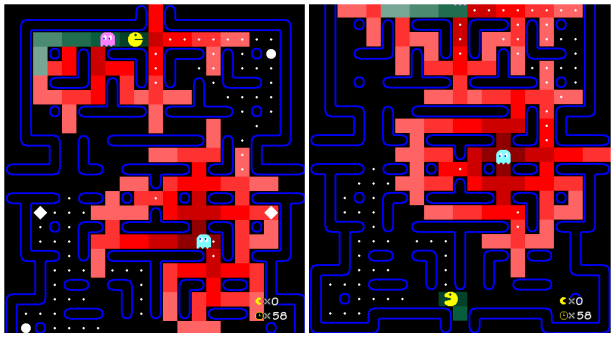


**Figura 3.9:** La presa se dirige al teletransporte y sale por el otro lado.

#### Peligros

Al aprovecharse de que los cazadores no pueden usar los teletransportes la presa puede conseguir que los cazadores le pierdan el rastro, siempre que estos no sepan donde se encuentra la salida de los teletransportes.

### 3.2.4. Versión 1.3

#### Funcionamiento

En esta versión empezarán a usarse las *Power-Pellets*, que hacen que los cazadores pasen a modo vulnerable. Mientras se mantengan en este estado la presa podrá comerse a los cazadores, haciendo que regresen a la posición de salida 3.10.

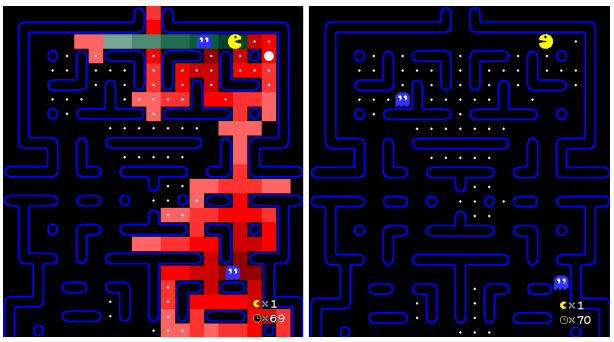


**Figura 3.10:** Izquierda: los cazadores están en modo vulnerable. Derecha: La presa se come a un cazador y este vuelve a su posición de salida.

Como la presa no puede ser cazada mientras que el modo vulnerable este activo, ignorará completamente a los cazadores, sabiendo que no le pueden hacer nada.

#### Peligros

Dado que cuando los cazadores son comido regresan a su posición original, puede que de estar cerca de la presa se les mande al otro lado del mapa si son comidos, perdiendo así mucho tiempo. Si esto pasara se le daría mucha ventaja a la presa para comer gran parte de la comida.

### 3.2.5. Versión 1.4

#### Funcionamiento

En esta nueva versión la presa empezará a acercarse más a los cazadores, ya que antes se alejaba cuando los cazadores estaban demasiado lejos, dejando de comer comida que estaba a su alcance.

#### Peligros

Al hacer que la presa se acerque más a los cazadores a simple vista parece que será capturada más veces, pero esto no tiene por que ser así. De hecho, si los cazadores están defendiendo una zona concreta puede que la presa se acerque a comer toda la comida de los alrededores dejando solo la zona en la que se encuentra el cazador, sin que este llegue a ver nada.

### 3.2.6. Versión 1.5

#### Funcionamiento

En esta última versión la presa intentará usar de mejor forma los teletransportes, haciendo que si toma uno de estos no se de la vuelta, para que no le atrape el cazador que le estaba persiguiendo.

#### Peligros

Esta versión no supone un gran peligro para los cazadores, lo único que hace es arreglar un error imprevisto de la presa.

## 3.3. Versión 1.0

El objetivo de esta versión será ver cual es el rendimiento de los fantasmas trabajando por separado con todas las versiones de la presa.

### 3.3.1. Influencias

Para hacer este primer comportamiento nos hemos basado en la inteligencia del fantasma rojo, el perseguidor. Como queremos que los comportamientos de los cazadores sean realistas respecto a su conocimiento del entorno, solo activarán este comportamiento cuando puedan saber la posición de la presa de alguna forma visual u olfativa.

### 3.3.2. Inteligencia

Esta inteligencia será la mas básica de todas las que se van a implementar, siendo el objetivo de esta perseguir a la presa sin ningún tipo de trabajo en equipo. Los cazadores tendrán dos tipos de comportamiento: movimiento aleatorio cuando no vean a la presa y persecución cuando la vean o la huelan. Teniendo en cuenta que los cazadores se mueven a la misma velocidad que la presa sera imposible para ellos atraparlo por separado, haciendo que las muertes de la presa sean pura casualidad, cuando cada uno de los cazadores se lo encuentre de frente mientras el otro lo persigue de antes, trabajando en equipo sin saberlo. En la figura 3.11 podemos ver el diagrama de flujos de esta primera versión.

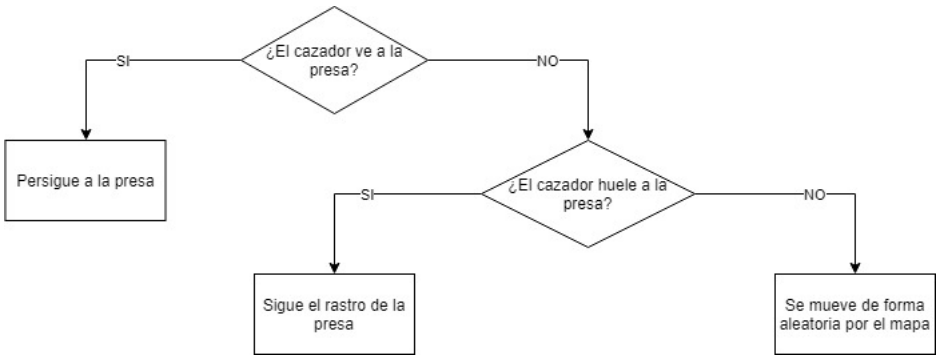



**Figura 3.11:** Diagrama de flujos de la versión 1.0.

### 3.3.3. Implementación

Para implementar esta inteligencia no hemos tenido que hacer muchas cosas nuevas, ya que la implementación del olor y la vista ya ha sido hecha antes. Para hacer que los cazadores persigan a la presa solo tendremos que usar las funciones anteriormente creadas, como se puede ver en el código A.3, que se ha hecho en la función *Move* del archivo *ghostV0*.

Si analizamos el pseudocódigo, podemos ver que la prioridad la tendrá la visión, luego el olfato y en último lugar el movimiento aleatorio. Esto se debe a que la visión siempre tendrá el camino más corto hacia la presa, mientras que con el olfato puede que se de una vuelta más larga para llegar al mismo sitio. También podemos ver que cuando los cazadores se mueven de forma aleatoria no se les permite darse la vuelta, para que no se queden siempre en el mismo pasillo.

### 3.3.4. Pruebas

Vamos a probar la efectividad de esta primera inteligencia con todas las versiones de la presa. Como se ha mencionado antes contaremos con dos tablas y dos gráficos: una tabla y un gráfico para el primer mapa, tabla 3.1 y figura 3.12, y lo mismo para el segundo, tabla 3.2 y figura 3.13.

**Tabla 3.1:** 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** | 31 | 1 | 26 | 1 | 24 | 1 | 34 | 1 | 28 | 2 | 25 | 1 | 35 | 3 | 35 | 3 | 35 | 3 | 23 | 1 |
| **V2** | 29 | 2 | 32 | 0 | 32 | 0 | 32 | 0 | 32 | 0 | 56 | 5 | 36 | 0 | 39 | 4 | 43 | 2 | 41 | 1 |
| **V3** | 29 | 3 | 27 | 0 | 57 | 2 | 24 | 0 | 41 | 2 | 32 | 0 | 54 | 2 | 34 | 1 | 37 | 1 | 24 | 0 |
| **V4** | 28 | 2 | 28 | 0 | 29 | 0 | 38 | 3 | 37 | 4 | 24 | 0 | 26 | 0 | 30 | 1 | 44 | 2 | 30 | 3 |
| **V5** | 28 | 2 | 25 | 2 | 35 | 3 | 27 | 2 | 29 | 1 | 31 | 1 | 31 | 1 | 50 | 3 | 35 | 0 | 33 | 3 |

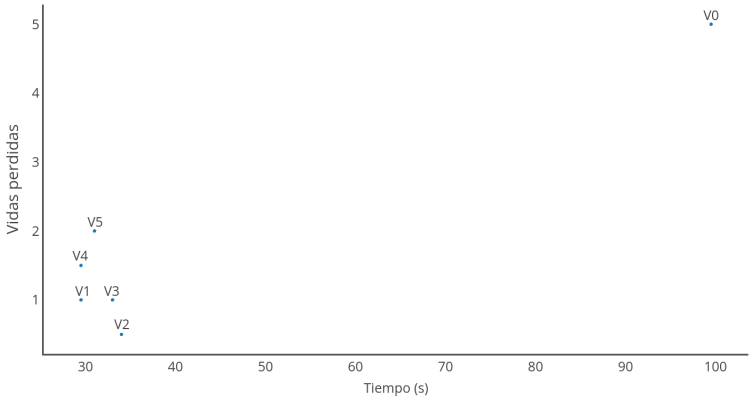



**Figura 3.12:** Gráfico con las medianas de la tabla 3.1.

**Tabla 3.2:** 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** | 19 | 0 | 16 | 0 | 27 | 1 | 25 | 3 | 21 | 0 | 20 | 1 | 25 | 2 | 33 | 1 | 29 | 1 | 32 | 2 |
| **V2** | 22 | 0 | 25 | 1 | 36 | 3 | 24 | 1 | 24 | 1 | 24 | 0 | 19 | 0 | 24 | 3 | 26 | 2 | 28 | 2 |
| **V3** | 26 | 0 | 52 | 4 | 23 | 0 | 24 | 0 | 26 | 2 | 22 | 1 | 25 | 2 | 34 | 0 | 26 | 4 | 25 | 1 |
| **V4** | 23 | 2 | 24 | 0 | 19 | 0 | 19 | 0 | 23 | 1 | 22 | 2 | 28 | 1 | 21 | 1 | 29 | 4 | 27 | 2 |
| **V5** | 31 | 3 | 23 | 1 | 22 | 2 | 26 | 1 | 31 | 2 | 23 | 3 | 29 | 2 | 21 | 0 | 24 | 1 | 23 | 1 |

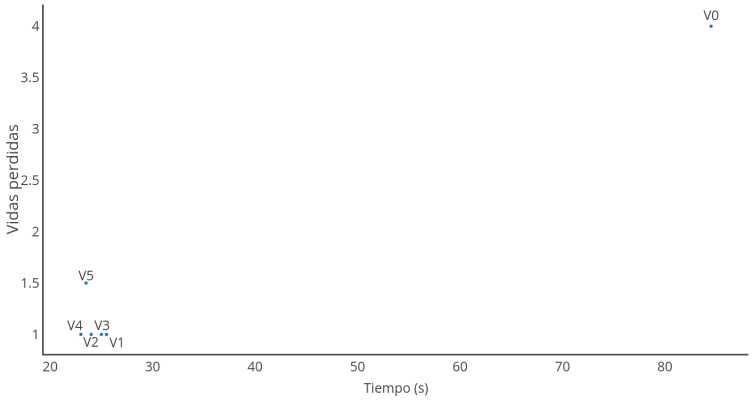



**Figura 3.13:** Gráfico con las medianas de la tabla 3.2.

### 3.3.5. Conclusiones

Como podemos apreciar en las pruebas, el número de veces que la presa es atrapada y el tiempo que necesita para completar el mapa varía mucho en todas las pruebas. Esto sucede por la falta de trabajo en equipo y el movimiento aleatorio que hacen los cazadores cuando no tienen localizada a la presa. Como ese movimiento es totalmente aleatorio pueden estar cortándole el paso a la presa o pueden estar alejándose de ella mientras su compañero la persigue sin ser capaz de atraparla.

### 3.3.6. A mejorar

Este primer comportamiento tiene muchos cosas a mejorar. Primero, si los dos cazadores se ponen a perseguir a la presa (Fig. 3.14) no tendrán ninguna opción de atraparla, ya que los dos seguirán el mismo camino a la misma velocidad que la presa. Por otra parte, puede que uno de los dos cazadores tenga localizada a la presa y el otro este dando vueltas en la esquina de la otra punta del mapa, sin intención de acercarse. Para poder arreglar estos dos problemas vamos a implementar comunicación entre los cazadores, lo que les servirá para poder cortarle el paso a la presa mientras el otro la persigue.

**Figura 3.14:** Los dos cazadores persiguen a la presa.

## 3.4. Versión 1.1

El objetivo de esta versión será ver cual es el rendimiento de los cazadores si trabajan en equipo, probándolo con todas las versiones de la presa.

### 3.4.1. Influencias

Para crear este comportamiento nos hemos basado en dos de las inteligencias originales del *Pac-Man*, la del fantasma rojo y la del fantasma rosa. Cuando uno de los dos cazadores vea o huela a la presa empezará a perseguirle, como hace el fantasma rojo. Mientras uno la persigue, el otro intentará emboscarla intentando imitar el comportamiento del fantasma rosa.

### 3.4.2. Inteligencia

Como hemos dicho en el anterior apartado, esta versión cuenta con dos inteligencias separadas, el perseguidor y el emboscador, sin tener en cuenta como inteligencia cuando se mueven de forma aleatoria por el mapa si no detectan a la presa. El primero sigue el mismo comportamiento que en la versión 1.0, el cazador perseguirá a la presa si la ve o la huele.

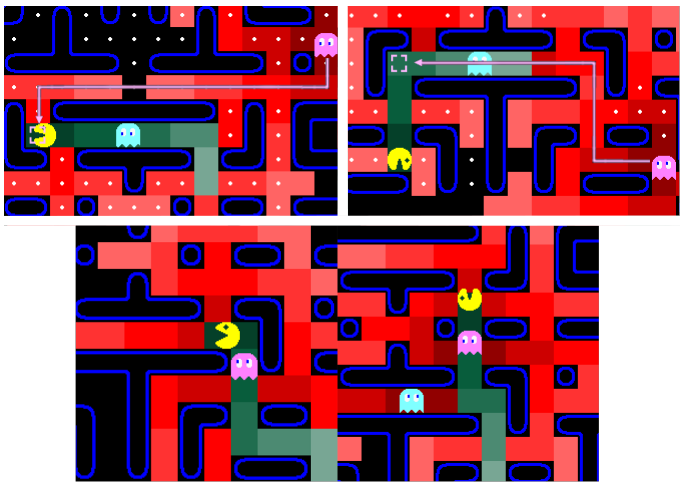


**Figura 3.15:** Arriba a la derecha: el emboscador va directo a la presa, ya que el perseguidor la ve. Arriba a la izquierda: el emboscador va a intentar emboscar a la presa, ya que el perseguidor solo la huele. Abajo: el fantasma azul se separa para intentar emboscar a la presa.

El comportamiento del emboscador variará según si el perseguidor le ve a la presa o solo la huele. En el caso de que la vea el perseguidor sabrá la posición exacta de la presa, siendo capaz de decirle el camino exacto que debe seguir al emboscador (Fig. 3.15, arriba a la derecha).

En el caso de que la huela, solo sabrá hacia que dirección se ha movido, haciendo que solo le pueda pasar una posición aproximada al otro cazador, le manda cuatro casillas hacia adelante desde la casilla en la que esta el perseguidor (Fig. 3.15, arriba a la izquierda). Esta posición puede no ser del todo acertada para atrapar a la presa, pero sirve para que el emboscador se mueva hacia ella.

Por último, si los dos cazadores están haciendo de perseguidor al mismo tiempo, uno de los dos se desviará del camino para poder ser el emboscador, facilitando la captura de la presa (Fig. 3.15, abajo).

Podemos ver el diagrama de flujos de esta versión el la figura 3.16.

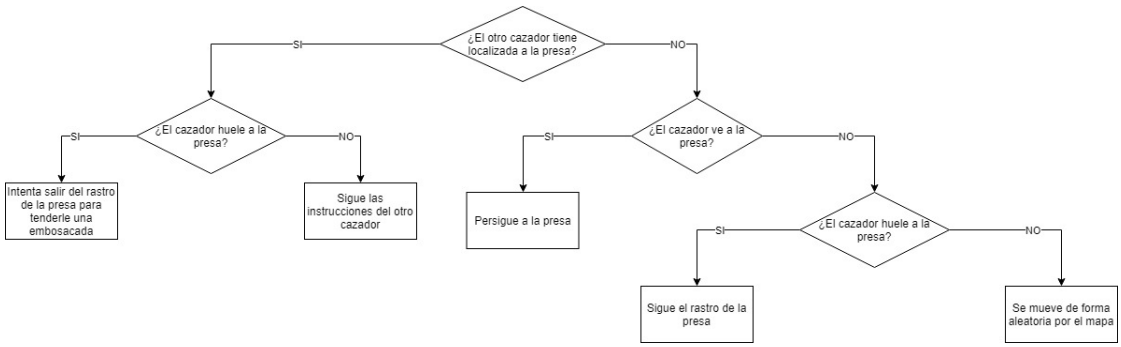



**Figura 3.16:** Diagrama de flujos de la versión 1.1.

### 3.4.3. Implementación

Para implementar el trabajo en equipo se ha tenido que trabajar más en el código que en la versión anterior. Esta vez se ha creado una nueva variable, *comunicado*. Esta se usará para saber si el otro cazador tiene localizada a la presa, haciendo que se puedan comunicar la dirección a seguir. Además también se ha creado la lista ghostmove, en la que se guardará la posición a la que le mande un cazador al otro para perseguir a la presa. Como ya hemos explicado en el anterior apartado, en el caso de que el cazador la vea se pasará la posición exacta de la presa y en el caso de que la huela se pasará la posición cuatro casillas hacia adelante desde la posición del cazador que manda el aviso. En el siguiente pseudocódigo

podemos ver cómo se gestiona la comunicación entre los cazadores, implementado en la función *move* como en la anterior versión pero esta vez en el archivo *ghostV1*. Solo se analizará la versión del primer cazador, ya que la del otro cazador es igual, lo único que cambia es cual de los dos es el que manda el aviso. El código se puede ver en el apéndice A.4.

Como podemos ver, la presa puede estar en 4 direcciones principales, arriba, abajo, derecha, izquierda, pero estas direcciones las podemos dividir también cada una en otras tres para que el movimiento sea más preciso, puede estar exactamente en esa dirección o puede estar en una de las dos diagonales. Como vemos al principio del pseudocódigo, las indicaciones se seguirán si solo el otro cazador sigue el rastro de la presa. Si los dos cazadores están persiguiendo a la presa uno de los dos se desviará del camino, haciendo que luego siga las indicaciones de su compañero.

### 3.4.4. Pruebas

Probemos la eficacia de esta mejorada inteligencia con todas la versiones de la presa. Contaremos con dos tablas y dos gráficos: una tabla y un gráfico para el primer mapa, tabla 3.3 y figura 3.17, y lo mismo para el segundo, tabla 3.4 y figura 3.18.

**Tabla 3.3:** 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** | 167 | 12 | 154 | 12 | 123 | 8 | 227 | 20 | 127 | 3 | 90 | 9 | 161 | 19 | 157 | 12 | 86 | 11 | 131 | 10 |
| **V1** | 29 | 3 | 32 | 2 | 28 | 2 | 26 | 1 | 30 | 3 | 32 | 1 | 25 | 1 | 46 | 4 | 27 | 3 | 29 | 3 |
| **V2** | 34 | 0 | 52 | 2 | 41 | 2 | 35 | 2 | 39 | 2 | 30 | 1 | 31 | 1 | 28 | 0 | 75 | 7 | 25 | 1 |
| **V3** | 43 | 3 | 29 | 1 | 39 | 0 | 44 | 3 | 32 | 0 | 26 | 1 | 36 | 2 | 26 | 0 | 27 | 0 | 27 | 0 |
| **V4** | 27 | 2 | 29 | 0 | 32 | 1 | 29 | 1 | 46 | 2 | 29 | 1 | 31 | 2 | 29 | 1 | 33 | 0 | 28 | 1 |
| **V5** | 43 | 4 | 30 | 2 | 37 | 2 | 36 | 2 | 35 | 1 | 33 | 1 | 36 | 2 | 30 | 1 | 23 | 0 | 22 | 1 |

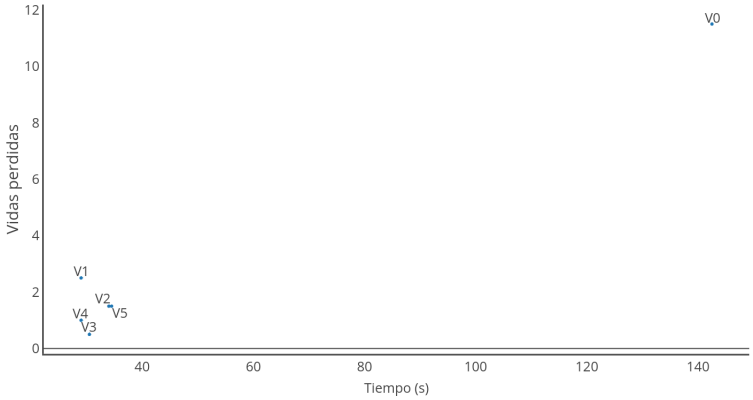



**Figura 3.17:** Gráfico con las medianas de la tabla 3.3.

**Tabla 3.4:** 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** | 69 | 10 | 90 | 9 | 69 | 6 | 72 | 4 | 83 | 6 | 98 | 8 | 66 | 5 | 72 | 6 | 68 | 4 | 89 | 7 |
| **V1** | 27 | 4 | 28 | 2 | 21 | 1 | 29 | 2 | 24 | 3 | 45 | 6 | 27 | 3 | 23 | 0 | 29 | 3 | 28 | 4 |
| **V2** | 27 | 2 | 31 | 2 | 20 | 0 | 27 | 2 | 21 | 0 | 25 | 0 | 27 | 0 | 29 | 0 | 22 | 1 | 35 | 1 |
| **V3** | 21 | 0 | 30 | 3 | 23 | 1 | 28 | 1 | 31 | 0 | 29 | 1 | 27 | 2 | 23 | 2 | 22 | 1 | 25 | 3 |
| **V4** | 27 | 1 | 23 | 1 | 25 | 2 | 21 | 0 | 22 | 1 | 22 | 2 | 26 | 1 | 27 | 3 | 23 | 2 | 26 | 3 |
| **V5** | 20 | 0 | 24 | 0 | 27 | 2 | 23 | 0 | 30 | 1 | 31 | 4 | 32 | 3 | 20 | 1 | 23 | 1 | 31 | 1 |

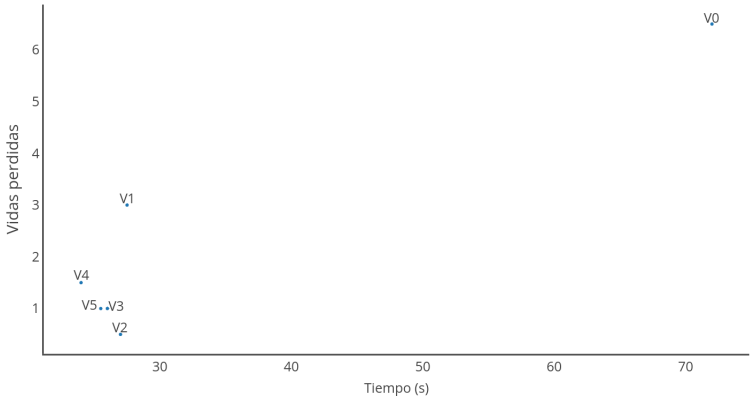



**Figura 3.18:** Gráfico con las medianas de la tabla 3.4.

### 3.4.5. Conclusiones

Como vemos en las pruebas los resultados han mejorado significativamente en la primera prueba respecto a la versión 1.0, aunque por la aleatoriedad de esta versión hay mucha diferencia entre pruebas. Si miramos las demás pruebas, veremos que los cazadores siguen atrapando pocas veces a la presa, aunque la atrapan más veces que en la anterior versión.

### 3.4.6. A mejorar

En esta versión los cazadores solo se centran en intentar atrapar a la presa, sin pensar que la presa intentará comerse la comida del mapa. Para la próxima versión se intentará mejorar este aspecto, dándoles a los cazadores conocimiento sobre la comida.

## 3.5. Versión 1.2

El objetivo de esta versión será mejorar la versión anterior y ver el nivel de la mejora probándolo con todas las versiones de la presa.

### 3.5.1. Influencias

Para crear este comportamiento no nos hemos basado en ninguna inteligencia original del *Pac-Man*, ya que todas ellas funcionaban según la posición actual del agente al que se quiere alcanzar. En vez de mirar solo en las inteligencias del *Pac-Man*, hemos mirado en otros entornos y hemos visto que defender el objetivo que quiere conseguir el rival es bastante efectivo. Esto lo podemos ver en muchos ámbitos, como el deporte, defender la zona de peligro del rival, o la naturaleza, rondar la zona en la que abunda comida para tender una trampa a otros animales.

### 3.5.2. Inteligencia

La mayor parte de la inteligencia es la usada en la anterior versión, solo le sumaremos la percepción de la comida a uno de los cazadores, dejando que el otro se mueva libre por el mapa. Al hacer esto, hemos definido dos nuevos roles a los cazadores: el protector y el buscador. El protector se quedará en una zona con comida esperando a la llegada de la presa, mientras que el buscador dará vueltas por todo el mapa buscando a la presa, por si esta no se acerca lo suficiente al protector (Fig. 3.19).

Cuando uno de los dos cazadores encuentre a la presa seguirán el mismo comportamiento que en la anterior versión, siendo uno el perseguidor y el otro el emboscador.

Podemos ver el diagrama de flujos de esta versión en la figura 3.20.

### 3.5.3. Implementación

Para implementar esta mejora en el trabajo en equipo se ha tenido que crear una nueva función, *FindFood*. Esta función mirará en las casillas que estén a menos de 10 celdas de distancia y guardará las que contengan comida. Después de guardar todas las casillas con comida se calculará una posición media de la comida, a la que se dirigirá el protector.

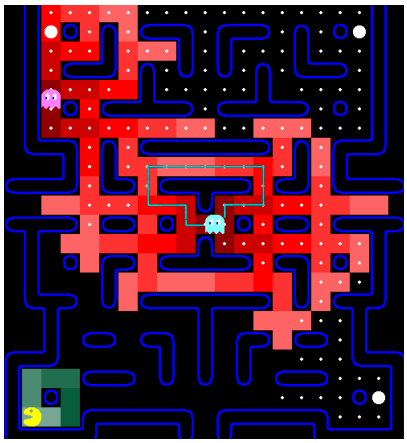


**Figura 3.19:** El protector se queda dando vueltas mientras el buscador explora el mapa.

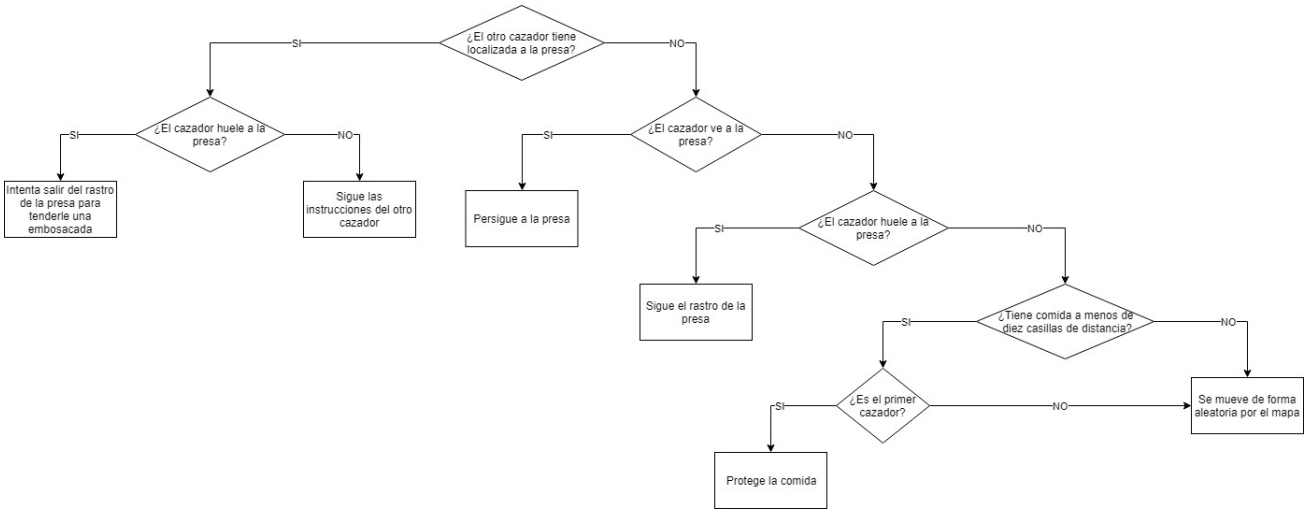



**Figura 3.20:** Diagrama de flujos de la versión 1.2.

Esta función sera usada por solo uno de los dos cazadores, usando casi el mismo código usado en la anterior versión para seguir las instrucciones del otro cazador, solo que esta vez se usará como objetivo el resultado de la función. En los pseudocódigos A.5 y A.6 podemos ver como se ha implementado la función *FinfFood* y como se usa.

Como podemos ver en el pseudocódigo, vamos a mirar en todas las casillas que el cazador tiene a 10 casillas de distancia como máximo, guardando la fila y la columna de las casillas con comida en dos listas. Cuando todas las casillas han sido guardadas calculamos la media y si esta casilla es una pared cogemos una de las casillas adyacentes que no lo sean.

### 3.5.4. Pruebas

Contaremos con dos tablas y dos gráficos como en las demás versiones: una tabla y un gráfico para el primer mapa, tabla 3.5 y figura 3.21, y lo mismo para el segundo, tabla 3.6 y figura 3.22.

**Tabla 3.5:** 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** | 235 | 13 | 276 | 11 | 333 | 24 | 111 | 2 | 232 | 11 | 132 | 4 | 263 | 12 | 237 | 10 | 279 | 15 | 109 | 9 |
| **V1** | 53 | 7 | 34 | 4 | 35 | 4 | 34 | 0 | 28 | 2 | 28 | 0 | 47 | 2 | 46 | 3 | 33 | 2 | 38 | 3 |
| **V2** | 29 | 1 | 78 | 5 | 36 | 1 | 32 | 1 | 36 | 0 | 56 | 3 | 57 | 4 | 54 | 4 | 43 | 4 | 65 | 1 |
| **V3** | 28 | 1 | 47 | 1 | 78 | 4 | 63 | 2 | 35 | 2 | 39 | 2 | 66 | 4 | 113 | 9 | 44 | 4 | 65 | 2 |
| **V4** | 32 | 2 | 48 | 1 | 39 | 3 | 30 | 2 | 33 | 2 | 34 | 2 | 67 | 2 | 38 | 2 | 39 | 0 | 36 | 1 |
| **V5** | 37 | 2 | 50 | 3 | 40 | 1 | 52 | 2 | 51 | 5 | 39 | 0 | 28 | 0 | 34 | 2 | 55 | 3 | 41 | 1 |

### 3.5.5. Conclusiones

Podemos ver en las pruebas que los resultados hasta la versión 3 de la presa han mejorado respecto a la anterior versión, dejando claro que la protección de la comida da bastantes buenos resultados. Aún así, las últimas versiones de la presa siguen dando resultados parecidos, dejando claro que aún hay mucho por mejorar.

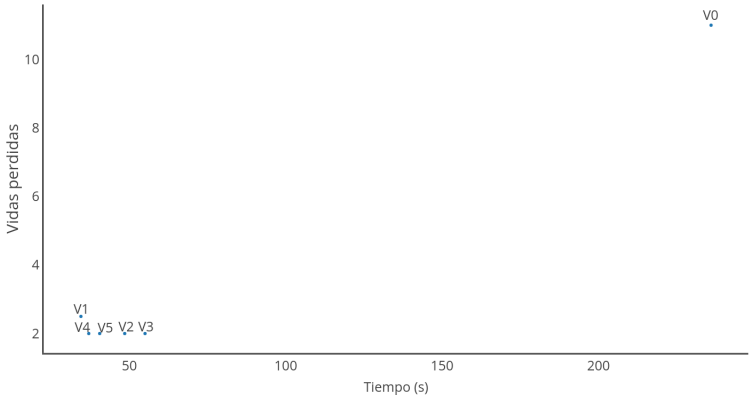



**Figura 3.21:** Gráfico con las medianas de la tabla 3.5.

**Tabla 3.6:** 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** | 287 | 21 | 91 | 2 | 135 | 11 | 256 | 22 | 98 | 4 | 309 | 17 | 144 | 14 | 122 | 11 | 283 | 23 | 136 | 7 |
| **V1** | 123 | 7 | 95 | 6 | 43 | 1 | 35 | 1 | 33 | 2 | 27 | 1 | 33 | 1 | 22 | 1 | 85 | 4 | 79 | 4 |
| **V2** | 44 | 4 | ∞ | 0 | 62 | 3 | 33 | 3 | 62 | 1 | 67 | 2 | 65 | 4 | 72 | 4 | 56 | 0 | 63 | 4 |
| **V3** | 48 | 1 | 24 | 3 | 23 | 2 | 46 | 1 | 70 | 6 | 49 | 0 | 37 | 1 | 52 | 3 | 28 | 1 | 54 | 5 |
| **V4** | 27 | 1 | 20 | 0 | 24 | 0 | 20 | 1 | 30 | 2 | 43 | 1 | 25 | 1 | 23 | 0 | 31 | 1 | 40 | 6 |
| **V5** | 21 | 1 | 23 | 2 | 42 | 0 | 29 | 1 | 28 | 2 | 30 | 2 | 29 | 1 | 27 | 1 | 27 | 1 | 20 | 0 |

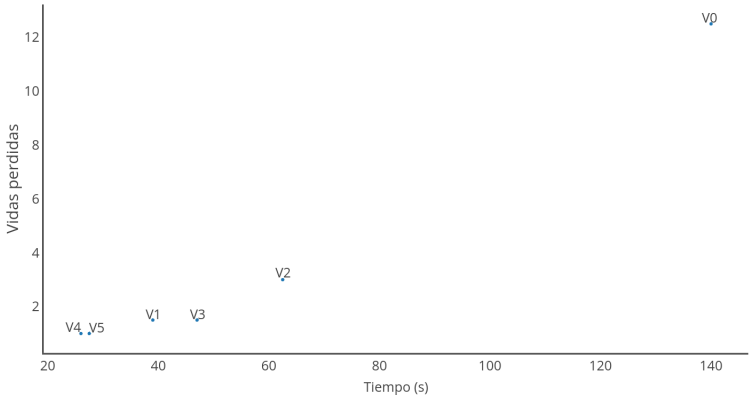



**Figura 3.22:** Gráfico con las medianas de la tabla 3.6.

### 3.5.6. A mejorar

El mayor problema de la versión es que puede que los dos cazadores se queden en la misma parte del mapa, dejando a la presa comer libremente toda la comida del resto del mapa. En la siguiente versión se intentará arreglar esto, dando instrucciones a los fantasmas para que protejan una zona concreta.

## 3.6. Versión 1.3

El objetivo de esta versión será mejorar el comportamiento de la anterior dividiendo el mapa entre los dos cazadores y probarlo con todas las versiones de la presa.

### 3.6.1. Influencias

Al tener la misma base que la anterior versión la mayoría de influencias son las mismas. El único cambio es la división del trabajo. Esta vez se ha decidido implementar también la defensa en zona, muy usada en multitud de deportes para tener controlado al enemigo en todo el entorno.

### 3.6.2. Inteligencia

La inteligencia ha tenido un cambio importante aunque solo se haya cambiado una pequeña parte de ella. En esta versión los dos cazadores serán protectores, pero esta vez cada uno defenderá una zona diferente: el fantasma azul protegerá la zona de arriba y el fantasma rosa defenderá la zona de abajo (Fig. 3.23). Esto se hace para que la presa no pueda moverse libremente por una gran zona del mapa, ya que siempre andará un cazador por sus alrededores.

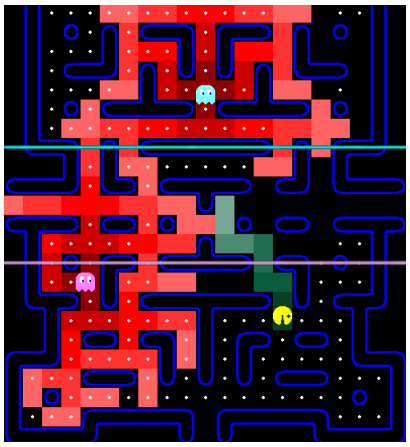


**Figura 3.23:** Los dos cazadores se dividen el mapa para protegerlo.

Como en la anterior versión, cuando uno de los cazadores vea la presa empezará a seguirla y le avisará al otro de su posición, usando la inteligencia de la versión 1.1.

Podemos ver el diagrama de flujos de esta versión en la figura 3.24.

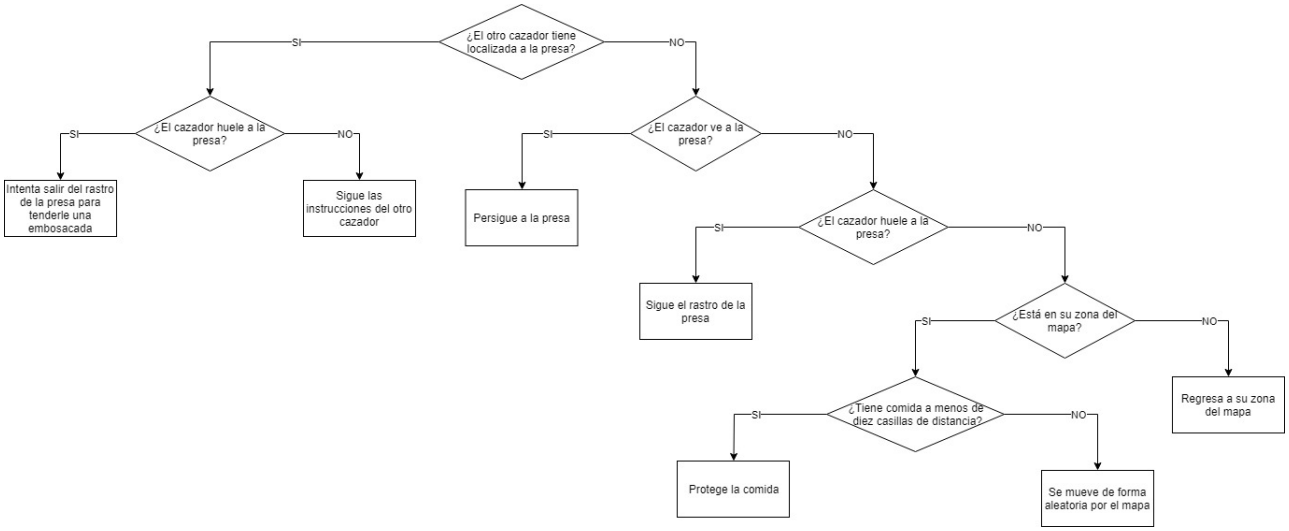



**Figura 3.24:** Diagrama de flujos de la versión 1.3.

### 3.6.3. Implementación

Para hacer esta versión no ha habido muchos cambios en el código, pero dado el alto cambio en el comportamiento de los fantasmas se ha decidido que este pequeño cambio sea una versión nueva.

El cambio principal ha sido al momento de decidir cómo siguen la comida los cazadores, como se ve en el pseudocódigo A.7.

Como vemos, el casi único cambio al código es en que ahora un fantasma intentará no pasar de la fila 9 hacia abajo y el otro intentara no pasar de la fila 15 hacia arriba. Aparte de este cambio, ahora los dos fantasmas seguirán la comida y solo tendrán en cuenta la comida que esté como mucho a 7 casillas de distancia, para que no detecten la comida de la zona del otro fantasma. Con estos pequeños cambios hacemos que el comportamiento de los cazadores cambie mucho, ya que ahora se centrarán en proteger su zona, complicándole el trabajo a la presa.

### 3.6.4. Pruebas

Como en las anteriores versiones, contaremos con dos tablas y dos gráficos: una tabla y un gráfico para el primer mapa, tabla 3.7 y figura 3.25, y lo mismo para el segundo, tabla

3.8 y figura 3.26.

**Tabla 3.7:** 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** | 351 | 18 | 161 | 8 | 252 | 14 | 180 | 5 | 302 | 16 | 176 | 16 | 137 | 3 | 183 | 17 | 267 | 16 | 243 | 12 |
| **V1** | 36 | 2 | 53 | 4 | 51 | 7 | 43 | 2 | 41 | 3 | 56 | 7 | 49 | 4 | 88 | 11 | 31 | 3 | 83 | 4 |
| **V2** | 40 | 2 | 42 | 5 | 44 | 4 | 68 | 2 | 96 | 6 | 83 | 1 | 68 | 3 | 85 | 2 | 121 | 2 | 34 | 1 |
| **V3** | 96 | 4 | 36 | 1 | 45 | 4 | 34 | 3 | 32 | 0 | 48 | 2 | 38 | 1 | 35 | 5 | 35 | 2 | 117 | 2 |
| **V4** | 25 | 0 | 32 | 1 | 26 | 1 | 32 | 3 | 29 | 1 | 27 | 2 | 23 | 2 | 43 | 4 | 34 | 2 | 29 | 1 |
| **V5** | 38 | 0 | 37 | 1 | 53 | 3 | ∞ | 1 | 41 | 1 | 51 | 2 | 38 | 1 | 30 | 0 | 28 | 0 | 32 | 1 |

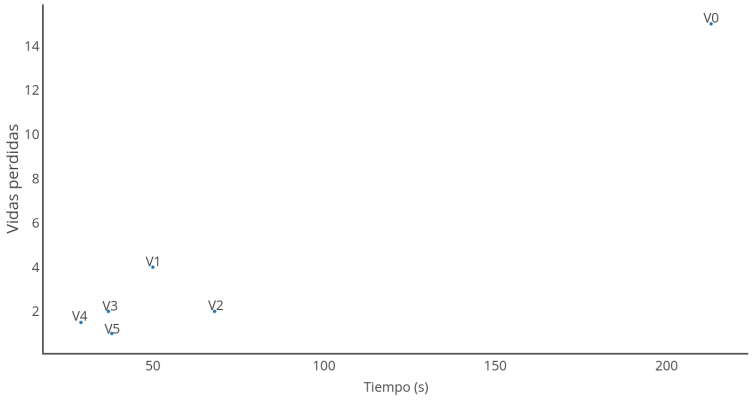



**Figura 3.25:** Gráfico con las medianas de la tabla 3.7.

### 3.6.5. Conclusiones

Como podemos ver en las pruebas, los resultados hasta la segunda versión de la presa vuelven a mejorar un poco, pero de ahí en adelante los resultados se mantienen estables respecto al número de veces que la presa ha sido capturada. Si nos fijamos en el tiempo que ha necesitado la presa podemos apreciar una mejora en todas las versiones, aunque en las 3 últimas no le haya servido para atrapar más veces a la presa.

**Tabla 3.8:** Resultados de la versión 3 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** | ∞ | 8 | ∞ | 2 | ∞ | 0 | ∞ | 4 | ∞ | 0 | ∞ | 0 | ∞ | 0 | 367 | 19 | ∞ | 2 | ∞ | 1 |
| **V1** | 41 | 1 | 36 | 12 | ∞ | 2 | 80 | 2 | ∞ | 0 | ∞ | 5 | 206 | 4 | 39 | 1 | ∞ | 0 | 33 | 6 |
| **V2** | ∞ | 0 | ∞ | 0 | 35 | 0 | ∞ | 0 | 50 | 1 | 90 | 3 | ∞ | 0 | ∞ | 0 | ∞ | 1 | ∞ | 1 |
| **V3** | 54 | 1 | 29 | 1 | ∞ | 0 | 43 | 1 | 36 | 2 | ∞ | 1 | 27 | 3 | 182 | 2 | 78 | 2 | 34 | 2 |
| **V4** | 21 | 1 | 32 | 0 | ∞ | 0 | 20 | 1 | 23 | 2 | 27 | 1 | 23 | 2 | 23 | 2 | 30 | 2 | 21 | 2 |
| **V5** | 29 | 2 | 32 | 1 | 28 | 1 | 30 | 1 | 31 | 3 | 29 | 2 | 41 | 2 | ∞ | 1 | 27 | 3 | 35 | 5 |

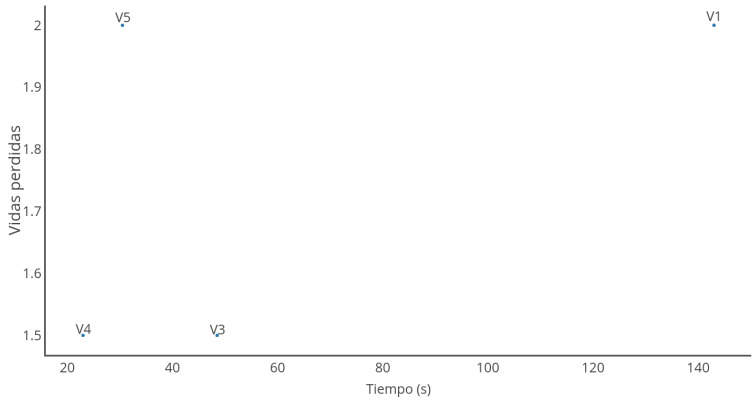



**Figura 3.26:** Gráfico con las medianas de la tabla 3.8.

### 3.6.6. A mejorar

El mayor problema de esta versión empieza a partir de la versión 1.3 de la presa, que es cuando empieza a usar las *Power Pellets*, que le sirven para poder comerse a los fantasmas. En la próxima versión se intentará gestionar esto, haciendo que los fantasmas se alejen de la presa cuando están en modo vulnerable.

## 3.7. Versión 1.4

En esta versión analizaremos la mejora que supone el huir de la presa cuando los cazadores están en modo vulnerable, probándolo con las versiones de la presa que usan las *Power Pellets*.

### 3.7.1. Influencias

En esta versión seguimos usando influencias de deportes, en este caso el contraataque. Mientras los cazadores estén en modo vulnerable no pueden atrapar a la presa, pero si pueden mantener una distancia prudencial y atacarla cuando esta ya no tenga la ventaja.

### 3.7.2. Inteligencia

En esta nueva versión hemos añadido un nuevo comportamiento a los cazadores, el de huir. Mientras estén en modo vulnerable escaparán de la presa en el momento que la huelan o la vean. Mientras huyen le dirán al otro cazador por donde anda la presa para que vaya acercándose y poder atacarla cuando ya no estén en modo vulnerable (Fig. 3.27).

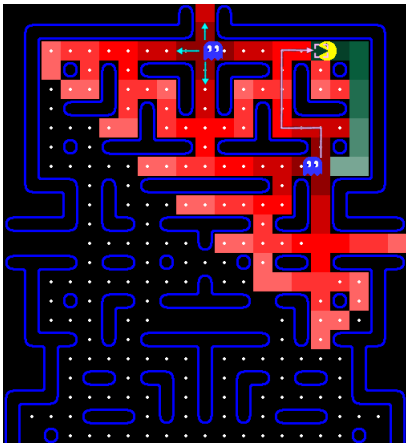


**Figura 3.27:** Un cazador escapa mientras que el otro se acerca.

Mientras no tengan contacto con la presa los fantasmas seguirán actuando exactamente igual que en la anterior versión, protegiendo su zona del mapa y la comida que allí se encuentra.

Podemos ver el diagrama de flujos de esta versión en la figura 3.28.

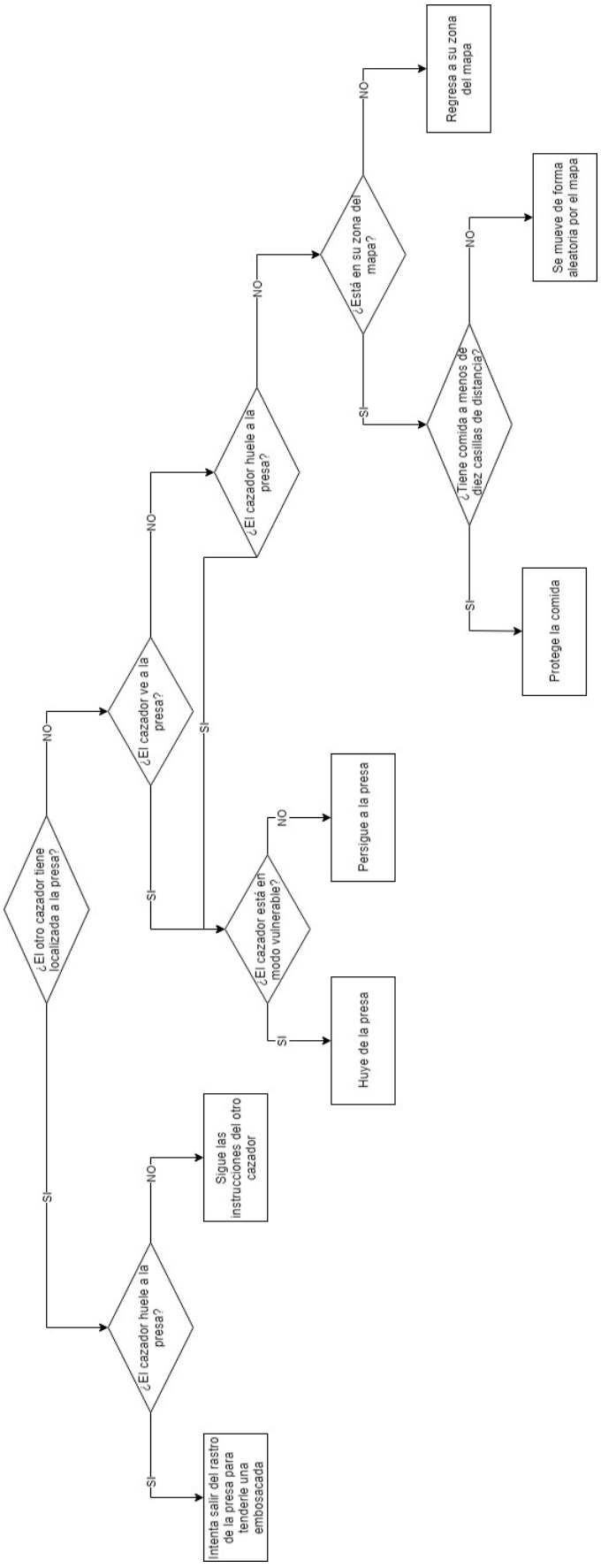



**Figura 3.28:** Diagrama de flujos de la versión 1.4.

### 3.7.3. Implementación

Para implementar esta versión se ha tenido que añadir código para gestionar la visión y el olor cuando los cazadores están en modo vulnerable, como se puede ver en el pseudocódigo A.8.

Como se ve en el pseudocódigo, se han dejado intactas las partes en las que los cazadores no están amenazados por la presa y siempre se le dirá al otro cazador la posición de ésta. Cuando los cazadores vean o huelan a la presa y estén en modo vulnerable intentarán alejarse, metiendo en la lista de direcciones posibles cualquier dirección que no tenga pared menos la que se dirige hacia la presa.

### 3.7.4. Pruebas

Contaremos con dos tablas y dos gráficos: una tabla y un gráfico para el primer mapa, tabla 3.9 y figura 3.29, y lo mismo para el segundo, tabla 3.10 y figura 3.30. Esta vez solo analizaremos las versiones en las que la presa usa las *Power-Pellets*, ya que el comportamiento de esta versión de los cazadores solo cambia cuando la presa toma una de estas.

**Tabla 3.9:** 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** |
| **V3** | 64 | 7 | 36 | 2 | 39 | 0 | 35 | 0 | 48 | 2 | 52 | 3 | 33 | 1 | 32 | 4 | 44 | 0 | 49 | 2 |
| **V4** | 29 | 1 | 27 | 1 | 29 | 1 | 27 | 0 | 32 | 1 | 44 | 2 | 32 | 2 | 32 | 1 | 30 | 1 | 49 | 4 |
| **V5** | 34 | 1 | 39 | 2 | 41 | 3 | 50 | 0 | 28 | 0 | 37 | 4 | 29 | 2 | 36 | 3 | ∞ | 2 | 31 | 1 |

**Tabla 3.10:** 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** |
| **V3** | ∞ | 0 | 33 | 1 | 36 | 5 | ∞ | 0 | ∞ | 0 | 30 | 1 | 36 | 1 | 27 | 2 | 39 | 3 | ∞ | 1 |
| **V4** | 28 | 3 | 59 | 3 | 40 | 4 | 31 | 2 | 26 | 1 | 61 | 7 | 26 | 1 | 25 | 3 | 31 | 1 | 26 | 1 |
| **V5** | ∞ | 1 | ∞ | 0 | 29 | 0 | 30 | 1 | 38 | 3 | 26 | 2 | 40 | 3 | ∞ | 0 | 38 | 2 | 23 | 2 |

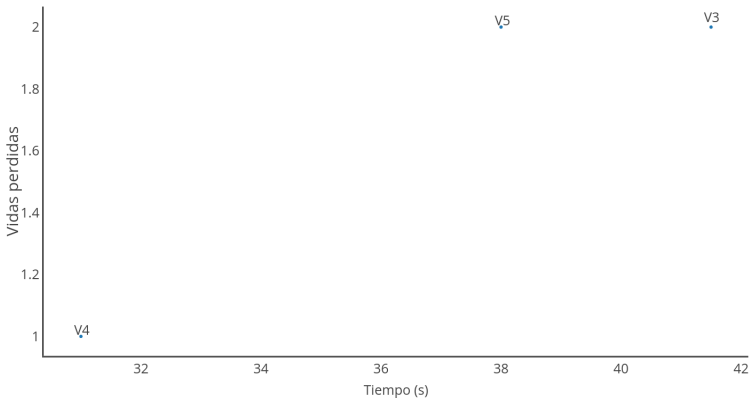



**Figura 3.29:** Gráfico con las medianas de la tabla 3.9.

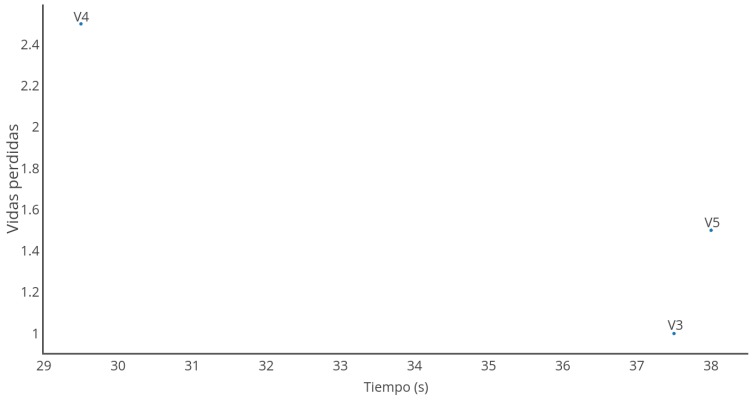



**Figura 3.30:** Gráfico con las medianas de la tabla 3.10.

### 3.7.5. Conclusiones

Como vemos en las pruebas, los resultados son muy parecidos a los de la anterior versión, aunque en la versión 3 podemos ver una pequeña mejora. En el segundo mapa también podemos apreciar mejores resultados respecto a la anterior versión, hecho que no se da en el primer mapa. Esto se debe a la estructura del segundo mapa, que tiene menos pasillos largos y cuenta con más zonas cerradas.

### 3.7.6. A mejorar

Como vemos al ejecutar esta última versión, la presa usa mucho los teletransportes para huir de los cazadores, haciendo que estos le pierdan el rastro. En la siguiente versión se intentará gestionar esto haciendo que uno de los cazadores vaya a la salida del teletransporte.

## 3.8. Versión 1.5

En esta versión analizaremos la mejora de la inteligencia intentando predecir el uso de los teletransportes, probándolo con todos las versiones de la presa.

### 3.8.1. Influencias

Esta vez intentaremos que la presa crea que esta siendo perseguida por varios cazadores en una zona obligándola a usar el teletransporte y esperándola al otro lado para poder cazarla.

### 3.8.2. Inteligencia

Para intentar gestionar los teletransportes se va a dividir el mapa en 5 partes: arriba, abajo, derecha, izquierda y centro. En la parte del centro se seguirá usando la misma inteligencia que en la anterior versión, los cambios estarán en las otras partes. Si un cazador está siguiendo a la presa en cualquiera de estas cuatro partes el otro cazador intentará cortarle el paso en la salida del teletransporte, como podemos ver en las figuras 3.31 y 3.32.

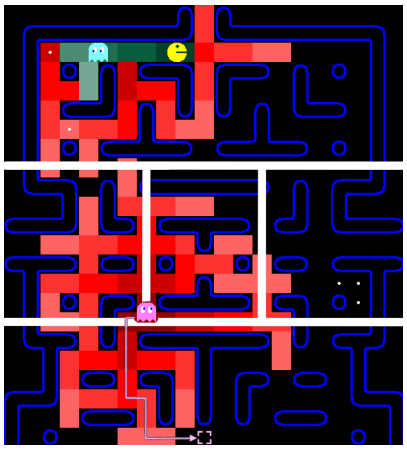


**Figura 3.31:** Uno de los cazadores le corta el paso en el tp de abajo.

Podemos ver el diagrama de flujos de esta versión en la figura 3.33.

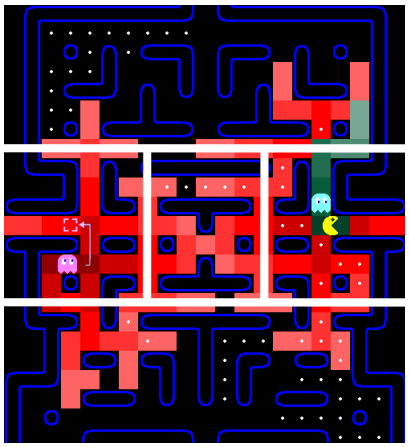


**Figura 3.32:** Uno de los cazadores le corta el paso en el tp de la izquierda.

### 3.8.3. Implementación

Se van a cambiar pocas cosas en esta versión, solo cambiaremos algunas cosas al mandarle la posición a la que debe ir al otro cazador. Como hemos dicho antes, se le mandará al otro cazador a la salida del teletransporte, como podemos ver en el pseudocódigo A.9.

Como vemos se han añadido 4 condiciones nuevas para comunicarse entre los cazadores, el resto del código se ha dejado igual.

### 3.8.4. Pruebas

Contaremos con dos tablas y dos gráficos: una tabla y un gráfico para el primer mapa, tabla 3.11 y figura 3.34, y lo mismo para el segundo, tabla 3.12 y figura 3.35.

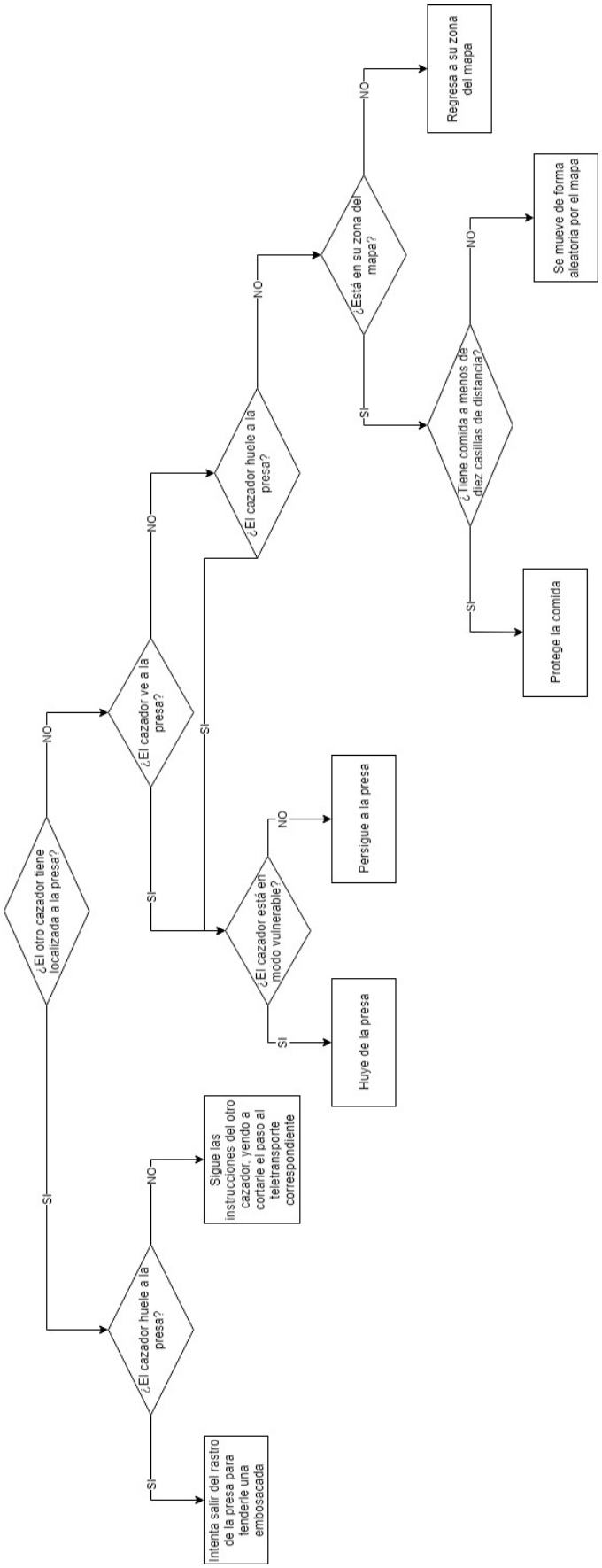



**Figura 3.33:** Diagrama de flujos de la versión 1.5.

**Tabla 3.11:** 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** | 264 | 9 | 196 | 7 | 343 | 13 | 190 | 11 | 301 | 6 | 134 | 6 | 266 | 16 | 225 | 6 | 343 | 6 | 314 | 9 |
| **V1** | 80 | 0 | 39 | 1 | 35 | 3 | 33 | 3 | 30 | 2 | 50 | 1 | 28 | 1 | 29 | 3 | 28 | 1 | 48 | 1 |
| **V2** | 60 | 5 | ∞ | 2 | 44 | 2 | 40 | 4 | 49 | 5 | 56 | 5 | 35 | 2 | 41 | 1 | 91 | 5 | 57 | 3 |
| **V3** | 37 | 1 | 38 | 5 | 68 | 5 | 57 | 5 | 38 | 7 | 45 | 7 | 65 | 9 | 87 | 6 | 79 | 2 | 52 | 5 |
| **V4** | 33 | 4 | 65 | 8 | 36 | 5 | 56 | 7 | 50 | 3 | 43 | 5 | 45 | 4 | 30 | 2 | 25 | 0 | 34 | 1 |
| **V5** | 58 | 3 | 24 | 1 | 51 | 3 | 54 | 5 | 49 | 5 | 32 | 2 | 37 | 2 | 26 | 2 | 45 | 4 | 31 | 2 |

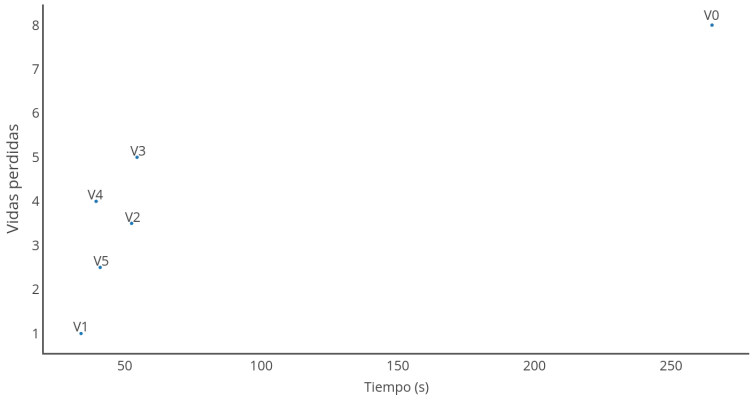



**Figura 3.34:** Gráfico con las medianas de la tabla 3.11.

### 3.8.5. Conclusiones

Como podemos ver en las pruebas, los resultados han empeorado en las primeras versiones y mejorado en las últimas. Esto se debe a que en las primeras versiones la presa no le daba casi ningún uso a los teletransportes, haciendo que intentar predecir su uso no tenga ningún sentido. Por otro lado, esta vez se han conseguido mejorar los resultados de las dos últimas versiones, por lo que tendremos que seguir por este camino para conseguir los mejores resultados posibles en todas las versiones.

**Tabla 3.12:** 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** | 103 | 2 | 174 | 4 | ∞ | 0 | ∞ | 1 | ∞ | 0 | ∞ | 0 | ∞ | 2 | ∞ | 1 | ∞ | 0 | 73 | 7 |
| **V1** | ∞ | 1 | 22 | 1 | 22 | 0 | 147 | 3 | 38 | 4 | 36 | 0 | 32 | 1 | 34 | 1 | 21 | 0 | ∞ | 0 |
| **V2** | ∞ | 1 | ∞ | 0 | ∞ | 0 | ∞ | 0 | ∞ | 1 | 67 | 2 | 39 | 1 | ∞ | 1 | 39 | 0 | ∞ | 1 |
| **V3** | 35 | 4 | ∞ | 0 | 39 | 2 | 63 | 3 | 70 | 3 | 29 | 5 | 30 | 1 | ∞ | 0 | ∞ | 0 | ∞ | 0 |
| **V4** | 27 | 2 | 24 | 3 | 24 | 3 | ∞ | 0 | 26 | 4 | ∞ | 1 | ∞ | 2 | 43 | 3 | ∞ | 2 | 37 | 1 |
| **V5** | ∞ | 0 | 23 | 2 | 37 | 3 | 23 | 1 | 37 | 3 | 61 | 4 | 34 | 1 | 49 | 5 | ∞ | 0 | 41 | 1 |

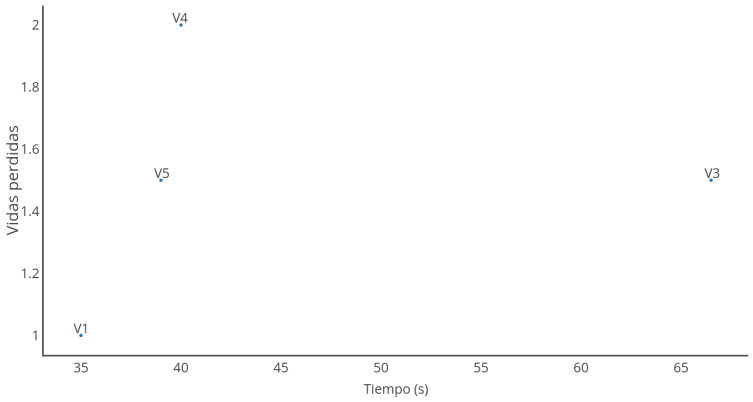



**Figura 3.35:** Gráfico con las medianas de la tabla 3.12.

### 3.8.6. A mejorar

Vemos que al añadirle una ventaja a la presa sus resultados han mejorado bastante, por lo que en la siguiente versión vamos a añadirle una ventaja nueva a los fantasmas, la que les servirá para tener una ventaja temporal frente a la presa.

## 3.9. Versión 1.6

En esta versión se va a implementar un cambio en la estructura del juego, que hará que aparezcan unas nuevas pastillas en el mapa que ayudarán a los cazadores. Esta vez vamos a implementar dos subversiones: una en la que los cazadores intentarán cortarle el paso a la presa en los teletransportes y otra en la que intentarán atraparla por todo el mapa sin tener en cuenta los teletransportadores.

### 3.9.1. Influencias

En muchos juegos se pueden encontrar mejoras para el jugador, repartidas por diferentes zonas del mapa. En nuestro caso, el juego *Pac-Man*, solo se pueden encontrar mejoras para el jugador, pero como nosotros ya no controlamos al *Pac-Man* hemos decidido añadirle también una mejora a los fantasmas. Al pensar en una mejora posible para estos se ha inspirado en todas las películas, libros y series en las que los fantasmas tienen la capacidad de traspasar paredes. Como seria demasiado darles esta capacidad de forma ilimitada se ha decidido que sea por un número de paredes determinado.

### 3.9.2. Inteligencia

Como se ha mencionado más arriba, esta vez vamos a trabajar en dos subversiones, por lo que contaremos con dos inteligencias, aunque las dos tengan bastantes cosas en común.

#### Cosas en común entre las dos versiones

En esta nueva versión vamos a aplicar una nueva pastilla. Ésta hará que los fantasmas sean capaces de traspasar las paredes tres veces antes de que se acabe su efecto. Se han colocado cuatro por todo el mapa (Fig. 3.36, izquierda), para que sean el mismo número que las *power-pellets* que usa la presa. Basándose en ese nombre se ha decidido que las nuevas pastillas se llamen *ghost-pellets*.

Cuando uno de los cazadores tome una de estas pastillas se volverá casi invisible, quedando solo sus ojos a la vista, como podemos ver en la figura 3.36, a la derecha.

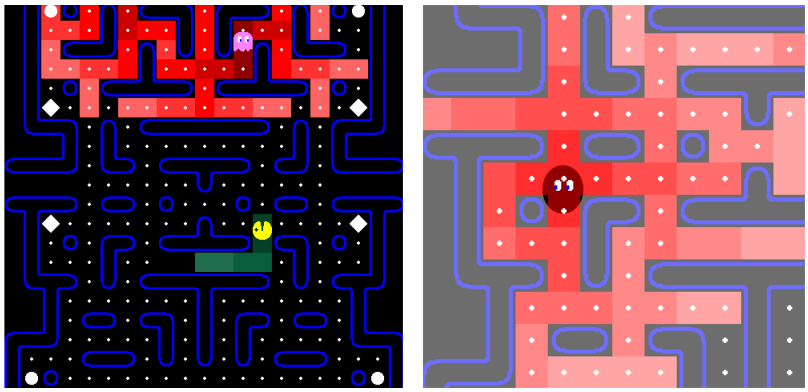


**Figura 3.36:** Izquierda: posición de las *ghost-pellets* en el mapa. Derecha: cazador después de tomar una *ghost-pellet*.

### Versión 1.6.1

En esta versión le hemos añadido las *ghost-pellets* a la versión 1.5 haciendo que los cazadores intenten cortarle el paso en los teletransportes mientras son capaces de traspasar algunas paredes, como podemos ver en la figura 3.37. Al buscar el camino más corto hacia su objetivo, ignorará las paredes siempre que le quede algún uso de la *ghost-pellet*, por lo demás la inteligencia es exactamente la misma que en la versión 1.5.

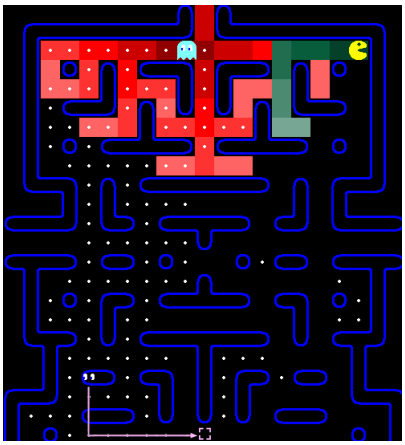


**Figura 3.37:** Cazador yendo a cortar el paso al tp traspasando paredes.

Podemos ver el diagrama de flujos de esta versión en la figura 3.38.

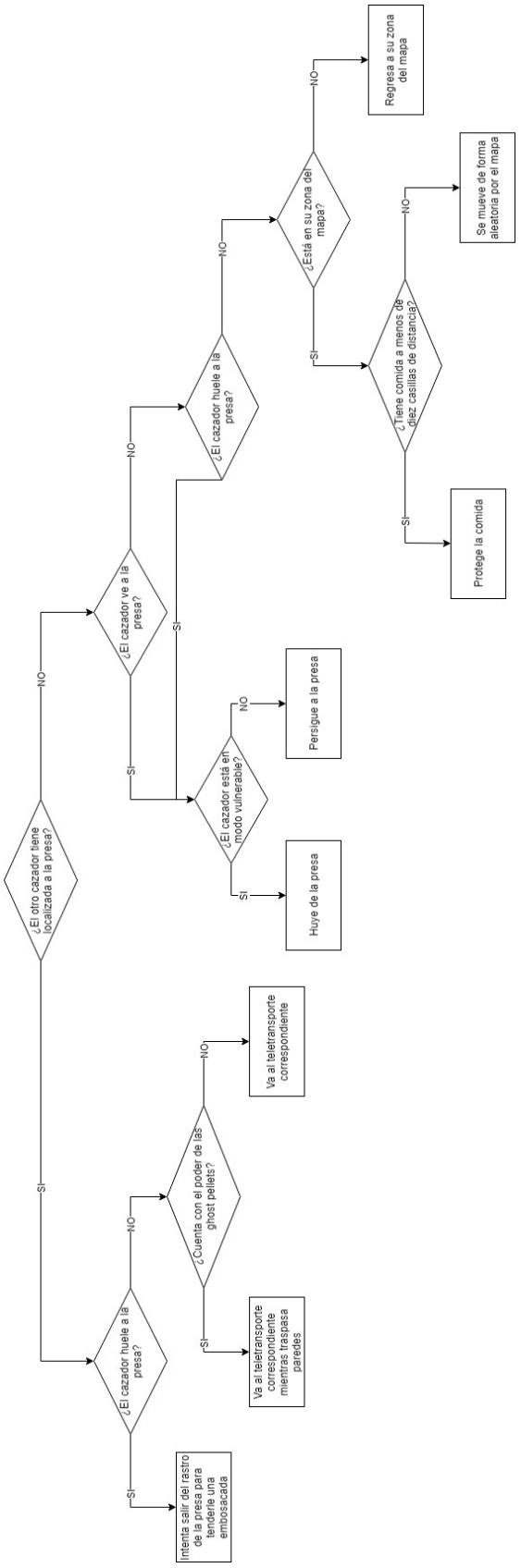



**Figura 3.38:** Diagrama de flujos de la versión 1.6.1.

Versión 1.6.2

Esta vez le añadiremos el uso de las *ghost-pellets* a la versión 1.4 haciendo que los cazadores intenten cortarle el paso a la presa mientras cruzan algunas paredes, como podemos ver en la figura 3.39. Como en la versión 1.6.1, los cazadores ignorarán las paredes mientras estén persiguiendo a la presa siempre que les quede algún uso de la *power-pellet*, el resto del comportamiento será igual que en la versión 1.4.

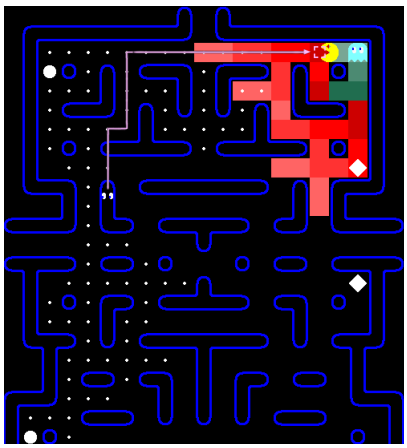


**Figura 3.39:** Cazador yendo a cortar el paso traspasando paredes.

Podemos ver el diagrama de flujos de esta versión en la figura 3.40.

### 3.9.3. Implementación

Aunque hayamos creado dos versiones el código añadido es el mismo, solo que en la versión 1.6.1 se lo hemos añadido a la versión 1.5 y en la versión 1.6.2 se lo hemos añadido a la versión 1.4. En el código A.10 podemos ver lo que se ha añadido en el archivo *level* y en el archivo *ghost*.

Como vemos en esta parte del código, en el archivo *level* se ha añadido una función para que los cazadores puedan coger las *ghost-pellets* y en *ghost* hemos añadido a las condiciones de moverse en una dirección que no haya pared hacia ese lado o que el número de paredes que puede cruzar sea mayor que 0.

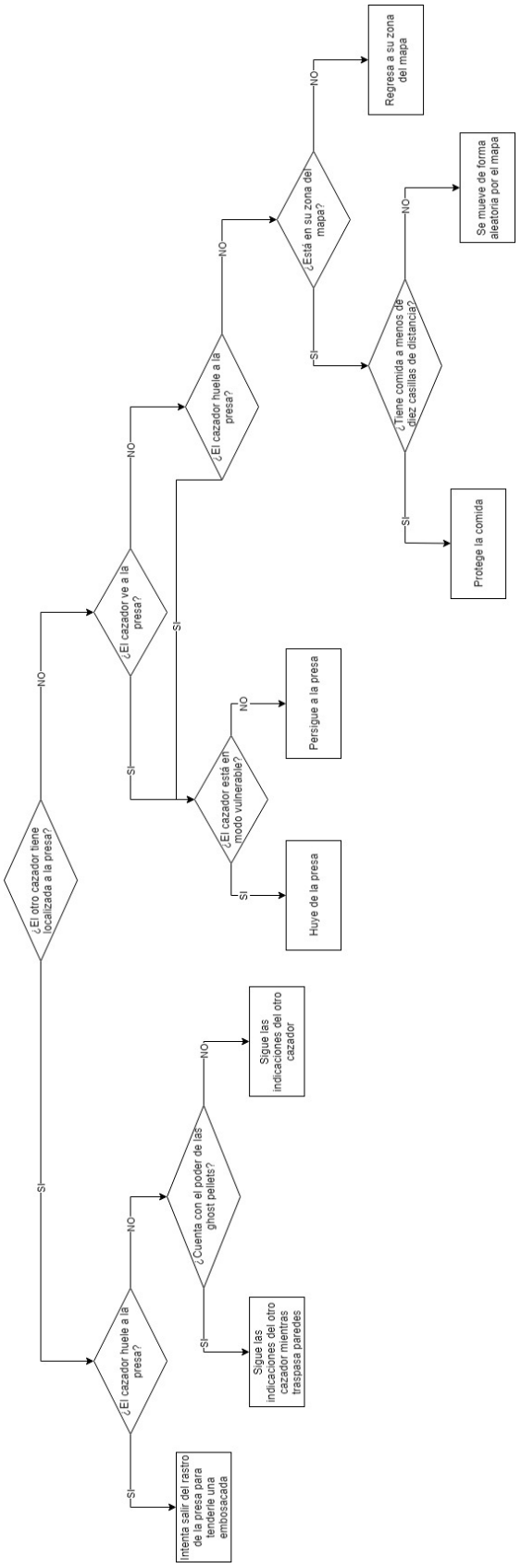



**Figura 3.40:** Diagrama de flujos de la versión 1.6.2.

### 3.9.4. Pruebas

Esta vez contamos con dos subversiones, por lo que en lugar de dos contaremos con cuatro tablas y gráficos. Como siempre, para cada version tendremos dos tablas y dos gráficos, una tabla y un gráfico para cada nivel, tabla 3.13 y figura 3.41 y tabla 3.14 y figura 3.42 para la primera versión y tabla 3.15 y figura 3.43 y tabla 3.16 y figura 3.44 para la segunda.

#### Versión 1.6.1

**Tabla 3.13:** Resultados de la versión 6.1 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** | 165 | 9 | 320 | 24 | 252 | 10 | 139 | 4 | 89 | 4 | 352 | 11 | 285 | 12 | 152 | 8 | 222 | 7 | 289 | 7 |
| **V1** | 62 | 4 | 58 | 6 | 80 | 1 | 42 | 3 | 48 | 2 | 33 | 1 | 26 | 1 | 23 | 0 | 26 | 0 | 29 | 1 |
| **V2** | 66 | 2 | ∞ | 0 | 107 | 3 | 126 | 3 | 72 | 6 | 49 | 1 | 31 | 1 | 46 | 4 | 40 | 3 | 41 | 4 |
| **V3** | 43 | 2 | 33 | 1 | 34 | 1 | 39 | 5 | 84 | 3 | 45 | 4 | 50 | 3 | 81 | 3 | 38 | 5 | 44 | 7 |
| **V4** | 31 | 3 | 29 | 3 | 85 | 10 | 35 | 2 | 55 | 4 | 30 | 1 | 89 | 6 | 30 | 3 | 88 | 13 | 100 | 14 |
| **V5** | 44 | 4 | 48 | 6 | 44 | 4 | 55 | 4 | 34 | 1 | 103 | 8 | 33 | 3 | 35 | 3 | 34 | 4 | 44 | 7 |

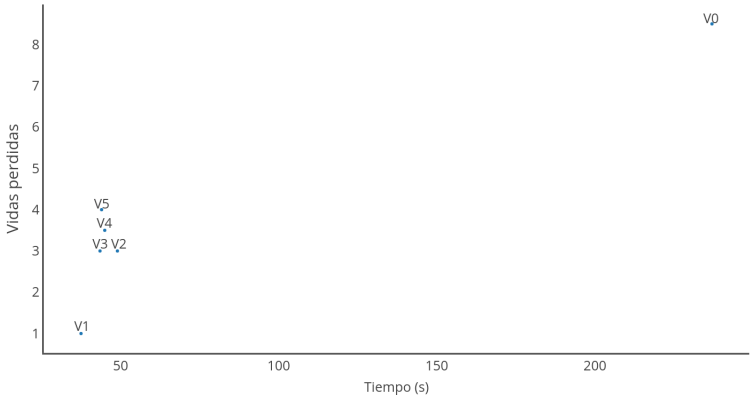



**Figura 3.41:** Gráfico con las medianas de la tabla 3.13.

**Tabla 3.14:** Resultados de la versión 6.1 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** | ∞ | 0 | ∞ | 0 | 111 | 1 | 275 | 13 | ∞ | 1 | 125 | 4 | ∞ | 0 | ∞ | 2 | ∞ | 2 | 248 | 10 |
| **V1** | 20 | 0 | ∞ | 2 | ∞ | 0 | 32 | 5 | 50 | 0 | ∞ | 0 | 19 | 0 | 25 | 1 | ∞ | 0 | ∞ | 0 |
| **V2** | ∞ | 0 | 51 | 7 | 84 | 2 | 32 | 4 | ∞ | 0 | ∞ | 1 | 93 | 6 | ∞ | 0 | 237 | 12 | 83 | 4 |
| **V3** | ∞ | 2 | 43 | 1 | 30 | 2 | 45 | 3 | 36 | 3 | 105 | 2 | 43 | 4 | ∞ | 1 | 38 | 2 | 21 | 1 |
| **V4** | 19 | 1 | 55 | 5 | 43 | 3 | ∞ | 0 | ∞ | 2 | 39 | 4 | 24 | 3 | 39 | 5 | 29 | 1 | 27 | 2 |
| **V5** | 23 | 3 | ∞ | 0 | 23 | 0 | 28 | 2 | 23 | 2 | 35 | 1 | 35 | 6 | 22 | 0 | 25 | 2 | 69 | 9 |

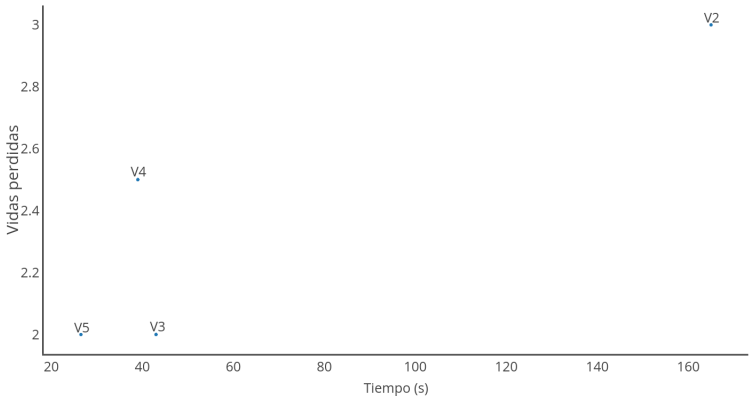



**Figura 3.42:** Gráfico con las medianas de la tabla 3.14.

**Tabla 3.15:** Resultados de la versión 6.2 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** | 193 | 12 | 83 | 7 | 182 | 13 | ∞ | 4 | 398 | 19 | 292 | 12 | 199 | 14 | 316 | 21 | ∞ | 16 | 239 | 17 |
| **V1** | 36 | 4 | 52 | 10 | 39 | 3 | 94 | 7 | ∞ | 5 | 43 | 2 | 40 | 3 | 42 | 4 | 39 | 4 | 57 | 4 |
| **V2** | 77 | 1 | 133 | 6 | 46 | 3 | 51 | 2 | 36 | 5 | 35 | 3 | 70 | 4 | 77 | 5 | 52 | 2 | 64 | 0 |
| **V3** | 62 | 3 | 84 | 7 | 28 | 1 | 37 | 1 | 189 | 7 | 33 | 2 | 49 | 1 | 43 | 1 | 50 | 4 | 56 | 2 |
| **V4** | 29 | 1 | 39 | 2 | 34 | 1 | 43 | 5 | 28 | 1 | 27 | 1 | 26 | 1 | 37 | 1 | 40 | 0 | 30 | 2 |
| **V5** | 31 | 0 | 32 | 0 | 27 | 1 | 35 | 3 | 28 | 2 | 83 | 7 | 45 | 1 | 43 | 4 | 35 | 3 | 31 | 1 |

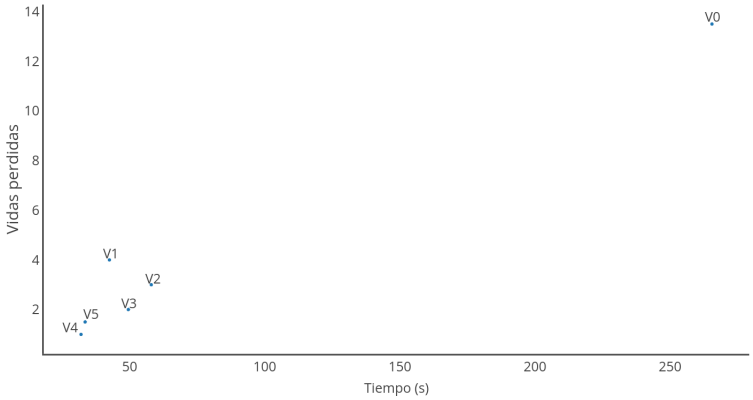



**Figura 3.43:** Gráfico con las medianas de la tabla 3.15.

**Tabla 3.16:** Resultados de la versión 6.2 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** | ∞ | 4 | ∞ | 3 | ∞ | 1 | ∞ | 0 | ∞ | 0 | 100 | 8 | ∞ | 1 | ∞ | 0 | ∞ | 2 | ∞ | 3 |
| **V1** | 85 | 7 | 25 | 7 | 59 | 3 | ∞ | 0 | 34 | 6 | ∞ | 0 | ∞ | 0 | 61 | 8 | 63 | 4 | ∞ | 0 |
| **V2** | ∞ | 3 | 122 | 4 | 41 | 1 | 68 | 1 | ∞ | 4 | 122 | 5 | ∞ | 0 | 28 | 1 | ∞ | 0 | 67 | 0 |
| **V3** | ∞ | 4 | ∞ | 1 | ∞ | 2 | 60 | 1 | 25 | 2 | 37 | 1 | 27 | 2 | 26 | 2 | ∞ | 0 | 71 | 4 |
| **V4** | 27 | 2 | 58 | 3 | 21 | 0 | 34 | 3 | 26 | 0 | 45 | 4 | 21 | 0 | 19 | 0 | 29 | 1 | 35 | 2 |
| **V5** | 27 | 5 | 26 | 2 | 29 | 0 | 22 | 2 | 23 | 0 | 38 | 3 | 51 | 2 | 29 | 1 | 28 | 2 | 40 | 5 |

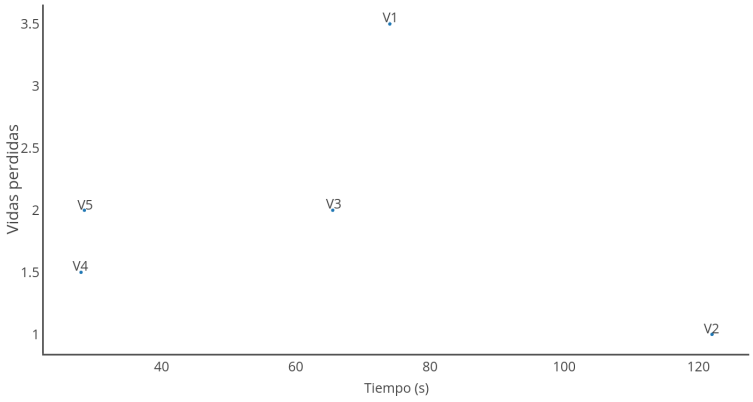



**Figura 3.44:** Gráfico con las medianas de la tabla 3.16.

Versión 1.6.2

### 3.9.5. Conclusiones

Como vemos en sus respectivas pruebas las dos son versiones mejoradas de anteriores cazadores, la 1.6.1 es una mejora a la 1.5 y la 1.6.2 es una mejora a la versión 1.4. Estas nuevas versiones intentar mejorar a sus predecesoras, pero también arrastran sus mayores problemas, haciendo que ninguna de las dos nuevas versiones sea efectiva con todas las versiones de la presa.

### 3.9.6. A mejorar

Como se ha analizado en las conclusiones en algunos casos es mejor atacar a la presa y en otros es mejor acercarse al teletransporte, por lo que en la siguiente versión intentaremos gestionar esto, juntando estas dos subversiones en una.

## 3.10. Versión 1.7

Como se ha dicho al final de la anterior versión, esta vez vamos a juntar las dos subversiones anteriores y comprobar su calidad con todas las versiones de la presa.

### 3.10.1. Influencias

Esta versión se va a hacer que los cazadores vayan al lugar de ataque más cercano para intentar conseguir los mejores resultados posibles.

### 3.10.2. Inteligencia

La inteligencia de esta versión será una combinación de las dos anteriores, haciendo que alguna vez los cazadores vayan directo hacia la presa y otras veces vayan hacia los teletransportes. Esta decisión dependerá de cual esté mas cerca del cazador, haciendo que vaya hacia la presa cuando esté cerca de ella y si no que vaya al teletransporte a intentar cortarle el paso, como podemos ver en la figura 3.45.

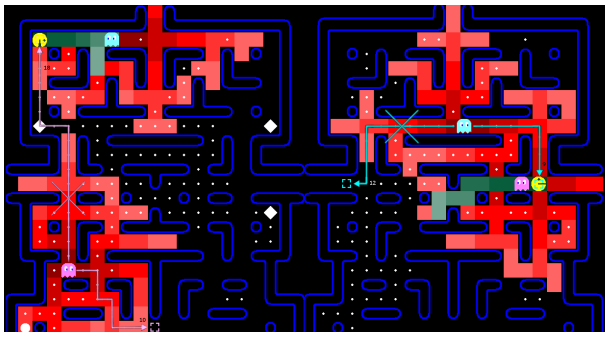


**Figura 3.45:** Comportamientos posibles del cazador en esta versión.

Como vemos en la figura el comportamiento del cazador depende de la distancia hacia los dos posibles objetivos. En el caso de la izquierda la presa está a 17 casillas de distancia y el teletransporte a 10, por lo que se decantará por ir hacia el segundo. En cambio en el caso de la derecha la presa está a 9 casillas de distancia y el teletransporte a 12, por lo que se decantará por el primero.

Podemos ver el diagrama de flujos de esta versión en la figura 3.46.

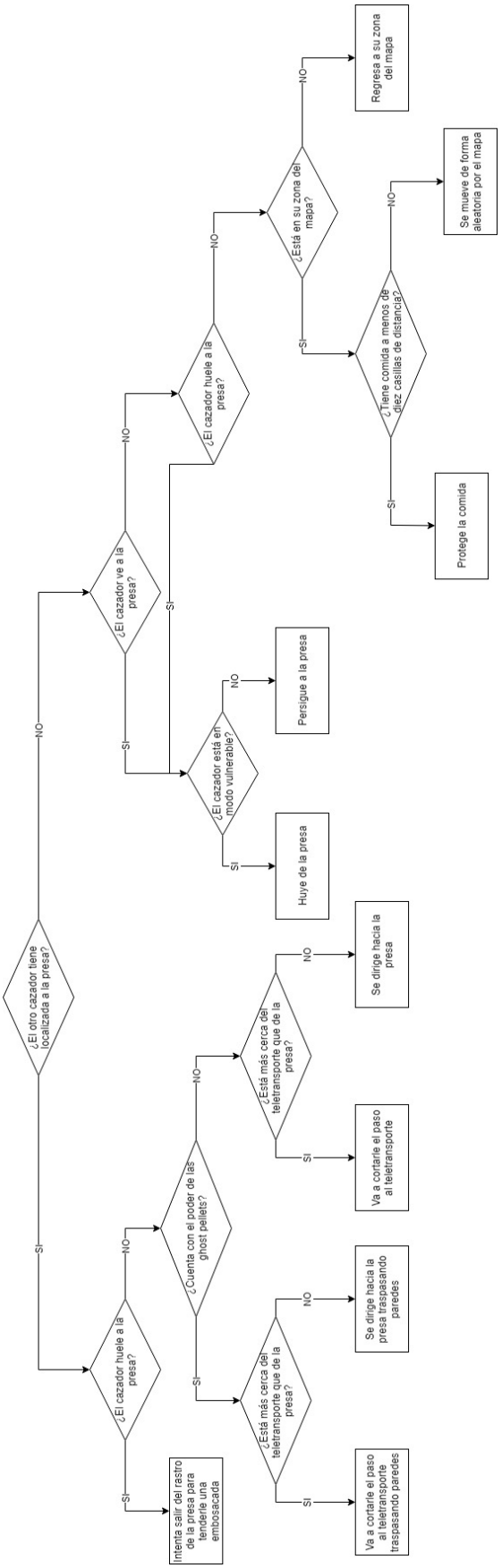



**Figura 3.46:** Diagrama de flujos de la versión 1.7.

### 3.10.3. Implementación

Vamos a aplicarle un pequeño cambio a la versión 1.6.1, haciendo que se guarde en una nueva variable la puerta a la que debería ir, lo que luego le servirá para comparar con el otro objetivo, como podemos ver en el código A.11.

Como se puede apreciar, cada vez que el cazador correspondiente vaya a hacer un movimiento se compararán las distancias y el que más cerca esté se guardará en la variable *ghostmove*, que será hacia donde se dirija el cazador.

### 3.10.4. Pruebas

Esta vez volvemos a contar con dos tablas y dos gráficos: una tabla y un gráfico para el primer mapa, tabla 3.17 y figura 3.47, y lo mismo para el segundo, tabla 3.18 y figura 3.48.

**Tabla 3.17:** 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** |
| **V0** | 221 | 13 | 134 | 7 | 140 | 6 | 170 | 6 | 209 | 17 | 369 | 8 | 161 | 6 | 265 | 7 | 175 | 9 | 189 | 9 |
| **V1** | 52 | 3 | 27 | 3 | 54 | 2 | 22 | 0 | 55 | 5 | 29 | 3 | 103 | 5 | 38 | 2 | 54 | 6 | 42 | 1 |
| **V2** | 45 | 4 | 51 | 3 | 44 | 4 | 67 | 5 | 31 | 2 | 60 | 5 | 106 | 7 | 67 | 2 | 52 | 4 | 100 | 2 |
| **V3** | 61 | 4 | 64 | 4 | 52 | 2 | 69 | 1 | 46 | 2 | 60 | 3 | 49 | 2 | 40 | 2 | 88 | 2 | 30 | 3 |
| **V4** | 46 | 5 | 35 | 1 | 26 | 2 | 30 | 3 | 31 | 5 | 31 | 3 | ∞ | 1 | 24 | 1 | 31 | 2 | 50 | 12 |
| **V5** | 36 | 3 | 30 | 2 | 55 | 5 | 39 | 4 | 31 | 1 | 32 | 2 | 47 | 4 | 35 | 3 | 40 | 3 | 40 | 1 |

**Tabla 3.18:** 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** |
| **V0** | ∞ | 2 | ∞ | 5 | ∞ | 0 | ∞ | 0 | ∞ | 1 | ∞ | 0 | ∞ | 7 | ∞ | 1 | 177 | 9 | ∞ | 1 |
| **V1** | 24 | 4 | 38 | 4 | ∞ | 0 | 30 | 7 | 50 | 4 | 28 | 0 | 29 | 2 | ∞ | 0 | 55 | 6 | 60 | 3 |
| **V2** | ∞ | 1 | 118 | 3 | 160 | 3 | 59 | 2 | 61 | 1 | 74 | 4 | 37 | 2 | 83 | 1 | ∞ | 0 | 47 | 2 |
| **V3** | 37 | 3 | ∞ | 0 | ∞ | 0 | 31 | 0 | 56 | 2 | ∞ | 0 | ∞ | 0 | ∞ | 1 | 42 | 2 | ∞ | 1 |
| **V4** | 24 | 1 | 28 | 2 | ∞ | 0 | 26 | 1 | 22 | 2 | 49 | 4 | 25 | 1 | 26 | 1 | 46 | 1 | 28 | 4 |
| **V5** | ∞ | 0 | 35 | 2 | 46 | 5 | ∞ | 0 | 47 | 4 | 30 | 3 | 58 | 7 | 37 | 4 | 26 | 2 | 25 | 2 |

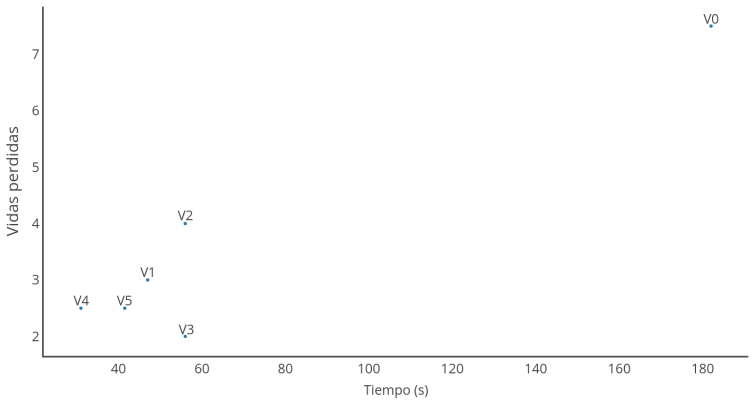



**Figura 3.47:** Gráfico con las medianas de la tabla 3.17.

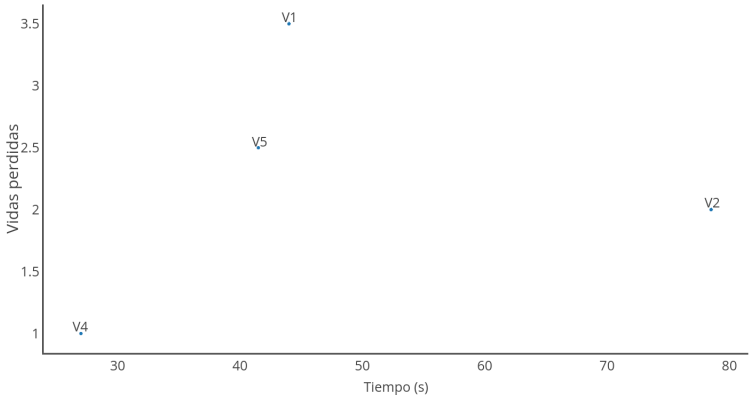



**Figura 3.48:** Gráfico con las medianas de la tabla 3.18.

### 3.10.5. Conclusiones

Como vemos en las pruebas, los resultados se han estabilizado respecto a las anteriores dos subversiones, consiguiendo resultados aceptables en todas las versiones. Es verdad que no conseguimos los mejores resultados en ninguno de las versiones de la presa, pero si nuestro objetivo es crear una version que pueda trabajar con cualquier inteligencia de la presa este parece el camino más adecuado.

### 3.10.6. A mejorar

En las pruebas vemos claramente que el sonido que hacen los cazadores es la base de la detección de los cazadores de la presa, por lo que intentaremos cambiar el alcance del ruido en la siguiente versión, intentando que la presa se acerque más a los cazadores.

## 3.11. Versión 1.8

En esta versión vamos a intentar que la presa se acerque más a los cazadores, haciendo que estos se muevan más despacio mientras protegen la comida, haciendo menos ruido. Como todas las anteriores versiones, probaremos la eficacia de esta nueva versión con todas las versiones de la presa.

### 3.11.1. Influencias

Como en la vida real, en esta versión vamos a hacer que al moverse más despacio los cazadores hagan menos ruido, facilitando el acercamiento de la presa. Con esto intentaremos reforzar objetivos de anteriores versiones, como intentar tenderle una trampa a la presa en zonas de comida.

### 3.11.2. Inteligencia

Vamos a trabajar con una inteligencia muy parecida a la anterior versión, solo que esta vez cambiaremos el ruido y la velocidad de los cazadores. Cuando los cazadores no tengan a la presa localizada se moverán a la mitad de la velocidad y harán un poco menos de la mitad del ruido, a la velocidad normal el ruido se expande por diez casillas y cuando se desplazan lento se expande cuatro casillas (Fig. 3.49, izquierda). Cuando uno de los dos cazadores vea o huela a la presa los dos cazadores volverán a la velocidad normal, siguiendo la misma inteligencia que en la anterior versión (Fig. 3.49, derecha).

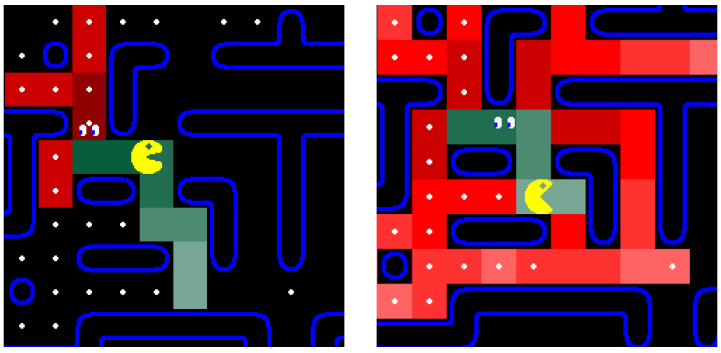


**Figura 3.49:** A la izquierda: alcance del ruido cuando el cazador se mueve despacio. A la derecha: alcance del ruido cuando el cazador se mueve rápido.

Podemos ver el diagrama de flujos de esta versión en la figura 3.50.

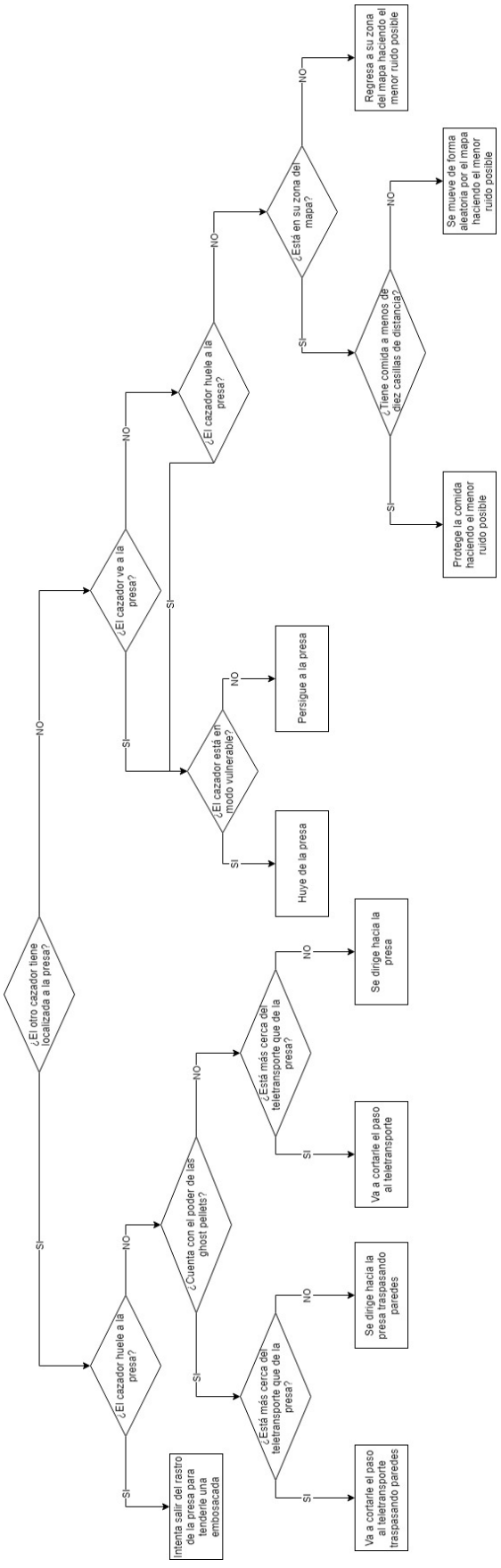



**Figura 3.50:** Diagrama de flujos de la versión 1.8.

### 3.11.3. Implementación

Para cambiar la inteligencia solo haremos 2 pequeños cambios a la versión anterior, como podemos ver en el código A.12.

Como vemos en el código hemos cambiado cosas en el archivo *ghost* y en el *main*. En el primero hemos cambiado parte del programa *Move*, para que cambie la velocidad del cazador dependiendo de si está persiguiendo a la presa o no. Además al final de la ejecución devolverá el valor de la velocidad actual del cazador. En el segundo hemos usado los cambios del archivo *ghost* para poder saber la velocidad del cazador, así pudiendo decidir el alcance del sonido: 4 si se desplaza lento y 10 si se desplaza normal.

### 3.11.4. Pruebas

Contaremos con dos tablas y dos gráficos: una tabla y un gráfico para el primer mapa, tabla 3.19 y figura 3.51, y lo mismo para el segundo, tabla 3.20 y figura 3.52.

**Tabla 3.19:** 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** |
| **V0** | 78 | 8 | 165 | 12 | 146 | 13 | 112 | 8 | 80 | 5 | 87 | 6 | 185 | 21 | 113 | 10 | 73 | 3 | 66 | 5 |
| **V1** | 26 | 2 | 22 | 2 | 28 | 2 | 46 | 3 | 30 | 3 | 27 | 3 | 25 | 1 | 28 | 2 | 29 | 2 | 46 | 5 |
| **V2** | 50 | 6 | 28 | 3 | 34 | 2 | 34 | 3 | 34 | 4 | 55 | 3 | 73 | 6 | 27 | 2 | 36 | 4 | 28 | 4 |
| **V3** | 25 | 2 | 35 | 5 | 32 | 1 | 28 | 3 | 41 | 3 | 38 | 5 | 35 | 2 | 46 | 3 | 28 | 2 | 59 | 6 |
| **V4** | 26 | 1 | 28 | 3 | 26 | 1 | 39 | 4 | 32 | 2 | 32 | 6 | 26 | 1 | 24 | 3 | 34 | 3 | 54 | 5 |
| **V5** | 52 | 4 | 32 | 2 | 31 | 2 | 34 | 0 | 48 | 3 | 35 | 2 | 25 | 2 | 28 | 2 | 34 | 4 | 34 | 0 |

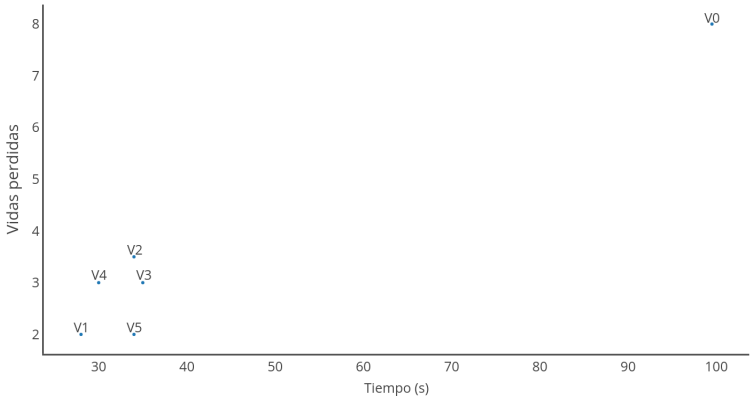



**Figura 3.51:** Gráfico con las medianas de la tabla 3.19.

**Tabla 3.20:** 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** |
| **V0** | 69 | 10 | 70 | 8 | 78 | 7 | 119 | 14 | 61 | 2 | 82 | 8 | 64 | 6 | 81 | 9 | 90 | 11 | 99 | 10 |
| **V1** | 21 | 3 | 19 | 2 | 25 | 2 | 21 | 3 | 27 | 3 | 21 | 3 | 33 | 5 | 39 | 5 | 21 | 3 | 21 | 3 |
| **V2** | 21 | 2 | 26 | 2 | 33 | 3 | 26 | 2 | 35 | 4 | 21 | 2 | 52 | 3 | 27 | 5 | 20 | 1 | 46 | 4 |
| **V3** | 28 | 2 | 28 | 2 | 25 | 2 | 18 | 1 | 30 | 4 | 28 | 5 | 24 | 1 | 34 | 2 | 21 | 2 | 27 | 2 |
| **V4** | 30 | 3 | 23 | 1 | 24 | 1 | 19 | 1 | 18 | 1 | 31 | 2 | 23 | 1 | 22 | 2 | 22 | 0 | 26 | 4 |
| **V5** | 30 | 2 | 19 | 1 | 27 | 2 | 22 | 1 | 24 | 0 | 23 | 2 | 25 | 2 | 22 | 2 | 21 | 2 | 34 | 1 |

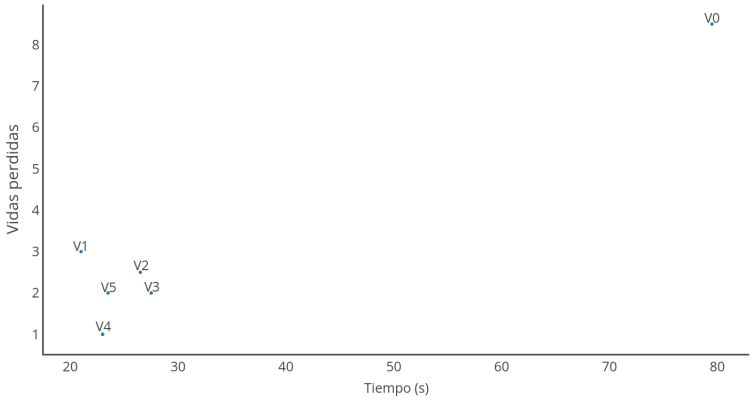



**Figura 3.52:** Gráfico con las medianas de la tabla 3.20.

### 3.11.5. Conclusiones

Podemos ver que los resultados de esta versión son ligeramente peores a la anterior versión. Esto se debe a un efecto colateral de moverse más despacio y así hacer menos ruido. Primero, al moverse más despacio los cazadores tardarán más tiempo en llegar a su objetivo, dejándole a la presa más tiempo para comerse la comida de la zona objetivo. Además, al hacer menos ruido la presa se acercará más hacia los cazadores, pero estos puede que no la vean igual, dejando como resultado una zona protegida más pequeña sin ningún tipo de ventaja.

### 3.11.6. A mejorar

Como hemos visto en las conclusiones de las pruebas, esta versión no mejora a la anterior, por lo que partiremos de esa para la siguiente versión. Esta vez intentaremos aplicarles una memoria a corto plazo para que recuerden la última posición conocida de la presa y se acerquen hacia allí.

## 3.12. Versión 1.9

En esta última versión con dos cazadores vamos a intentar que estos tengan memoria a corto plazo, acordándose de dónde han visto a la presa por última vez y acercándose a esa posición. Probaremos esta última version con todas las versiones de la presa.

### 3.12.1. Influencias

Si en la naturaleza un cazador ve una presa en la lejanía y le pierde el rastro intentará acercarse a esa última zona en la que ha visto a su presa para intentar recuperar el rastro. Esto es exactamente lo que se quiere conseguir en esta versión, haciendo que uno de los cazadores se acerque a la última zona en la que se ha visto a la presa para intentar recuperar el rastro.

### 3.12.2. Inteligencia

La inteligencia solo ha cambiado en un aspecto, ahora cuando uno de los cazadores vea a la presa a lo lejos se acercará hasta esa zona para intentar seguirle el rastro, en vez de perder el interés en el mismo momento en el que la deja de ver, como podemos ver en la figura 3.53.

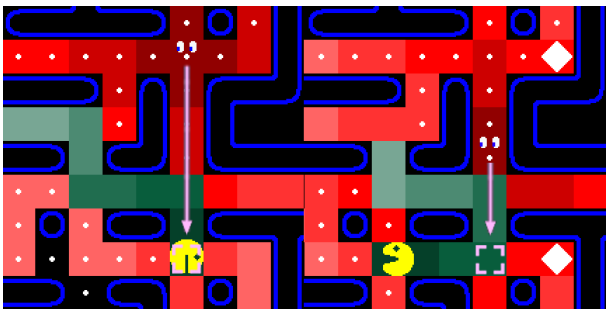


**Figura 3.53:** Uno de los cazadores ve a la presa y se acerca hasta ella aunque pierda el rastro.

El resto de la inteligencia será exactamente igual que en la versión 1.8.

Podemos ver el diagrama de flujos de esta versión en la figura 3.54.

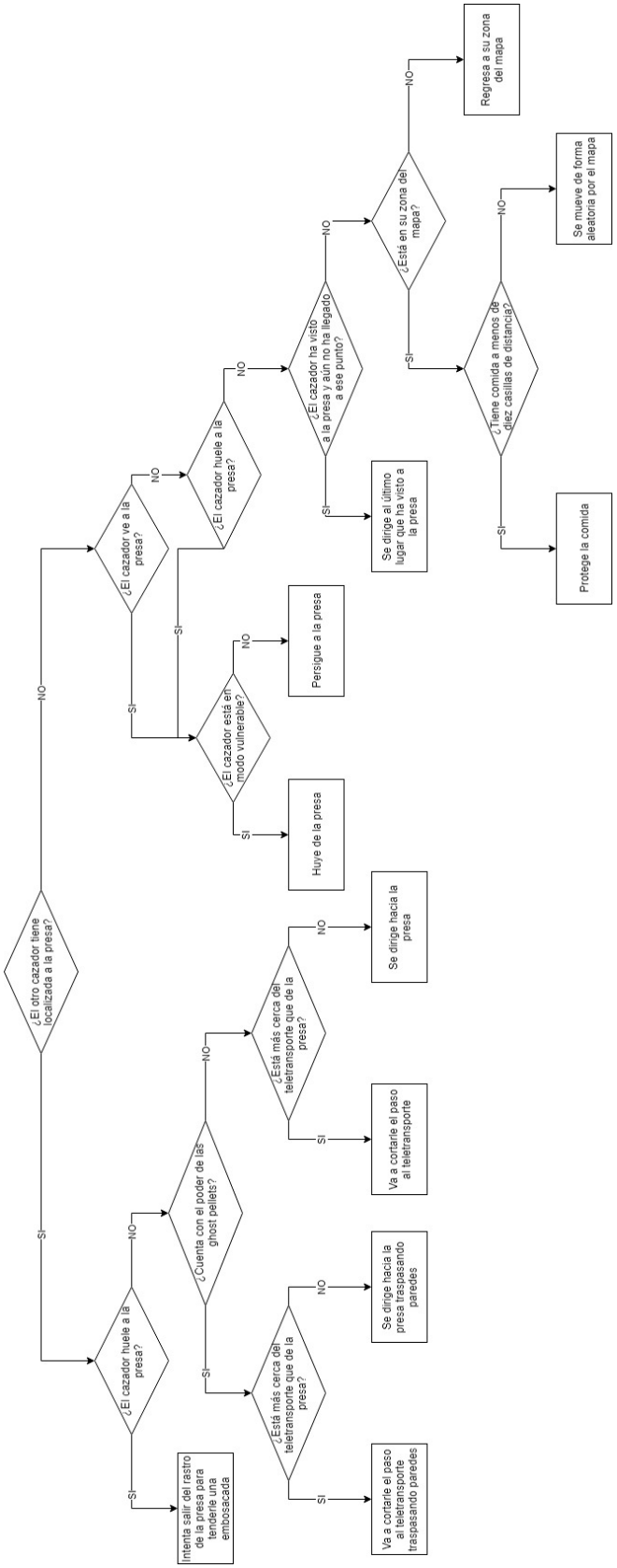



**Figura 3.54:** Diagrama de flujos de la versión 1.9.

### 3.12.3. Implementación

Se han cambiado pocas cosas en los archivos *main* y *ghost* para implementar la memoria a corto plazo. Se ha añadido la variable *visto* en la que guardaremos la última posición en la que cada uno de los cazadores ha visto a la presa hasta que se llegue a ese lugar.

Como vemos en el código A.13, se ha hecho que visto solo tome un valor a tener en cuenta cuando el cazador ve a la presa y se volverá a no ser relevante cuando llegue a esa posición o se teletransporte.

### 3.12.4. Pruebas

Contaremos con dos tablas y dos gráficos como en la mayoría de las anteriores versiones: una tabla y un gráfico para el primer mapa, tabla 3.21 y figura 3.55, y lo mismo para el segundo, tabla 3.22 y figura 3.56.

**Tabla 3.21:** 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** |
| **V0** | 182 | 13 | 160 | 14 | 298 | 17 | 123 | 7 | 97 | 2 | 204 | 7 | 241 | 20 | 185 | 13 | 101 | 9 | 169 | 11 |
| **V1** | 39 | 4 | 71 | 6 | 101 | 9 | 43 | 6 | 48 | 6 | 43 | 4 | 32 | 2 | 43 | 3 | 26 | 2 | 35 | 5 |
| **V2** | 50 | 4 | 36 | 2 | 52 | 5 | 64 | 5 | ∞ | 0 | 47 | 7 | 45 | 7 | 36 | 2 | 112 | 7 | 51 | 2 |
| **V3** | 135 | 6 | 59 | 3 | 41 | 4 | 76 | 6 | 38 | 3 | 55 | 2 | 32 | 1 | 34 | 2 | 46 | 4 | 58 | 3 |
| **V4** | 34 | 0 | 35 | 6 | 43 | 3 | 36 | 3 | 29 | 2 | 33 | 2 | 38 | 3 | 34 | 3 | 33 | 2 | 32 | 2 |
| **V5** | 45 | 4 | 29 | 1 | 48 | 7 | 36 | 3 | 37 | 3 | 51 | 4 | 27 | 2 | 25 | 2 | ∞ | 3 | 28 | 2 |

### 3.12.5. Conclusiones

Como podemos ver en las pruebas, esta es una de las mejores versiones de los fantasmas, aunque no sabemos con certeza cual es la mejor. Analizaremos los resultados generales en más profundidad en el apartado de las conclusiones del proyecto.

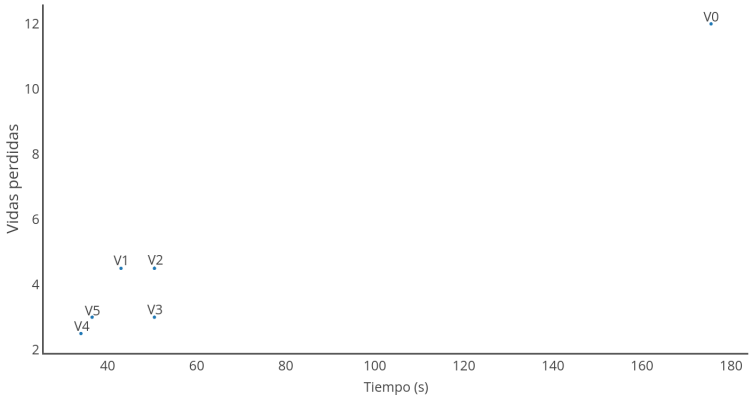



**Figura 3.55:** Gráfico con las medianas de la tabla 3.21.

**Tabla 3.22:** 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** |
| **V0** | ∞ | 2 | 118 | 17 | ∞ | 1 | 170 | 20 | 76 | 10 | ∞ | 4 | 226 | 10 | 110 | 18 | ∞ | 10 | 59 | 7 |
| **V1** | 26 | 3 | 128 | 4 | ∞ | 0 | ∞ | 3 | 31 | 3 | 27 | 4 | ∞ | 0 | 48 | 10 | 18 | 0 | ∞ | 3 |
| **V2** | 47 | 4 | 48 | 2 | 68 | 9 | ∞ | 0 | 84 | 7 | ∞ | 5 | ∞ | 5 | 49 | 7 | ∞ | 1 | 128 | 2 |
| **V3** | 69 | 0 | 22 | 3 | 22 | 0 | 74 | 2 | 138 | 10 | 32 | 2 | ∞ | 3 | 23 | 2 | ∞ | 3 | ∞ | 1 |
| **V4** | 27 | 2 | 28 | 5 | 22 | 2 | 26 | 1 | 22 | 1 | 28 | 4 | 27 | 0 | 34 | 9 | 34 | 1 | 22 | 1 |
| **V5** | 29 | 4 | ∞ | 0 | 20 | 0 | 25 | 2 | 22 | 1 | 36 | 3 | 25 | 2 | 48 | 6 | 28 | 1 | 27 | 1 |

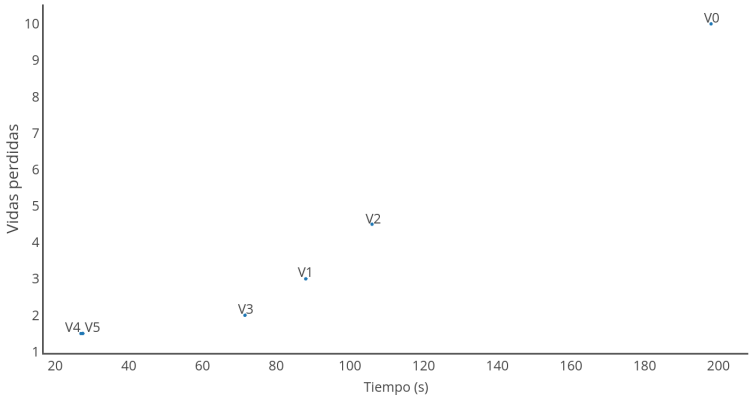



**Figura 3.56:** Gráfico con las medianas de la tabla 3.22.

## 3.13. Funcionamiento básico de las versiones de la presa del segundo modo

Vamos a analizar las versiones de la presa como con el primer modo de juego, analizando su funcionamiento básico y los mayores peligros que conllevan.

### 3.13.1. Versión 2.0

#### Funcionamiento

En esta primera versión lo único nuevo que encontraremos respecto al anterior modo de juego es que ahora hay dos presas en lugar de una. Las dos presas tendrán la misma inteligencia y no tendrán ningún tipo de comunicación entre ellos.

#### Peligros

El mayor peligro de esta versión es la aleatoriedad, ya que puede que las dos presas se dividan el mapa perfectamente haciendo que coman toda la comida en muy poco tiempo, pero esta también es su mayor desventaja, ya que puede que las dos presas trabajen en el mismo sitio, no aprovechando que son dos.

### 3.13.2. Versión 2.1

#### Funcionamiento

En esta versión se mejora la versión anterior haciendo que las presas se dividan la zona de acción, haciendo que trabajen en la misma zona el menor tiempo posible.

#### Peligros

Al dividirse el mapa las presas comerán la comida en muy poco tiempo, dejando poco tiempo a los cazadores para atraparlas.

### 3.13.3. Versión 2.2

#### Funcionamiento

En esta versión se implementa la mecánica de que las presas se puedan salvar entre ellas una vez sean capturadas. Cuando uno de los cazadores atrape a una de las presas se quedará inmóvil en el sitio, como podemos ver en la figura 3.57.

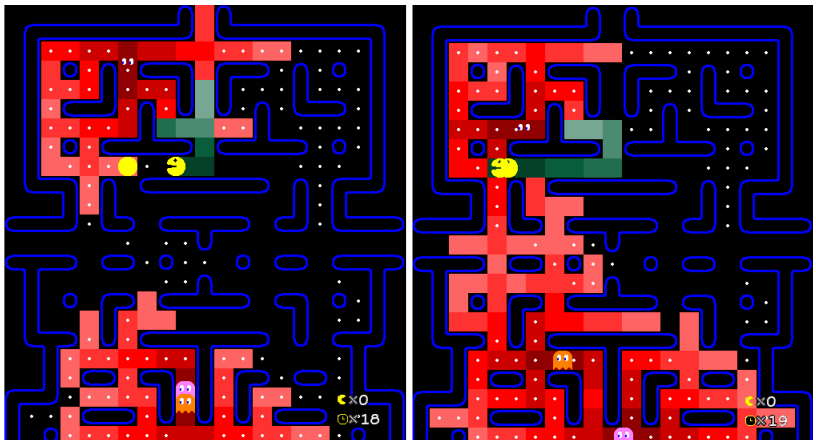


**Figura 3.57:** La presa se acerca a su compañera capturada y la salva.

Esto solo pasará de forma aleatoria, ya que las presas no tienen forma de saber si su compañera esta atrapada o no.

#### Peligros

El mayor peligro de esta versión es que las presas pueden salvarse entre ellas, haciendo que no pierdan ni una sola vez. Este peligro no es muy grande ya que solo se salvarán de forma aleatoria.

### 3.13.4. Versión 2.3

#### Funcionamiento

Esta versión les facilitará la posición de su compañera muerta a las presas, haciendo que sea más fácil salvarse entre ellas.

#### Peligros

El peligro es el mismo que en la anterior versión, pero esta vez es mucho mayor, ya que las presas siempre intentarán salvar a su compañera, haciendo que si los cazadores no defienden la zona donde esta atrapada la presa les sea muy fácil salvarse.

### 3.13.5. Versión 2.4

#### Funcionamiento

Esta vez las presas también sabrán donde está su compañera atrapada pero le darán prioridad a comer en lugar de salvarla, ya que es posible que haya cazadores protegiendo la zona.

#### Peligros

Al hacer que las presas no tengan prioridad a salvar a su compañera hará que pierda utilidad que un cazador defienda la zona donde una de las presas ha sido cazada.

## 3.14. Versión 2.0

Esta será la primera versión del segundo modo de juego, en el que habrá tres cazadores y dos presas. Partiremos de la última inteligencia creada en la primera versión e intentaremos amoldarla a esta, haciendo los cambios necesarios. Al hacerla funcionar la probaremos con todas las versiones de la presa de este modo de juego.

### 3.14.1. Influencias

Para crear esta inteligencia vamos a basarnos sobre todo en la versión 1.9, pero en esta solo había dos fantasmas. Para la inteligencia del tercero vamos a recuperar el buscador de versiones anteriores.

### 3.14.2. Inteligencia

Como se ha mencionado en el apartado anterior, la mayor parte de la inteligencia sera igual que en la versión 1.9, con un cazador protegiendo la zona de arriba y otro protegiendo la zona de abajo, pero cómo en esta versión tenemos 3 cazadores en lugar de dos vamos a recuperar un rol de anteriores versiones, el buscador. Este recorrerá el laberinto de forma aleatoria mientras los otros dos cazadores protegen sus respectivas zonas del mapa (Fig. 3.58). Al perseguir a las presas intentaremos atrapar primero a una y luego ir a por la otra, si se tienen a las dos localizadas se perseguirá solo a una, dejando a la otra libre.

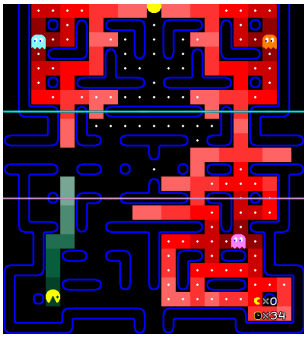


**Figura 3.58:** El cazador azul protege la zona alta del mapa, el cazador rosa protege la zona baja del mapa y el cazador naranja deambula por todo el mapa.

Podemos ver el diagrama de flujos de esta versión en la figura 3.59.

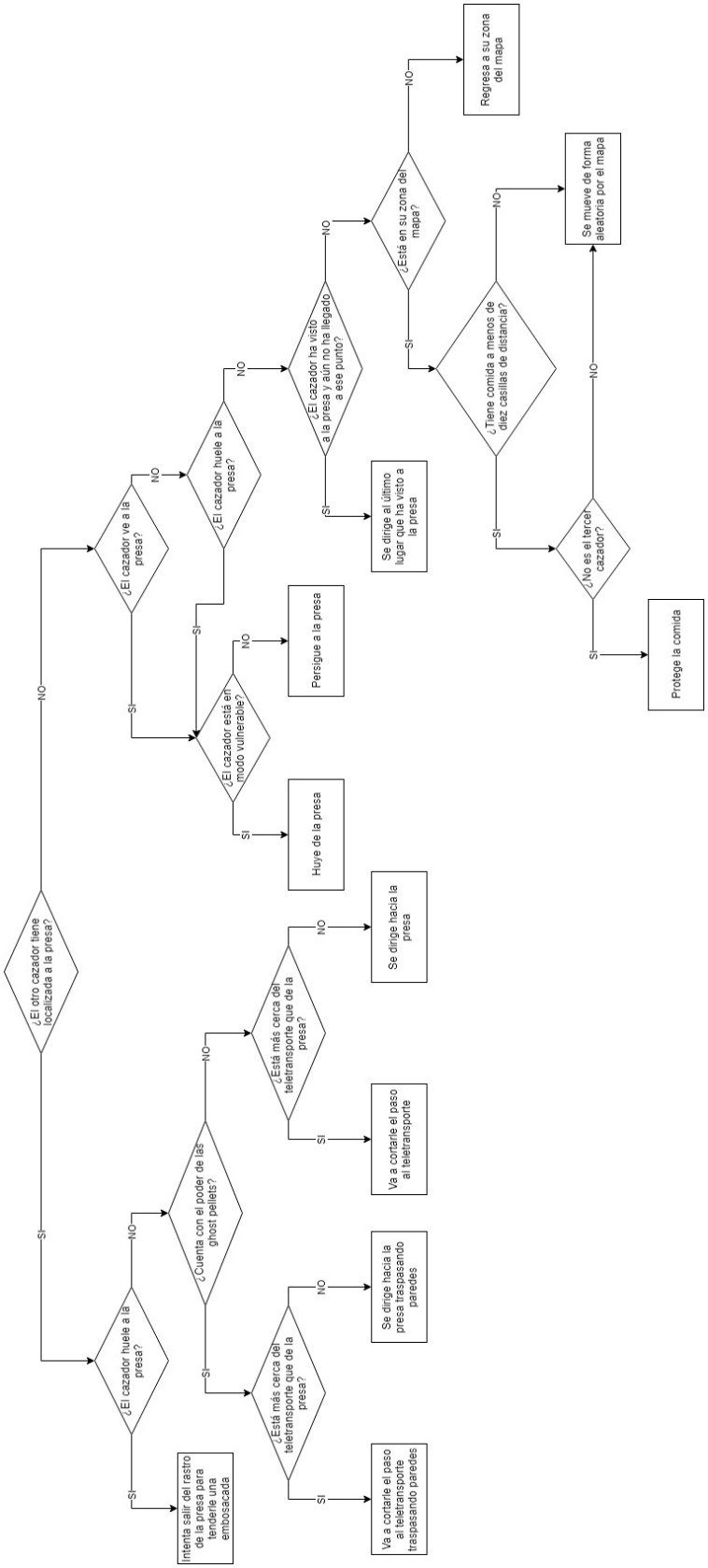



**Figura 3.59:** Diagrama de flujos de la versión 2.0.

### 3.14.3. Implementación

Esta versión no tiene grandes cambios respecto a la 1.9, salvo que en esta ocasión hay dos presas a las que ver y oler, por lo que las variables en las que se guardaban estos datos pasarán a ser vectores con dos posiciones. Además ahora contamos con un tercer fantasma, al que también debemos escuchar cuando vea a las presas. Estos cambios los podemos ver en el código A.14.

Como se puede ver en el código, los cambios más grandes respecto a la versión 1.9 son que ahora hay dos rastros de olor y dos presas a las que ver y que ahora la comunicación es entre tres cazadores. Esta vez los cazadores tendrán que mirar si alguno de sus otros dos compañeros ha visto a una de las presas, y en el caso de que los dos vean a una presa se seguirán las indicaciones del primer cazador en orden numérico. En el caso de que el mismo cazador vea o huela a las dos presas se centrará solo en la primera.

### 3.14.4. Pruebas

Contaremos con dos tablas y dos gráficos como en el primer modo: una tabla y un gráfico para el primer mapa, tabla 3.23 y figura 3.60, y lo mismo para el segundo, tabla 3.24 y figura 3.61.

**Tabla 3.23:** 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** | 29 | 1 | 22 | 1 | 30 | 3 | 19 | 1 | 35 | 2 | 17 | 0 | 25 | 3 | 23 | 1 | 37 | 3 | 21 | 1 |
| **V1** | 19 | 1 | 18 | 1 | 24 | 1 | 20 | 1 | 45 | 4 | 24 | 2 | 18 | 0 | 23 | 1 | 25 | 1 | 21 | 1 |
| **V2** | 23 | 0 | 24 | 1 | 21 | 1 | 18 | 1 | 21 | 1 | 40 | 4 | 26 | 1 | 41 | 6 | 28 | 2 | 27 | 2 |
| **V3** | 22 | 0 | 23 | 0 | 21 | 1 | 33 | 2 | 21 | 2 | 30 | 1 | 36 | 4 | 24 | 1 | 29 | 3 | 24 | 0 |
| **V4** | 34 | 1 | 21 | 1 | 17 | 0 | 22 | 1 | 16 | 1 | 23 | 2 | 24 | 1 | 33 | 3 | 27 | 2 | 20 | 2 |

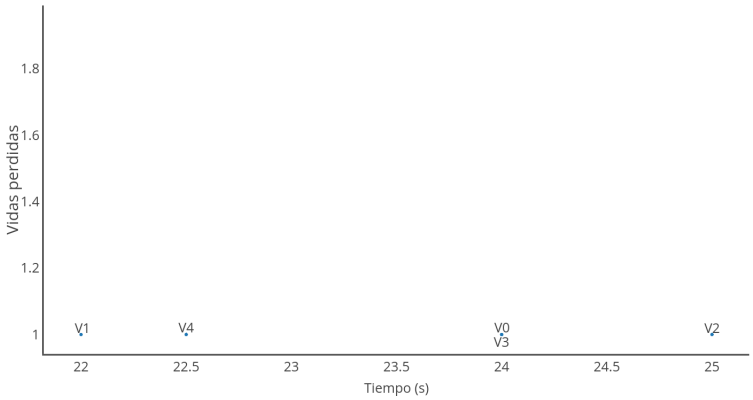



**Figura 3.60:** Gráfico con las medianas de la tabla 3.23.

**Tabla 3.24:** 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** | 31 | 5 | 13 | 0 | 24 | 2 | 18 | 1 | 18 | 1 | 16 | 1 | 42 | 5 | 21 | 2 | 16 | 1 | 18 | 0 |
| **V1** | 19 | 1 | 15 | 1 | 23 | 4 | 15 | 1 | 12 | 0 | 23 | 2 | 15 | 1 | 24 | 1 | 21 | 2 | 25 | 2 |
| **V2** | 24 | 1 | 25 | 2 | 14 | 1 | 20 | 2 | 25 | 1 | 16 | 1 | 25 | 1 | 19 | 1 | 16 | 1 | 20 | 2 |
| **V3** | 31 | 1 | 20 | 1 | 20 | 0 | 17 | 0 | 26 | 1 | 19 | 0 | 25 | 1 | 26 | 0 | 35 | 2 | 28 | 2 |
| **V4** | 23 | 1 | 22 | 1 | 22 | 1 | 16 | 0 | 22 | 0 | 19 | 0 | 17 | 2 | 17 | 2 | 18 | 0 | 21 | 2 |

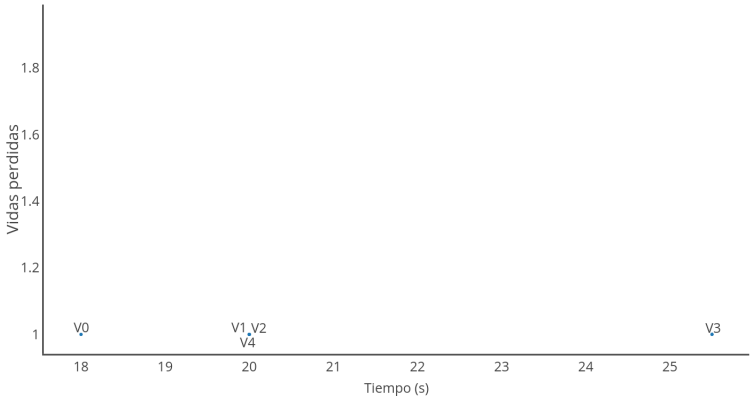



**Figura 3.61:** Gráfico con las medianas de la tabla 3.24.

### 3.14.5. Conclusiones

Como vemos en los resultados, el número de veces que la presa es atrapada es muy pequeña, la mediana es de uno con todas las versiones, y el tiempo necesario también es muy poco, pero esto es normal ya que es la primera versión que se implementa para este modo.

### 3.14.6. A mejorar

En esta primera versión no tenemos ninguna forma de gestionar los rescates entre las presas, por lo que en la siguiente versión nos centraremos en esto.

## 3.15. Versión 2.1

En esta nueva versión vamos a gestionar los rescates entre las presas, intentando que se salven el menor número de veces posible y lo probaremos con todas las versiones de la presa.

### 3.15.1. Influencias

Vamos a seguir con la ya mencionada defensa en zona para esta versión, solo que esta vez en lugar de defender la comida defenderemos la presa atrapada. El cazador encargado de esto será el vigilante, que rondará la zona donde está la presa capturada esperando la aparición de su compañera.

### 3.15.2. Inteligencia

Como ya hemos nombrado en el anterior apartado vamos a añadir una nueva inteligencia, el vigilante. Esta inteligencia se activará en el cazador naranja cuando uno de las dos presas esté capturada y la otra no este localizada. Podemos ver el comportamiento en la figura 3.62.

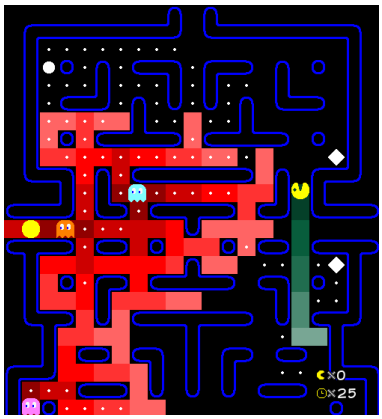



**Figura 3.62:** El cazador azul protege la zona alta del mapa, el cazador rosa protege la zona baja del mapa y el cazador naranja protege la zona en la que hay una presa atrapada.

El resto del tiempo el comportamiento de los cazadores será exactamente igual que en la anterior versión.

Podemos ver el diagrama de flujos de esta versión en la figura 3.63.

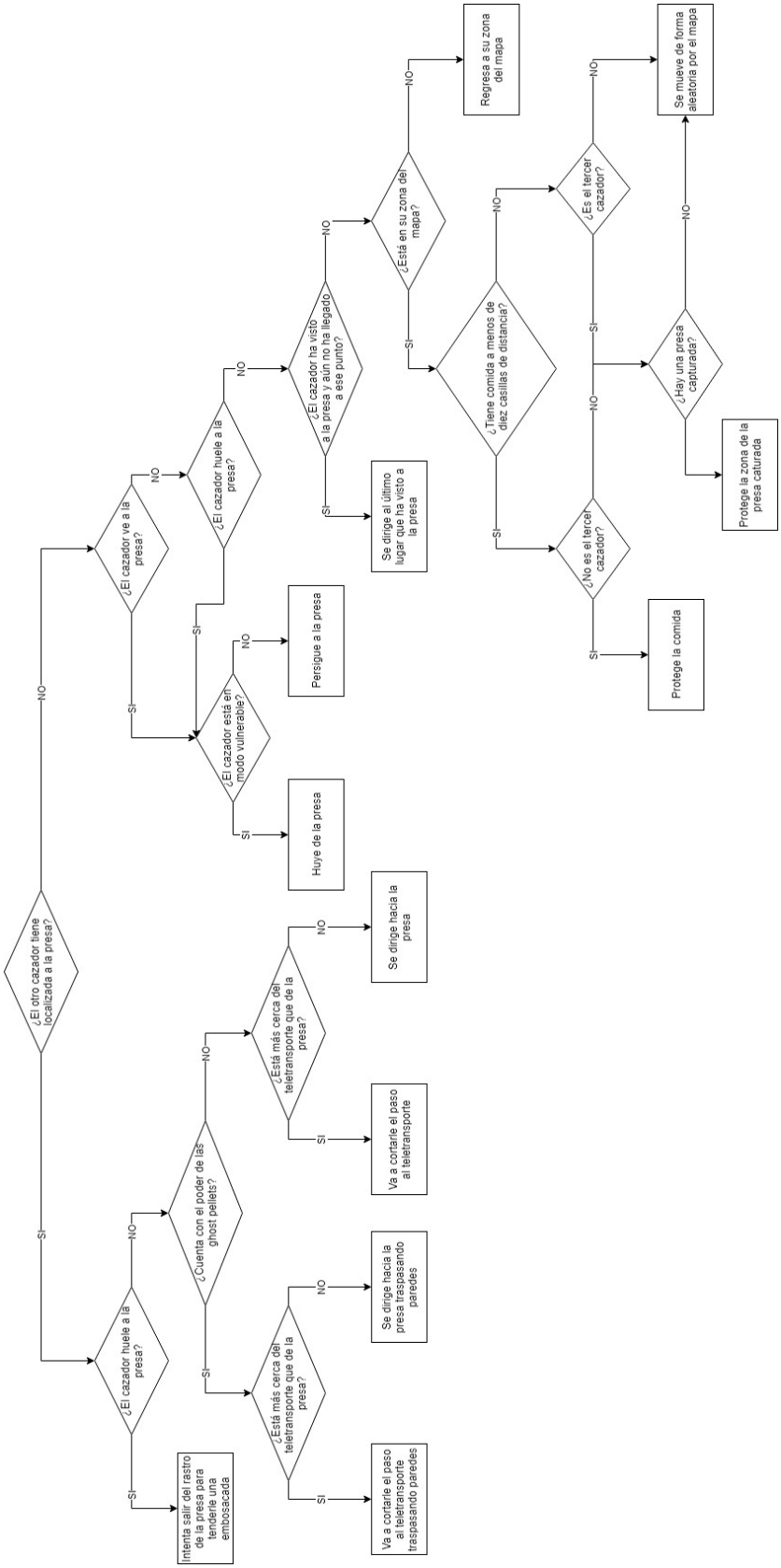



**Figura 3.63:** Diagrama de flujos de la versión 2.1.

### 3.15.3. Implementación

Vamos a añadir una nueva variable, *Dead*, que se usará para que el cazador naranja vaya a la posición marcada cuando uno de las presas haya sido atrapada. No se añadirá mucho código nuevo, ya que reutilizaremos código viejo que usamos paraque los cazadores se acerque a una casilla concreta, como podemos ver en el código A.15.

Cómo podemos ver en el código la única parte nueva es donde decidimos cual de las dos presas es la que hemos atrapado y por lo tanto a donde se tiene que acercar el vigilante.

### 3.15.4. Pruebas

Contaremos con dos tablas: una tabla y un gráfico para el primer mapa, tabla 3.25 y figura 3.64, y lo mismo para el segundo, tabla 3.26 y figura 3.65.

**Tabla 3.25:** 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** | 25 | 2 | 40 | 4 | 31 | 0 | 26 | 2 | 44 | 4 | 27 | 2 | 56 | 5 | 40 | 4 | 23 | 1 | 43 | 6 |
| **V1** | 26 | 2 | 30 | 2 | 34 | 2 | 31 | 2 | 22 | 1 | 23 | 3 | 25 | 3 | 22 | 1 | 19 | 0 | 19 | 0 |
| **V2** | 21 | 2 | 23 | 2 | 17 | 0 | 21 | 2 | 33 | 2 | 43 | 4 | 24 | 3 | 43 | 3 | 24 | 1 | 19 | 1 |
| **V3** | 17 | 1 | 28 | 1 | 18 | 1 | 33 | 3 | 28 | 2 | 31 | 2 | 38 | 1 | 27 | 2 | 28 | 2 | 36 | 3 |
| **V4** | 20 | 1 | 17 | 1 | 22 | 1 | 21 | 1 | 20 | 1 | 25 | 1 | 31 | 2 | 22 | 2 | 20 | 1 | 29 | 3 |

**Tabla 3.26:** 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** | 13 | 0 | 14 | 1 | 22 | 1 | 20 | 1 | 18 | 2 | 20 | 0 | 28 | 4 | 21 | 2 | 19 | 1 | 28 | 3 |
| **V1** | 12 | 1 | 24 | 2 | 18 | 2 | 32 | 3 | 37 | 2 | 30 | 3 | 13 | 0 | 35 | 5 | 30 | 1 | 19 | 1 |
| **V2** | 14 | 0 | 20 | 1 | 24 | 1 | 14 | 0 | 13 | 0 | 22 | 1 | 18 | 1 | 13 | 1 | 23 | 2 | 18 | 2 |
| **V3** | 20 | 1 | 24 | 3 | 24 | 1 | 18 | 0 | 19 | 0 | 14 | 1 | 15 | 1 | 18 | 0 | 22 | 3 | 16 | 0 |
| **V4** | 24 | 0 | 18 | 1 | 22 | 1 | 17 | 1 | 27 | 1 | 15 | 1 | 17 | 1 | 20 | 0 | 17 | 2 | 20 | 2 |

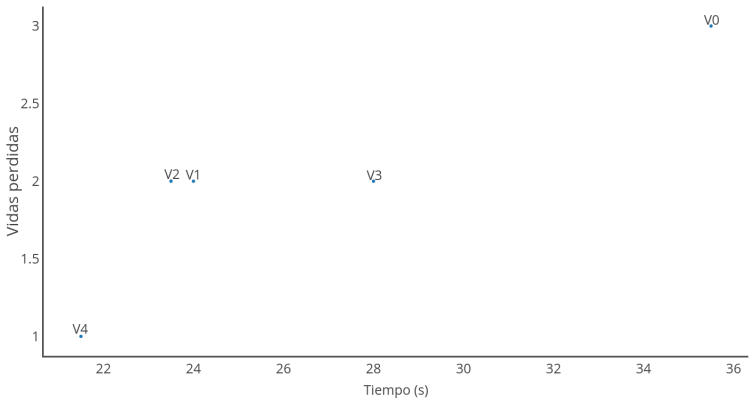



**Figura 3.64:** Gráfico con las medianas de la tabla 3.25.

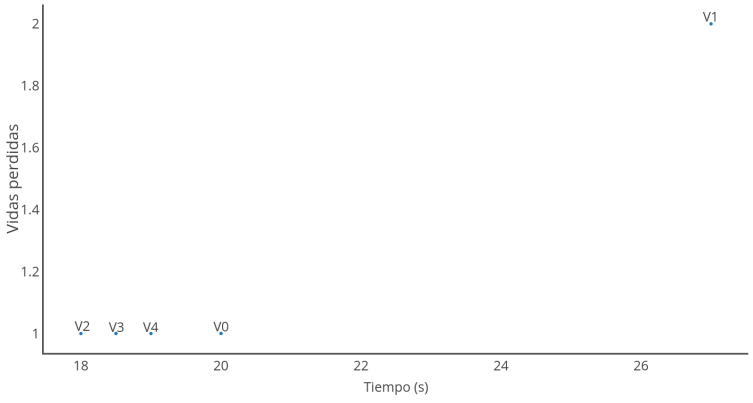



**Figura 3.65:** Gráfico con las medianas de la tabla 3.26.

### 3.15.5. Conclusiones

Esta vez podemos ver una mejora importante en el primer nivel respecto a la anterior versión. Aunque en el segundo nivel la mejora no es tan grande, se han conseguido mejores resultados contra la V1 de la presa, por lo que podemos ver que en este nivel también hay mejoras.

### 3.15.6. A mejorar

En las dos primeras versiones vemos que cuando las dos presas están localizadas el comportamiento de los cazadores no es lo mas óptimo posible, dejando escapar algunas veces a una presa casi atrapada para dirigirse hacia la otra. En la siguiente versión intentaremos gestionar esto.

## 3.16. Versión 2.2

En esta versión vamos a intentar que los cazadores se centren en una presa antes de ir a por la otra y lo probaremos con todas las versiones de la presa.

### 3.16.1. Influencias

Esta vez vamos a intentar establecer prioridades. Primero nos centraremos en uno de los objetivos antes de ir a por el otro, ya que entre 3 cazadores no podemos cazar dos presas a la vez.

### 3.16.2. Inteligencia

Para saber a que presa deben dirigirse vamos a hacer que los cazadores se comuniquen al ver una de las presas. Cuando uno de los cazadores vea una presa todos los cazadores irán a por esa presa dejando a la otra hacer lo que quiera. Al hacer esto se generarán dos situaciones de 3 para 1, facilitando las capturas.

Podemos ver el diagrama de flujos de esta versión en la figura 3.66.

### 3.16.3. Implementación

Vamos a añadir una nueva variable, *objetivo*, con la que marcaremos el objetivo actual de los cazadores. En el código A.16 podemos ver como se actualiza esta variable.

Como podemos ver en el código, hemos añadido la variable a la función *Move* del archivo *ghost*, actualizándola cuando uno de los cazadores ve una presa, actualizando su objetivo y el de todos los demás cazadores. En el momento que la pierdan de vista o la capturen volverán a no tener objetivo.

### 3.16.4. Pruebas

Contaremos con dos tablas: una tabla y un gráfico para el primer mapa, tabla 3.27 y figura 3.67, y lo mismo para el segundo, tabla 3.28 y figura 3.68.

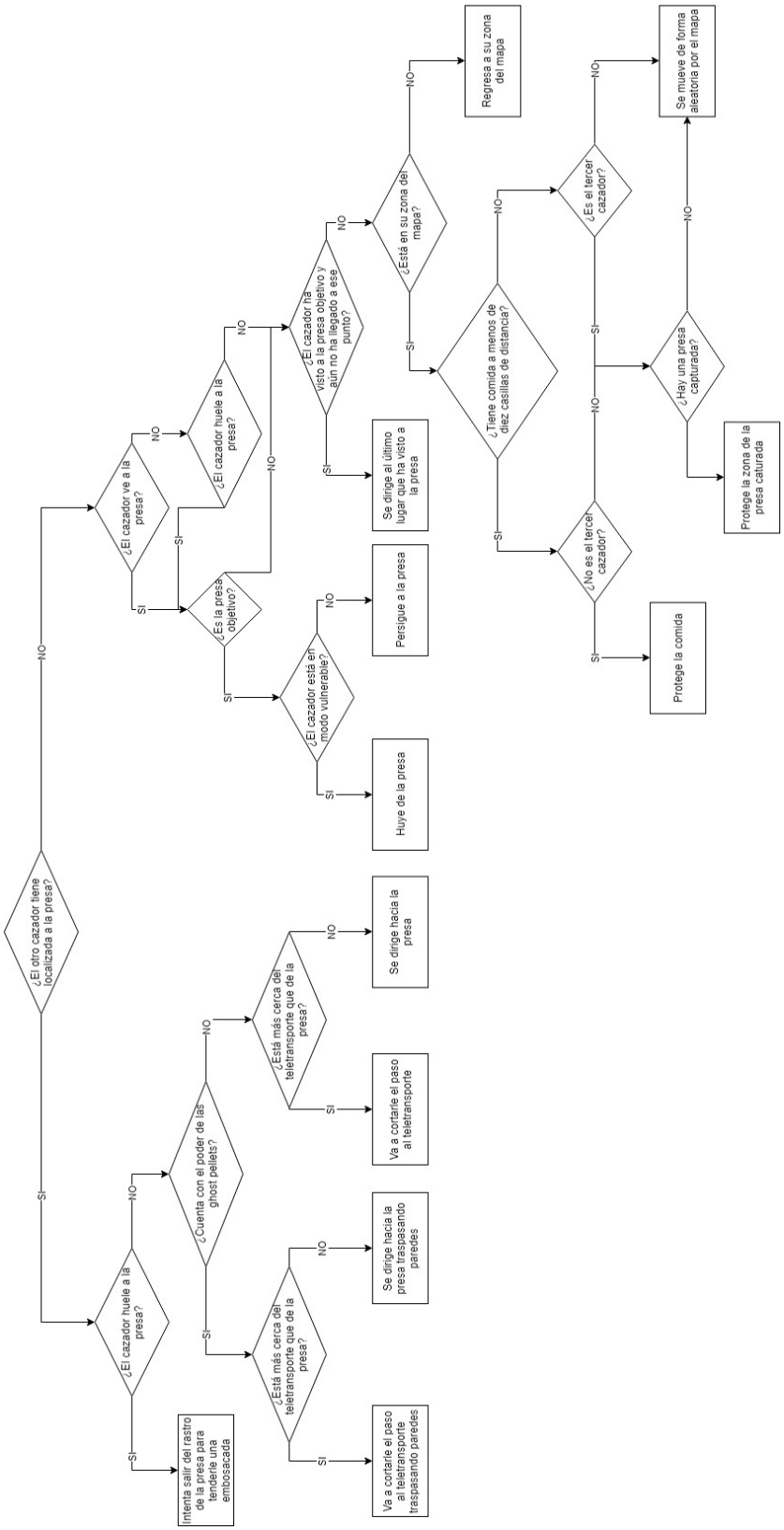



**Figura 3.66:** Diagrama de flujos de la versión 2.2.

**Tabla 3.27:** 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** | 18 | 1 | 45 | 4 | 21 | 1 | 42 | 1 | 42 | 3 | 19 | 2 | 26 | 2 | 31 | 2 | 30 | 3 | 38 | 2 |
| **V1** | 24 | 3 | 50 | 6 | 25 | 1 | 29 | 4 | 34 | 1 | 17 | 0 | 22 | 1 | 75 | 5 | 24 | 3 | 55 | 5 |
| **V2** | 24 | 3 | 27 | 1 | 27 | 1 | 26 | 2 | 25 | 2 | 27 | 1 | 31 | 2 | 24 | 1 | 27 | 2 | 30 | 2 |
| **V3** | 17 | 0 | 21 | 2 | 44 | 5 | 35 | 2 | 42 | 2 | 34 | 2 | 27 | 2 | 25 | 2 | 19 | 1 | 45 | 2 |
| **V4** | 30 | 3 | 22 | 1 | 24 | 0 | 36 | 3 | 24 | 1 | 24 | 2 | 33 | 2 | 34 | 2 | 20 | 1 | 22 | 2 |

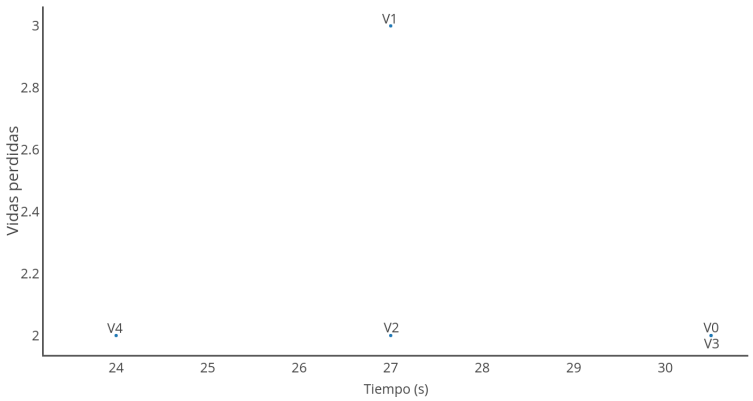



**Figura 3.67:** Gráfico con las medianas de la tabla 3.27.

**Tabla 3.28:** 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** | 16 | 1 | 26 | 4 | 15 | 1 | 29 | 2 | 20 | 2 | 20 | 1 | 38 | 3 | 14 | 0 | 20 | 2 | 47 | 3 |
| **V1** | 19 | 1 | 16 | 1 | 25 | 1 | 23 | 2 | 44 | 4 | 18 | 1 | 16 | 1 | 21 | 2 | 20 | 1 | 17 | 1 |
| **V2** | 15 | 0 | 20 | 0 | 15 | 2 | 23 | 2 | 29 | 3 | 22 | 3 | 31 | 0 | 25 | 1 | 18 | 0 | 25 | 2 |
| **V3** | 24 | 2 | 40 | 3 | 15 | 1 | 55 | 7 | 22 | 1 | 15 | 0 | 24 | 3 | 24 | 3 | 19 | 1 | 33 | 1 |
| **V4** | 17 | 1 | 19 | 0 | 21 | 2 | 13 | 0 | 21 | 2 | 21 | 1 | 18 | 2 | 21 | 1 | 15 | 0 | 18 | 1 |

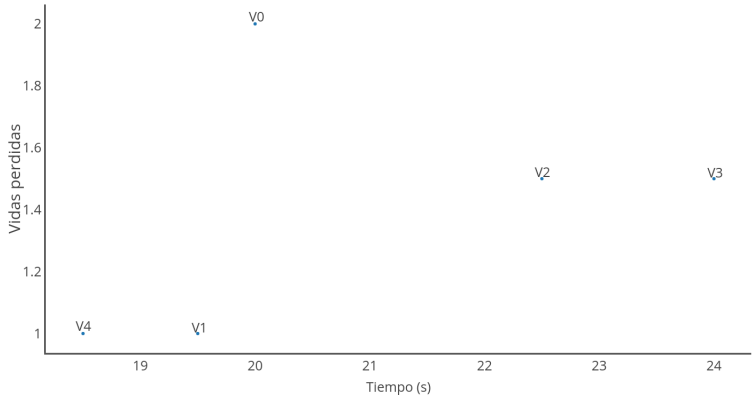



**Figura 3.68:** Gráfico con las medianas de la tabla 3.28.

### 3.16.5. Conclusiones

En esta versión podemos ver mejora en los dos niveles disponibles respecto a las anteriores versiones, sobre todo en el segundo nivel. El tiempo necesario sigue siendo bastante menor que en el anterior modo de juego, pero esto es normal ya que esta vez tenemos que lidiar con dos presas en lugar de con una.

### 3.16.6. A mejorar

Cuando uno de los cazadores tiene localizada a una de las presas es posible que los otros cazadores se dirijan al mismo sitio, perdiendo la ventaja que da tener 3 cazadores. En la siguiente versión se intentará mejorar esto, haciendo que uno de los cazadores se dirija a hacia la presa y otro hacia el teletransporte.

## 3.17. Versión 2.3

En esta última version vamos a intentar que los cazadores se separen por el mapa para intentar predecir todos los caminos que pueda tomar la presa y lo probaremos con todas sus versiones.

### 3.17.1. Influencias

Esta vez vamos a intentar mejorar al máximo el trabajo en equipo, intentando cubrir el máximo terreno posible entre todos los cazadores.

### 3.17.2. Inteligencia

Para decidir cual de los dos cazadores restantes irá al teletransporte y cual hacia la presa compararemos las distancias de los dos al teletransporte objetivo, haciendo que vaya hacia él el que este mas cerca y el otro hacia la presa, como podemos ver en la figura 3.69.

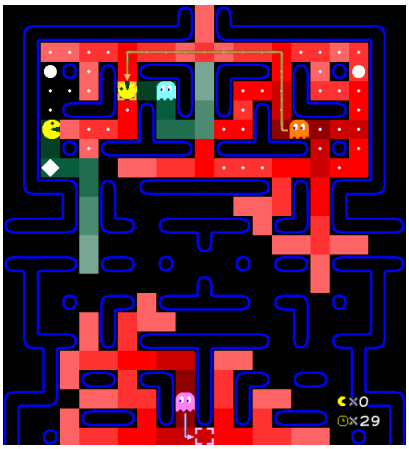



**Figura 3.69:** El cazador naranja va hacia la presa mientras que el rosa va hacia el teletransporte.

El resto de la inteligencia será exactamente igual a la anterior versión.

Podemos ver el diagrama de flujos de esta versión en la figura 3.70.

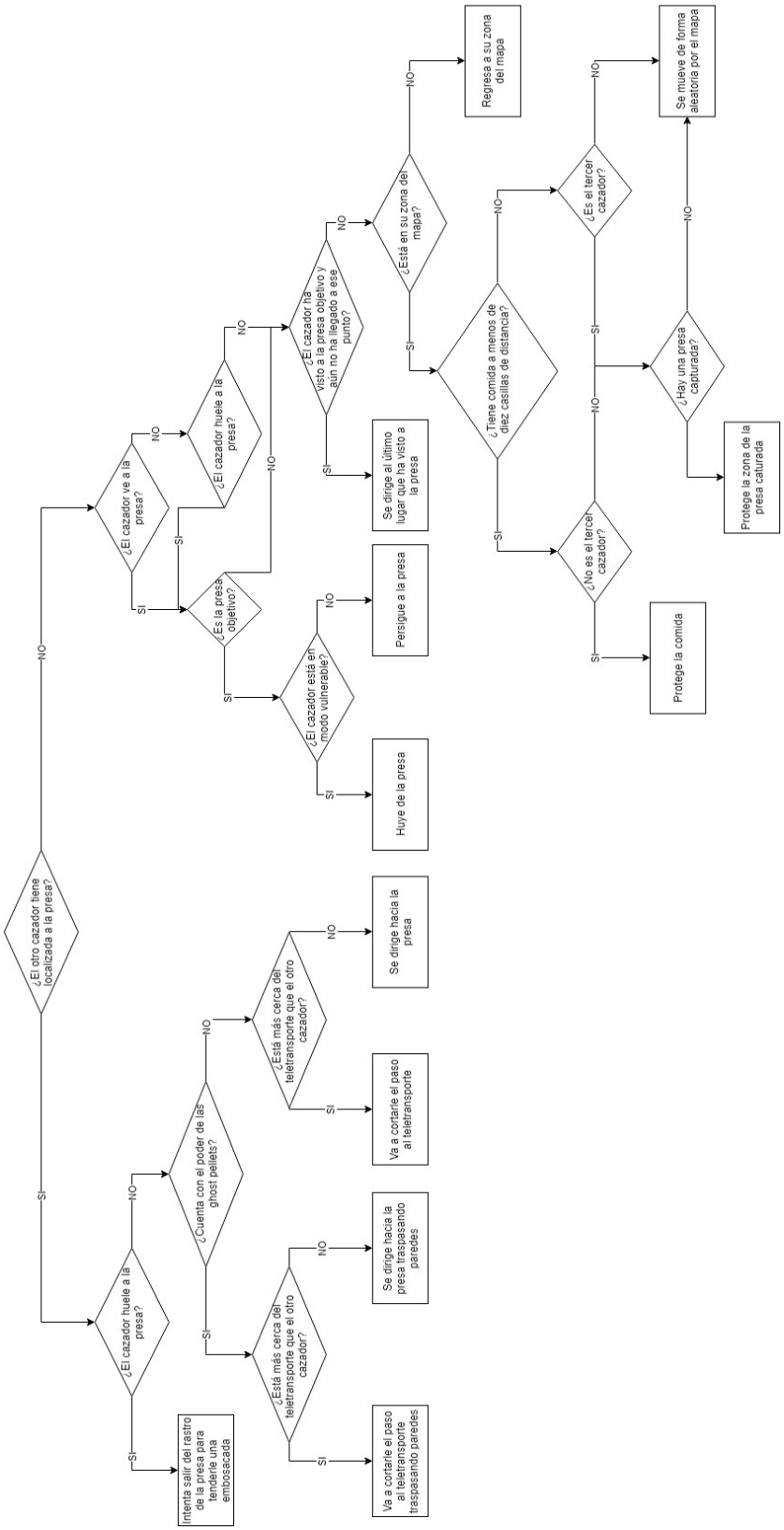



**Figura 3.70:** Diagrama de flujos de la versión 2.3.

### 3.17.3. Implementación

Para hacer esta inteligencia vamos a crear dos nueva variable, *posicion* y *camino*, donde guardaremos las posiciones de los tres cazadores. Así podremos comparar las distancias de los cazadores al teletransporte objetivo, decidiendo cual de los dos tiene que moverse hacia la presa y cual hacia el teletransporte. La implementación la podemos ver en el código A.17.

En el código podemos ver que no hay grandes cambios en la implementación, solo cambiaremos la comunicación entre los cazadores, haciendo que se muevan según la distancia que tengan al teletransporte respecto al otro cazador.

### 3.17.4. Pruebas

Contaremos con dos tablas y dos gráficos como en el primer modo: una tabla y un gráfico para el primer mapa, tabla 3.29 y figura 3.71, y lo mismo para el segundo, tabla 3.30 y figura 3.72.

**Tabla 3.29:** 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** | 31 | 3 | 17 | 1 | 24 | 2 | 38 | 4 | 19 | 2 | 33 | 1 | 27 | 1 | 24 | 2 | 18 | 1 | 30 | 2 |
| **V1** | 37 | 4 | 38 | 2 | 24 | 2 | 56 | 3 | 35 | 1 | 19 | 1 | 63 | 9 | 33 | 2 | 21 | 2 | 22 | 1 |
| **V2** | 36 | 2 | 26 | 2 | 19 | 2 | 18 | 1 | 15 | 1 | 23 | 2 | 30 | 3 | 28 | 2 | 54 | 3 | 25 | 1 |
| **V3** | 22 | 2 | 28 | 1 | 20 | 2 | 24 | 2 | 19 | 1 | 25 | 1 | 36 | 3 | 24 | 1 | 19 | 0 | 45 | 3 |
| **V4** | 22 | 1 | 28 | 1 | 19 | 2 | 29 | 2 | 36 | 2 | 23 | 2 | 28 | 3 | 42 | 2 | 66 | 5 | 24 | 0 |

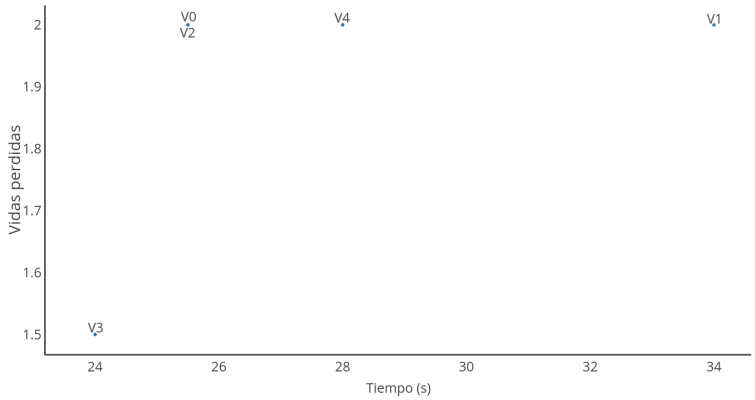



**Figura 3.71:** Gráfico con las medianas de la tabla 3.29.

**Tabla 3.30:** Resultados de la versión 3 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** | 28 | 2 | 18 | 1 | 25 | 2 | 14 | 1 | 26 | 2 | 14 | 2 | 22 | 3 | 21 | 3 | 23 | 1 | 20 | 2 |
| **V1** | 19 | 1 | 29 | 1 | 23 | 2 | 16 | 1 | 24 | 1 | 29 | 1 | 20 | 1 | 17 | 1 | 21 | 3 | 27 | 1 |
| **V2** | 23 | 3 | 24 | 3 | 24 | 3 | 19 | 2 | 17 | 1 | 18 | 0 | 34 | 2 | 27 | 2 | 16 | 0 | 20 | 2 |
| **V3** | 23 | 0 | 25 | 2 | 25 | 1 | 15 | 1 | 13 | 1 | 16 | 1 | 13 | 0 | 13 | 0 | 20 | 1 | 10 | 0 |
| **V4** | 29 | 1 | 18 | 1 | 17 | 2 | 21 | 1 | 21 | 1 | 18 | 1 | 15 | 1 | 15 | 0 | 23 | 0 | 19 | 1 |

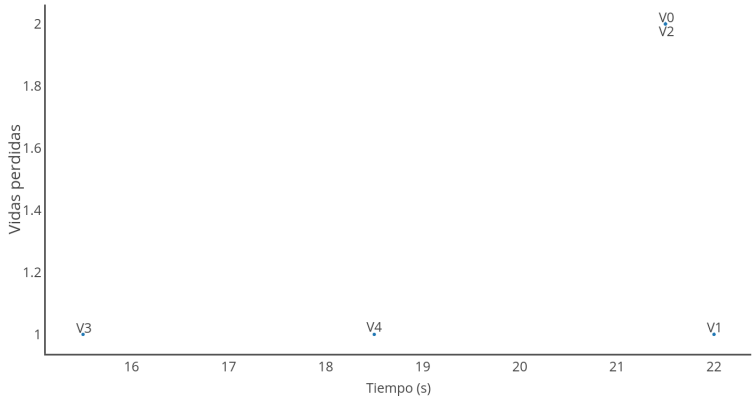



**Figura 3.72:** Gráfico con las medianas de la tabla 3.29.

### 3.17.5. Conclusiones

En los resultados podemos ver que no nos merece la pena que los cazadores se dividan para que uno vaya al teletransporte y el otro hacia la presa, ya que vemos que los resultados son peores en los dos niveles.

# 4. CAPÍTULO

# Conclusiones

Después de haber hecho todas las pruebas con todos los mapas, modos y versiones vamos a analizar cada modo para determinar cual es la mejor versión que hemos creado y analizaremos cual de las dos inteligencias es mejor, la del *Pac-Man* o la de los fantasmas. Para generar estos gráficos nos basaremos en los resultados conseguidos en cada versión, usando esos resultados para calcular nuevas medianas.

## 4.1. Primer modo de juego

Analizaremos cada nivel por separado y luego decidiremos cual de los dos tiene la mejor inteligencia en este modo.

### 4.1.1. Nivel 1

Podemos ver los resultados en la figura 4.1.

Podemos ver que conseguimos una mejoría clara en cada versión hasta llegar a la quinta. En este momento se intentan añadir nuevas mecánicas a los fantasmas haciendo que una inteligencia más compleja no consiga mejores resultados. Pasadas unas versiones conseguimos el mismo resultado con unos fantasmas más complejos, que ahora tienen la capacidad de traspasar algunas paredes y que cuentan con memoria a corto plazo.

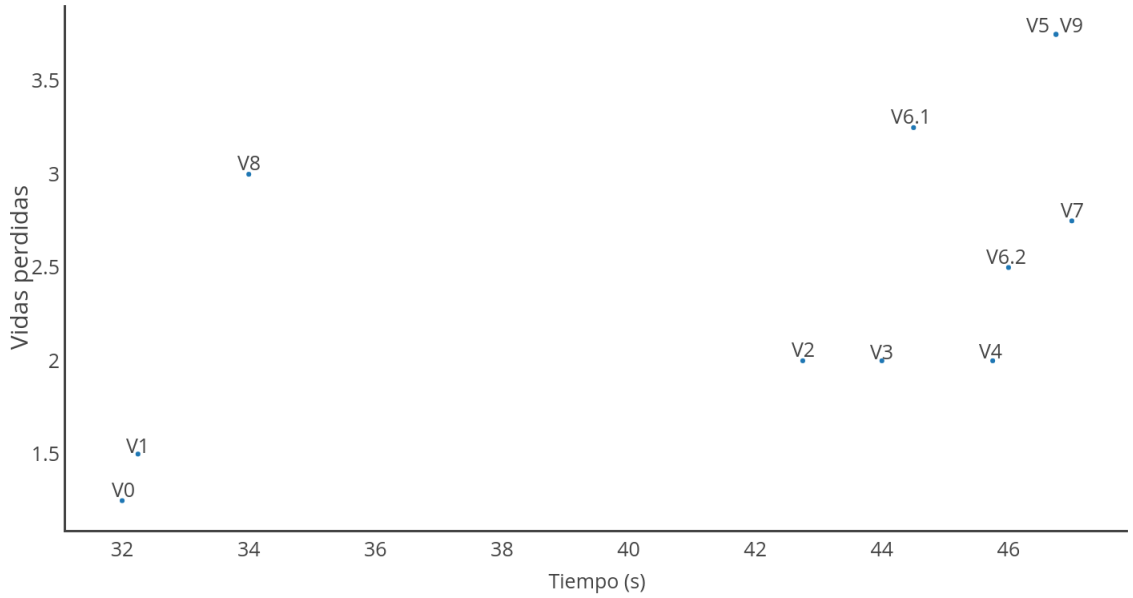



**Figura 4.1:** Gráfico con las medianas de las medianas del primer modo de juego en el primer nivel.

Con estos resultados podemos ver que no siempre es mejor una inteligencia más compleja, ya que puede que una inteligencia simple ya cumpla todo lo necesario, haciendo que complicar la inteligencia sea totalmente inútil.

### 4.1.2. Nivel 2

Podemos ver los resultados en la figura 4.2.

En este segundo nivel los resultados son más caóticos, ya que la estructura del mapa es completamente diferente a la del primer nivel, haciendo que inteligencias que resultaban muy útiles en allí resulten inútiles esta vez. En este nivel podemos ver que hay una inteligencia por encima de todas las demás, la última, aunque la mejora para llegar a esta no ha sido constante.

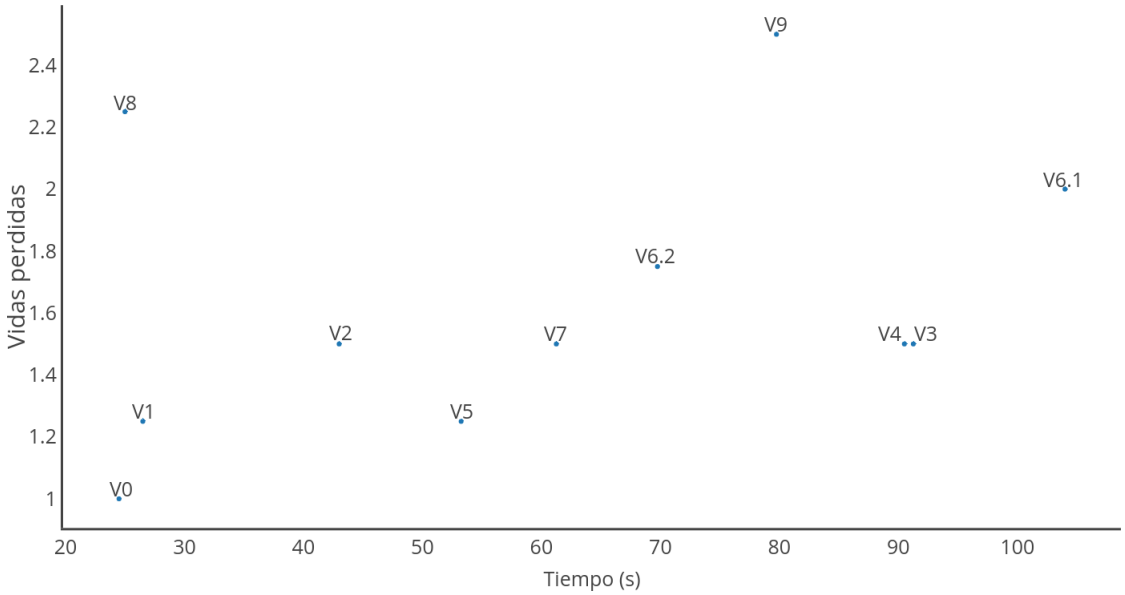



**Figura 4.2:** Gráfico con las medianas de las medianas del primer modo de juego en el segundo nivel.

### 4.1.3. Mejor inteligencia del primer modo de juego

Teniendo en cuenta que en el *Pac-Man* original el jugador tiene 3 vidas, por lo que diremos que los fantasmas han ganado si superan este número, y ganará el *Pac-Man* si no lo consiguen. Si miramos al primer mapa vemos que hay cuatro inteligencias que ganan: V5, V6.1, V8 y V9. Por otra parte, en el segundo mapa ninguna versión consigue atrapar a la presa más de 3 veces, pero la que mejores resultados consigue es la V9. Si hacemos la media de los resultados de la versión 9 en los dos mapas ésta atrapa a la presa 3 veces, por lo que los fantasmas son los que tienen la victoria en este modo.

## 4.2. Segundo modo de juego

Analizaremos los resultados de igual manera que con el primer modo de juego, analizando cada nivel por separado y luego decidiendo cual de los dos tiene la mejor inteligencia en este modo.

### 4.2.1. Nivel 1

Podemos ver los resultados en la figura 4.3.

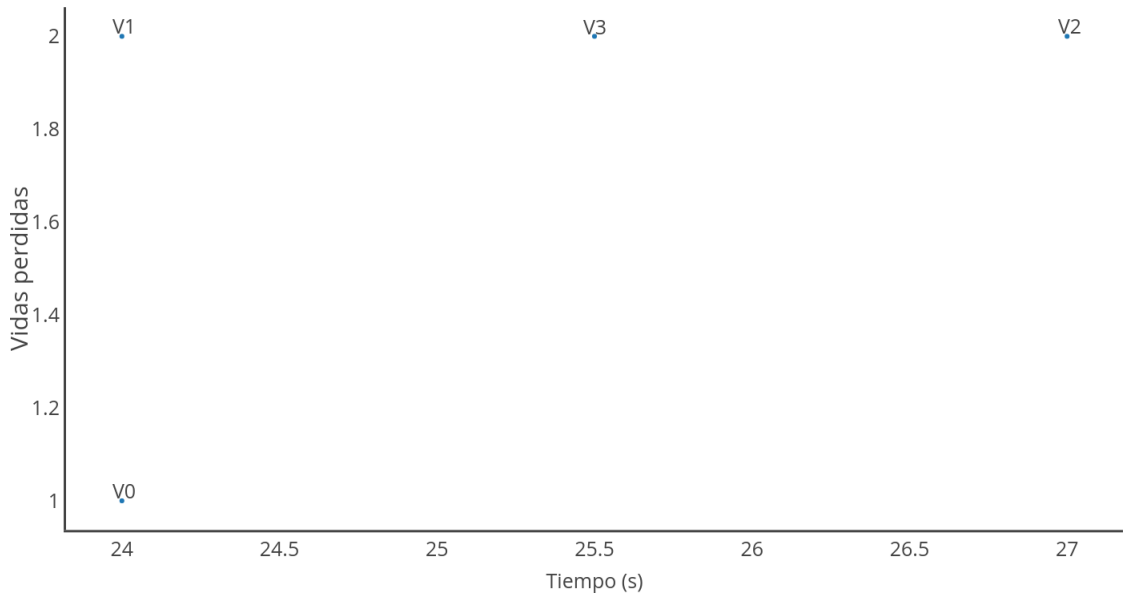



**Figura 4.3:** Gráfico con las medianas de las medianas del segundo modo de juego en el primer nivel.

Esta vez podemos ver que la mayoría de modos consiguen atrapar a la presa el mismo número de veces, por lo que miraremos el tiempo que ha necesitado la presa para terminar el nivel para ver cual es la mejor versión. Si miramos el tiempo veremos que la versión con la que más tiempo necesitan las presas para terminar el nivel es la segunda versión de los fantasmas, aunque la tercera sea más compleja. Como hemos mencionado al analizar el otro modo de juego, que una versión sea mas compleja no implica que sea mejor.

### 4.2.2. Nivel 2

Podemos ver los resultados en la figura 4.4.

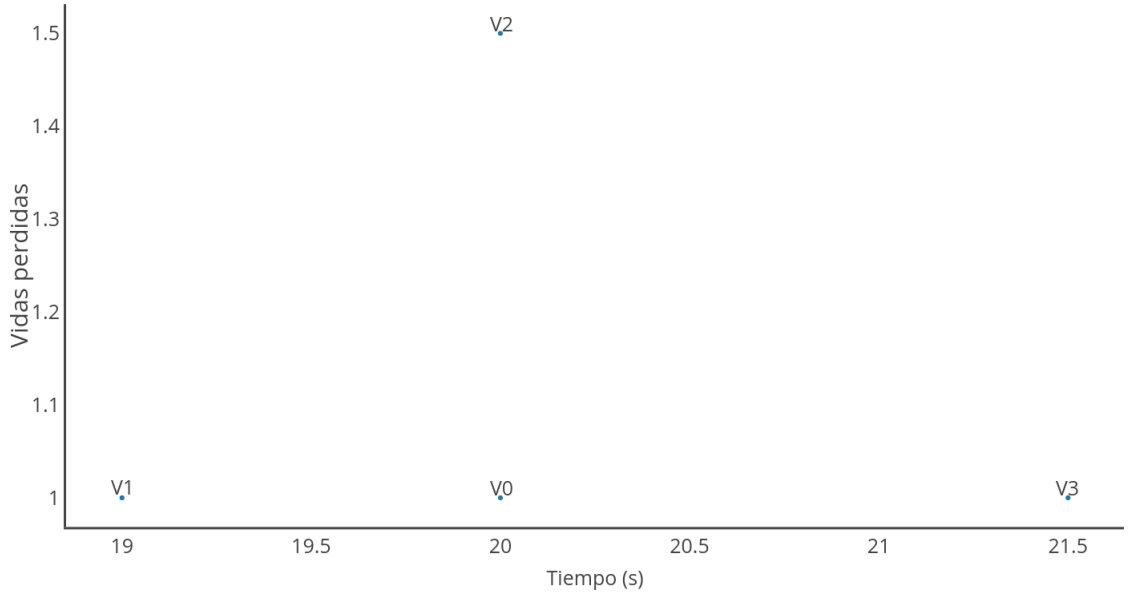



**Figura 4.4:** Gráfico con las medianas de las medianas del segundo modo de juego en el segundo nivel.

Esta vez no hay discusión posible, ya que todas atrapan a la presa una vez excepto la segunda versión, que la atrapa más veces. Aunque con la tercera versión la presa tarde más tiempo en terminar el nivel le daremos prioridad al número de veces que capturan a la presa, por lo que la mejor inteligencia para este mapa es la segunda.

### 4.2.3. Mejor inteligencia del segundo modo de juego

Analizándolo como el primer nivel, vemos que ninguna inteligencia consigue atrapar a las presas tres veces o más en ninguno de los dos niveles, por lo que damos por clara ganadora a la inteligencia de los *Pac-Man*s. Esto seguramente se debe a que a los fantasmas no les da tiempo a atrapar a los *Pac-Man*s, ya que los mapas son exactamente iguales en este modo de juego que en el anterior, cuando en este hay un *Pac-Man* más. Esto hace que el nivel termine mucho antes, dejando a los fantasmas menos tiempo para atraparlos.

## 4.3. Conclusiones generales

Desde el principio se ha pensado en este proyecto como una competición, por lo que debería haber un ganador y un perdedor. Como tenemos dos modos de juego y cada uno ha ganado en uno de los dos tenemos un empate. Cada uno ha intentado mejorar la versión donde tenía mas ventaja, como se puede ver en el número de versiones desarrolladas para cada modo. Aún así, en ninguno de los dos modos ha habido una victoria muy clara respecto al otro, por lo que ha estado bastante reñido hasta el final. Esta competición ha servido para motivar a los dos participantes ha conseguir la mejor inteligencia posible, por lo que se cree que se han conseguido mejores resultados que de otra forma con el mismo tiempo.

# Anexos

# A. ANEXO

## Código referenciado en el documento

**Listing A.1:** Código para generar el olor.

```
def Odor(self):
    if odorlist[0] != posicion_actual:
        if odorlist[0] != vacio and (odorlist[0][0] != fila_actual or odorlist[0][1] !=
     columna_actual):
            for i in range(8,-1,-1):
                if odorlist[i] != vacio:
                    odorlist[i+1] = odorlist[i]
        odorlist[0] = posicion_actual
```

**Listing A.2:** Código para generar la visión.

```
def Visible(self, thisLevel):
    visible = list()
    count = 0
    vertex = posicion_actual
    visible.append((vertex[0],vertex[1]))
    while not count==4:
        if count == 0:
            mientras vertex no sea pared:
                if not vertex == posicion_actual:
                    visible.append((vertex[0],vertex[1]))
                vertex = [vertex[0]+1,vertex[1]]
            else:
                if not visible[0]==vertex:
                    vertex = posicion_actual
                count +=1
        if count == 1:
            mientras vertex no sea pared:
```

```
                if not vertex == posicion_actual:
                    visible.append((vertex[0],vertex[1]))
                vertex = [vertex[0]-1,vertex[1]]
            else:
                if not visible[0]==vertex:
                    vertex = posicion_actual
                count +=1
        if count == 2:
            mientras vertex no sea pared:
                if not vertex == posicion_actual:
                    visible.append((vertex[0],vertex[1]))
                vertex = [vertex[0],vertex[1]+1]
            else:
                if not visible[0]==vertex:
                    vertex = posicion_actual
                count +=1
        if count == 3:
            mientras vertex no sea pared:
                if not vertex == posicion_actual:
                    visible.append((vertex[0],vertex[1]))
                vertex = [vertex[0],vertex[1]-1]
            else:
                if not visible[0]==vertex:
                    vertex = posicion_actual
                count +=1
```

**Listing A.3:** Código para usar la visión y el olor para seguir a la presa.

```
if casilla_nueva or (velX == 0 and velY == 0):
    v = thisLevel.GetTileVisible(self.nearestRow, self.nearestCol, player)
    s = thisLevel.GetTileOdor(self.nearestRow, self.nearestCol, player)
    if v!= 0:
        if v[0]< self.nearestRow:
            directions.append('U')
            count += 1
        elif v[0]> self.nearestRow:
            directions.append('D')
            count += 1
        elif v[1]< self.nearestCol:
            directions.append('L')
            count += 1
        else:
            directions.append('R')
            count += 1
    else:
        if s == 0:
            if not pared_arriba and orientation != 'D':
                directions.append('U')
                count += 1
            if not pared_abajo and orientation != 'U':
                directions.append('D')
```

```
                count += 1
            if not pared_derecha and orientation != 'L':
                directions.append('R')
                count += 1
            if not pared_izquierda and orientation != 'R':
                directions.append('L')
                count += 1
        else:
            directions.append(s)
            count += 1
    if count != 0:
        rand = random.randint(0,count-1)
        d = directions[rand]
```

**Listing A.4:** Código para gestionar la comunicación entre cazadores.

```

if ghostmove[1] != (-1,-1) and ghostmove[0] == (-1,-1):
    comunicado = 1
    if ghostmove[1][0] < self.nearestRow:
        if not pared_arriba and self.orientation != 'D':
            directions.append('U')
            count += 1
        elif ghostmove[1][1] < self.nearestCol:
            if not pared_izquierda and self.orientation != 'R':
                directions.append('L')
                count += 1
            elif not pared_derecha and self.orientation != 'U':
                directions.append('D')
                count += 1
            else:
                directions.append('R')
                count += 1
        elif ghostmove[1][1] > self.nearestCol:
            if not pared_derecha and self.orientation != 'L':
                directions.append('R')
                count += 1
            elif not pared_abajo and self.orientation != 'U':
                directions.append('D')
                count += 1
            else:
                directions.append('L')
                count += 1
        elif not pared_izquierda and self.orientation != 'R':
            directions.append('L')
            count += 1
        elif not pared_derecha and self.orientation != 'L':
            directions.append('R')
            count += 1
        else:
            directions.append('D')
```

```
            count += 1
    if ghostmove[1][0] > self.nearestRow:
        if not pared_abajo and self.orientation != 'U':
            directions.append('D')
            count += 1
        elif ghostmove[1][1] < self.nearestCol:
            if not pared_izquierda and self.orientation != 'R':
                directions.append('L')
                count += 1
            elif not pared_arriba and self.orientation != 'D':
                directions.append('U')
                count += 1
            else:
                directions.append('R')
                count += 1
        elif ghostmove[1][1] > self.nearestCol:
            if not pared_derecha and self.orientation != 'L':
                directions.append('R')
                count += 1
            elif not pared_arriba and self.orientation != 'D':
                directions.append('U')
                count += 1
            else:
                directions.append('L')
                count += 1
        elif not pared_izquierda and self.orientation != 'R':
            directions.append('L')
            count += 1
        elif not pared_derecha and self.orientation != 'L':
            directions.append('R')
            count += 1
        else:
            directions.append('U')
            count += 1
    if ghostmove[1][1] < self.nearestCol:
        if not pared_izquierda and self.orientation != 'R':
            directions.append('L')
            count += 1
        elif ghostmove[1][0] < self.nearestCol:
            if not pared_arriba and self.orientation != 'D':
                directions.append('U')
                count += 1
            elif not pared_derecha and self.orientation != 'L':
                directions.append('R')
                count += 1
            else:
                directions.append('D')
                count += 1
        elif ghostmove[1][0] > self.nearestCol:
            if not pared_abajo and self.orientation != 'U':
                directions.append('D')
                count += 1
```

```
            elif not pared_derecha and self.orientation != 'L':
                directions.append('R')
                count += 1
            else:
                directions.append('U')
                count += 1
        elif not pared_arriba and self.orientation != 'D':
            directions.append('U')
            count += 1
        elif not pared_abajo and self.orientation != 'U':
            directions.append('D')
            count += 1
        else:
            directions.append('R')
            count += 1
    else:
        if not pared_derecha and self.orientation != 'L':
            directions.append('R')
            count += 1
        elif ghostmove[1][0] < self.nearestCol:
            if not pared_arriba and self.orientation != 'D':
                directions.append('U')
                count += 1
            elif not pared_izquierda and self.orientation != 'R':
                directions.append('L')
                count += 1
            else:
                directions.append('D')
                count += 1
        elif ghostmove[1][0] > self.nearestCol:
            if not pared_abajo and self.orientation != 'U':
                directions.append('D')
                count += 1
            elif not pared_izquierda and self.orientation != 'R':
                directions.append('L')
                count += 1
            else:
                directions.append('U')
                count += 1
        elif not pared_arriba and self.orientation != 'D':
            directions.append('U')
            count += 1
        elif not pared_abajo and self.orientation != 'U':
            directions.append('D')
            count += 1
        else:
            directions.append('L')
            count += 1
elif ghostmove[0] != (-1,-1) and ghostmove[1] != (-1,-1):
    if self.orientation == 'U':
        if not pared_izquierda:
            directions.append('L')
```

```
            count += 1
        elif not pared_derecha:
            directions.append('R')
            count += 1
        else:
            directions.append('U')
            count += 1
    elif self.orientation == 'D':
        if not pared_izquierda:
            directions.append('L')
            count += 1
        elif not pared_derecha:
            directions.append('R')
            count += 1
        else:
            directions.append('D')
            count += 1
    elif self.orientation == 'L':
        if not pared_arriba:
            directions.append('U')
            count += 1
        elif not pared_abajo:
            directions.append('D')
            count += 1
        else:
            directions.append('L')
            count += 1
    elif self.orientation == 'R':
        if not pared_arriba:
            directions.append('U')
            count += 1
        elif not pared_abajo:
            directions.append('D')
            count += 1
        else:
            directions.append('R')
        count += 1
```

**Listing A.5:** Código para implementar la función *FindFood*.

```
def FindFood(self, thisLevel):
    visited, queue = set(), [(self.nearestRow, self.nearestCol, 0, [])]
    foodRow = list()
    foodCol = list()
    mientras que queue no este vacío:
        vertex = queue.pop(0)
        if vertex[2] < 10:
            if (vertex[0],vertex[1]) no está en visitados:
                visited.add((vertex[0],vertex[1]))
                if vertex no es pared:
                    if hay comida en la casilla y no es la posición actual del fantasma:
```

```
                        foodRow.append(vertex[0])
                        foodCol.append(vertex[1])
                    queue.append((vertex[0]+1,vertex[1],vertex[2]+1))
                    queue.append((vertex[0]-1,vertex[1],vertex[2]+1))
                    queue.append((vertex[0],vertex[1]+1,vertex[2]+1))
                    queue.append((vertex[0],vertex[1]-1,vertex[2]+1))
    if len(foodRow) != 0:
        food = (sum(foodRow)/len(foodRow),sum(foodCol)/len(foodCol))
        if food es pared:
            #miramos en todas las casillas alrededor hasta encontrar una que no sea pared
            if not thisLevel.IsWall((food[0]-1,food[1])):
                return((food[0]-1,food[1]))
            elif not thisLevel.IsWall((food[0]+1,food[1])):
                return((food[0]-1,food[1]))
            elif not thisLevel.IsWall((food[0],food[1]+1)):
                return((food[0],food[1]+1))
            elif not thisLevel.IsWall((food[0],food[1]-1)):
                return((food[0],food[1]-1))
            elif not thisLevel.IsWall((food[0]-1,food[1]-1)):
                return((food[0]-1,food[1]-1))
            elif not thisLevel.IsWall((food[0]-1,food[1]+1)):
                return((food[0]-1,food[1]+1))
            elif not thisLevel.IsWall((food[0]+1,food[1]-1)):
                return((food[0]+1,food[1]-1))
            elif not thisLevel.IsWall((food[0]+1,food[1]+1)):
                return((food[0]+1,food[1]+1))
        else:
            return(food)
    else:
        return None
```

**Listing A.6:** Código para que los cazadores sigan la comida.

```
if s == 0:
    D = self.FindFood(thisLevel)
    if D != None and i ==0:
        if D == (self.nearestRow,self.nearestCol):
            if self.orientation == 'U':
                if not thisLevel.IsWall((self.nearestRow-1,self.nearestCol)):
                    directions.append('U')
                    count += 1
                elif not thisLevel.IsWall((self.nearestRow,self.nearestCol-1)):
                    directions.append('L')
                    count += 1
                elif not thisLevel.IsWall((self.nearestRow,self.nearestCol+1)):
                    directions.append('R')
                    count += 1
            elif self.orientation == 'D':
                if not thisLevel.IsWall((self.nearestRow+1,self.nearestCol)):
                    directions.append('D')
                    count += 1
```

```
                elif not thisLevel.IsWall((self.nearestRow,self.nearestCol-1)):
                    directions.append('L')
                    count += 1
                elif not thisLevel.IsWall((self.nearestRow,self.nearestCol+1)):
                    directions.append('R')
                    count += 1
            elif self.orientation == 'L':
                if not thisLevel.IsWall((self.nearestRow,self.nearestCol-1)):
                    directions.append('L')
                    count += 1
                elif not thisLevel.IsWall((self.nearestRow+1,self.nearestCol)):
                    directions.append('D')
                    count += 1
                elif not thisLevel.IsWall((self.nearestRow-1,self.nearestCol)):
                    directions.append('U')
                    count += 1
            elif self.orientation == 'R':
                if not thisLevel.IsWall((self.nearestRow,self.nearestCol+1)):
                    directions.append('R')
                    count += 1
                elif not thisLevel.IsWall((self.nearestRow+1,self.nearestCol)):
                    directions.append('D')
                    count += 1
                elif not thisLevel.IsWall((self.nearestRow-1,self.nearestCol)):
                    directions.append('U')
                    count += 1
        elif D[0] < self.nearestRow:
            ...
```

**Listing A.7:** Código para que los cazadores se queden en su parte del mapa.

```
if i == 0 and self.nearestRow > 9:
    if not pared_arriba and self.orientation != 'D':
        directions.append('U')
        count += 1
    elif not pared_izquierda and self.orientation != 'R':
        directions.append('L')
        count += 1
    elif not pared_derecha and self.orientation != 'L':
        directions.append('R')
        count += 1
    elif not pared_abajo and self.orientation != 'U':
                directions.append('D')
                count += 1
elif i == 1 and self.nearestRow < 15:
    if not pared_abajo and self.orientation != 'U':
                directions.append('D')
                count += 1
    elif not pared_izquierda and self.orientation != 'R':
        directions.append('L')
        count += 1
```

```
    elif not pared_derecha and self.orientation != 'L':
        directions.append('R')
        count += 1
    elif not pared_arriba and self.orientation != 'D':
        directions.append('U')
        count += 1
```

**Listing A.8:** Código para gestionar la visión y el olor cuando los cazadores están en modo vulnerable.

```

#Gestionar la visión

ghostmove[i] = v
if self.state ==1:
    if v[0] encima de la posición actual:
        directions.append('U')
        count += 1
    elif v[0] debajo de la posición actual:
        directions.append('D')
        count += 1
    elif v[1] a la izquierda de la posición actual:
        directions.append('L')
        count += 1
    else:
        directions.append('R')
        count += 1
elif self.state == 2:
    if v[0] encima de la posición actual:
        if not pared_abajo:
            directions.append('D')
            count += 1
        if not pared_izquierda:
            directions.append('L')
            count += 1
        if not pared_derecha:
            directions.append('R')
            count += 1
    elif v[0] debajo de la posición actual:
        if not pared_arriba:
            directions.append('U')
            count += 1
        if not pared_izquierda:
            directions.append('L')
            count += 1
        if not pared_derecha:
            directions.append('R')
            count += 1
    elif v[1] a la izquierda de la posición actual:
        if not pared_derecha:
```

```
            directions.append('R')
            count += 1
        if not pared_arriba:
            directions.append('U')
            count += 1
        if not pared_abajo:
            directions.append('D')
            count += 1
    else:
        if not pared_izquierda:
            directions.append('L')
            count += 1
        if not pared_arriba:
            directions.append('U')
            count += 1
        if not pared_abajo:
            directions.append('D')
            count += 1

...

#Para gestionar el olor

if self.state == 1:
    directions.append(s)
    count += 1
elif self.state == 2:
    if s == 'U':
        if not pared_abajo:
            directions.append('D')
            count += 1
        if not pared_izquierda:
            directions.append('L')
            count += 1
        if not pared_derecha:
            directions.append('R')
            count += 1
    elif s == 'D':
        if not pared_arriba:
            directions.append('U')
            count += 1
        if not pared_izquierda:
            directions.append('L')
            count += 1
        if not pared_derecha:
            directions.append('R')
            count += 1
    elif s == 'L':
        if not pared_derecha:
            directions.append('R')
            count += 1
        if not pared_arriba:
```

```
            directions.append('U')
            count += 1
        if not pared_abajo:
            directions.append('D')
            count += 1
    else:
        if not pared_izquierda:
            directions.append('L')
            count += 1
        if not pared_arriba:
            directions.append('U')
            count += 1
        if not pared_abajo:
            directions.append('D')
            count += 1
if (s == 'U'):
    ghostmove[i] = (self.nearestRow-4,self.nearestCol)
elif (s == 'D'):
    ghostmove[i] = (self.nearestRow+4,self.nearestCol)
elif (s == 'R'):
    ghostmove[i] = (self.nearestRow,self.nearestCol+4)
elif (s == 'L'):
    ghostmove[i] = (self.nearestRow,self.nearestCol-4)
```

**Listing A.9:** Código para gestionar las emboscadas en los teletransportes.

```
#vision
if self.nearestRow < 8:
    ghostmove[i] = (22,10)
elif self.nearestRow > 16:
    ghostmove[i] = (2,10)
elif self.nearestCol < 6:
    ghostmove[i] = (12,17)
elif self.nearestCol > 13:
    ghostmove[i] = (12,3)
else:
    ghostmove[i] = v

...
#olor
if self.nearestRow < 8:
    ghostmove[i] = (22,10)
elif self.nearestRow > 16:
    ghostmove[i] = (2,10)
elif self.nearestCol < 7:
    ghostmove[i] = (12,17)
elif self.nearestCol > 12:
    ghostmove[i] = (12,3)
else:
    if (s == 'U'):
        ghostmove[i] = (self.nearestRow-4,self.nearestCol)
```

```
    elif (s == 'D'):
        ghostmove[i] = (self.nearestRow+4,self.nearestCol)
    elif (s == 'R'):
        ghostmove[i] = (self.nearestRow,self.nearestCol+4)
    elif (s == 'L'):
        ghostmove[i] = (self.nearestRow,self.nearestCol-4)
```

**Listing A.10:** Código para gestionar el uso de las *ghost-pellets*.

```

#Level
def CheckIfGhostHitSomething (self, (ghostX, ghostY), (row, col), ghost, i, version):
    for iRow in range(row - 1, row + 2, 1):
        for iCol in range(col - 1, col + 2, 1):
            if  (ghostX - (iCol * TILE_WIDTH) < TILE_WIDTH) and (ghostX - (iCol * TILE_WIDTH) > -
     TILE_WIDTH) and (ghostY - (iRow * TILE_HEIGHT) < TILE_HEIGHT) and (ghostY - (iRow *
     TILE_HEIGHT) > -TILE_HEIGHT):
                result = self.GetMapTile((iRow, iCol))
                if result == tileID[ 'pellet-ghost' ]:
                    self.SetMapTile((iRow, iCol), 0)
                    if version >= 6:
                        ghost.pellet += 3

#Ghost
if ghostmove[0][0] < self.nearestRow:
    if (not pared_arriba or self.pellet > 0) and self.orientation != 'D':
        if pared_arriba:
            self.pellet -= 1
        directions.append('U')
        count += 1
    elif ghostmove[0][1] < self.nearestCol:
        if (not pared_izquierda or self.pellet > 0) and self.orientation != 'R':
            if pared_izquierda:
                self.pellet -= 1
            directions.append('L')
            count += 1
        elif (not pared_abajo or self.pellet > 0) and self.orientation != 'U':
            if pared_abajo:
                self.pellet -= 1
            directions.append('D')
            count += 1
        else:
            directions.append('R')
            count += 1
    elif ghostmove[0][1] > self.nearestCol:
        if (not pared_derecha or self.pellet > 0) and self.orientation != 'L':
            if pared_derecha:
                self.pellet -= 1
            directions.append('R')
            count += 1
        elif (not pared_abajo or self.pellet > 0) and self.orientation != 'U':
```

```
            if pared_abajo:
                self.pellet -=1
            directions.append('D')
            count += 1
        else:
            directions.append('L')
            count += 1
    elif (not pared_izquierda or self.pellet > 0) and self.orientation != 'R':
        if pared_izquierda:
            self.pellet -=1
        directions.append('L')
        count += 1
    elif (not pared_derecha or self.pellet > 0) and self.orientation != 'L':
        if pared_derecha:
            self.pellet -=1
        directions.append('R')
        count += 1
    else:
        directions.append('D')
        count += 1
```

**Listing A.11:** Código para gestionar el camino que debe tomar el cazador en la versión 6.

```
if (abs(door[1][0] - self.nearestRow)+abs(door[1][1] - self.nearestCol) < abs(ghostmove[1][0] -
     self.nearestRow)+abs(ghostmove[1][1] - self.nearestCol)):
    ghostmove[1] = door[1]

...

if self.nearestRow < 8:
    door[i] = (22,10)
elif self.nearestRow > 16:
    door[i] = (2,10)
elif self.nearestCol < 6:
    door[i] = (12,17)
elif self.nearestCol > 13:
    door[i] = (12,3)
ghostmove[i] = v

...

if self.nearestRow < 8:
    door[i] = (22,10)
elif self.nearestRow > 16:
    door[i] = (2,10)
elif self.nearestCol < 7:
    door[i] = (12,17)
elif self.nearestCol > 12:
    door[i] = (12,3)
if (s == 'U'):
    ghostmove[i] = (self.nearestRow-4,self.nearestCol)
```

```
elif (s == 'D'):
    ghostmove[i] = (self.nearestRow+4,self.nearestCol)
elif (s == 'R'):
    ghostmove[i] = (self.nearestRow,self.nearestCol+4)
elif (s == 'L'):
    ghostmove[i] = (self.nearestRow,self.nearestCol-4)
```

**Listing A.12:** Código para geestionar el ruido y la velocidad del cazador.

```

#ghost
...
if (self.nearestRow, self.nearestCol) in thisLevel.doors:
    self.speed = 1
...
if ghostmove[1] != (-1,-1):
    self.speed = 2
...
if ghostmove[0] != (-1,-1):
    self.speed = 2
...
if v!= 0:
    self.speed = 2
...
if s == 0:
    self.speed = 1
    ...
else:
    self.speed = 2
...
return(self.speed)

#main
for i in range(0, 2, 1):
    speed = ghosts[i].Move(thisLevel, player, ghostmove, door, i)
    # actualizar la lista con las casillas de sonido
    if speed == 2:
        sonido = 10
    else:
        sonido = 4
    ghosts[i].UpdateSound(sonido, thisLevel)
```

**Listing A.13:** Código para generar la memoria a corto plazo.

```

#main

ghostmove = {}
door = {}
```

```
visto = {}

for i in range(0, 2, 1):
    ghostmove[i] = (-1,-1)
    door[i] = (-1,-1)
    visto[i] = (-1,-1)

#ghost

...
if (self.nearestRow, self.nearestCol) in thisLevel.doors:
    visto[i]= (-1,-1)
...
if ghostmove[1] != (-1,-1):
    visto[i]= (-1,-1)
...
if ghostmove[0] != (-1,-1):
    visto[i]= (-1,-1)
...
if v!= 0:
    if self.nearestRow < 8:
        door[i] = (23,10)
    elif self.nearestRow > 16:
        door[i] = (1,10)
    elif self.nearestCol < 6:
        door[i] = (12,19)
    elif self.nearestCol > 13:
        door[i] = (12,1)
    ghostmove[i] = v
    visto[i] = v
...
if s == 0:
    if visto[i] != (-1,-1):
        if visto[i] == (self.nearestRow, self.nearestCol):
            visto[i] = (-1,-1)
        elif visto[i] in thisLevel.doors:
            visto[i] = (-1,-1)
        elif visto[i][0] < self.nearestRow:
            if (not thisLevel.IsWall((self.nearestRow-1,self.nearestCol)) or (self.pellet > 0 and
     fuera == 0)) and self.orientation != 'D':
                if thisLevel.IsWall((self.nearestRow-1,self.nearestCol)):
                    self.pellet -= 1
                directions.append('U')
                count += 1
        #se seguirá la posición de visto como se ha hecho en las anteriores versiones con
     ghostmove
        ...
else:
    visto[i] = (-1,-1)
...
```

**Listing A.14:** Código para hacer compatible la versión 1.9 con el segundo modo de juego.

```

#Gestionar la visión y el olor
...
for k in range(0, 2, 1):
    v[k] = thisLevel.GetTileVisible(self.nearestRow, self.nearestCol, players[k])
    s[k] = thisLevel.GetTileOdor(self.nearestRow, self.nearestCol, players[k])
...
elif i==0:
    if ghostmove[1] != (-1,-1) or ghostmove[2] != (-1,-1):
        ghostmove[0] = (-1,-1)
        if ghostmove[1] != (-1,-1):
            g=1
            ghostmove[2] = (-1,-1)
        else:
            g=2
            ghostmove[1] = (-1,-1)
        visto[i]= (-1,-1)
        if s[0]!=0 or s[1]!=0:
            comunicado =2
        else:
            comunicado = 1
        if comunicado == 1:
            if (abs(door[g][0] - self.nearestRow)+abs(door[g][1] - self.nearestCol) < abs(
    ghostmove[g][0] - self.nearestRow)+abs(ghostmove[g][1] - self.nearestCol)):
                ghostmove[g] = door[g]
            if ghostmove[g][0] < self.nearestRow:
                if (not thisLevel.IsWall((self.nearestRow-1,self.nearestCol)) or (self.pellet > 0
     and fuera == 0)) and self.orientation != 'D':
                    if thisLevel.IsWall((self.nearestRow-1,self.nearestCol)):
                        self.pellet -= 1
                    directions.append('U')
                    count += 1
                elif ghostmove[g][1] < self.nearestCol:
    ...
elif i==1:
    if ghostmove[0] != (-1,-1) or ghostmove[2] != (-1,-1):
        ghostmove[1] = (-1,-1)
        if ghostmove[0]!= (-1,-1):
            g = 0
            ghostmove[2] = (-1,-1)
        else:
            g = 2
            ghostmove[0] = (-1,-1)
        visto[i]= (-1,-1)
        if s[0]!=0 or s[1]!=0:
            comunicado =2
        else:
            comunicado = 1
        if comunicado==1:
            if (abs(door[g][0] - self.nearestRow)+abs(door[g][1] - self.nearestCol) < abs(
```

```
     ghostmove[g][0] - self.nearestRow)+abs(ghostmove[g][1] - self.nearestCol)):
                ghostmove[g] = door[g]
            if ghostmove[g][0] < self.nearestRow:
                if (not thisLevel.IsWall((self.nearestRow-1,self.nearestCol)) or (self.pellet > 0
     and fuera == 0)) and self.orientation != 'D':
                    if thisLevel.IsWall((self.nearestRow-1,self.nearestCol)):
                        self.pellet -= 1
                    directions.append('U')
                    count += 1
                elif ghostmove[g][1] < self.nearestCol:
    ...
elif i==2:
    if ghostmove[0] != (-1,-1) or ghostmove[1]!= (-1,-1):
        ghostmove[2] = (-1,-1)
        if ghostmove[0] != (-1,-1):
            g=0
            ghostmove[1] = (-1,-1)
        else:
            g=1
            ghostmove[0] = (-1,-1)
        visto[i]= (-1,-1)
        if s[0]!=0 or s[1]!=0:
            comunicado =2
        else:
            comunicado = 1
        if comunicado == 1:
            if (abs(door[g][0] - self.nearestRow)+abs(door[g][1] - self.nearestCol) < abs(
    ghostmove[g][0] - self.nearestRow)+abs(ghostmove[g][1] - self.nearestCol)):
                ghostmove[g] = door[g]
            if ghostmove[g][0] < self.nearestRow:
                if (not thisLevel.IsWall((self.nearestRow-1,self.nearestCol)) or (self.pellet > 0
     and fuera == 0)) and self.orientation != 'D':
                    if thisLevel.IsWall((self.nearestRow-1,self.nearestCol)):
                        self.pellet -= 1
                    directions.append('U')
                    count += 1
                elif ghostmove[g][1] < self.nearestCol:
    ...
if(comunicado==0):
    ghostmove[i]=(-1,-1)
    if v[0]!= 0 or v[1]!= 0:
        if self.nearestRow < 8:
            door[i] = (23,10)
        elif self.nearestRow > 16:
            door[i] = (1,10)
        elif self.nearestCol < 6:
            door[i] = (12,19)
        elif self.nearestCol > 13:
            door[i] = (12,1)
        if v[0]!= 0:
            j=0
        else:
```

```
            j=1
        ghostmove[i] = v[j]
        visto[i] = v[j]
    ...
#no se detecta olor
else:
    D = self.FindFood(thisLevel)
    if i == 0 and self.nearestRow > 9:
    ...
    elif i == 1 and self.nearestRow < 15:
    ...
    elif D != None and i!=2:
    ...
    else:
        if not thisLevel.IsWall((self.nearestRow-1,self.nearestCol)) and self.orientation != 'D':
            directions.append('U')
            count += 1
        if not thisLevel.IsWall((self.nearestRow+1,self.nearestCol)) and self.orientation != 'U':
            directions.append('D')
            count += 1
        if not thisLevel.IsWall((self.nearestRow,self.nearestCol+1)) and self.orientation != 'L':
            directions.append('R')
            count += 1
        if not thisLevel.IsWall((self.nearestRow,self.nearestCol-1)) and self.orientation != 'R':
            directions.append('L')
            count += 1
...
#se detecta olor
else:
    visto[i] = (-1,-1)
    if s[0]!=0:
        j=0
    else:
        j=1
    if self.state == 1:
        directions.append(s[j])
        count += 1
...
```

**Listing A.15:** Código para implementar el vigilante.

```

elif players[0].dead or players[1].dead and i == 2:
    if players[0].dead:
        Dead = (players[0].nearestRow, players[0].nearestCol)
    else:
        Dead = (players[1].nearestRow, players[1].nearestCol)
    if Dead == (self.nearestRow,self.nearestCol):
        if self.orientation == 'U':
            if not thisLevel.IsWall((self.nearestRow-1,self.nearestCol)):
                directions.append('U')
```

```
                count += 1
            elif not thisLevel.IsWall((self.nearestRow,self.nearestCol-1)):
                directions.append('L')
                count += 1
            elif not thisLevel.IsWall((self.nearestRow,self.nearestCol+1)):
                directions.append('R')
                count += 1
        elif self.orientation == 'D':
            if not thisLevel.IsWall((self.nearestRow+1,self.nearestCol)):
                directions.append('D')
                count += 1
            elif not thisLevel.IsWall((self.nearestRow,self.nearestCol-1)):
                directions.append('L')
                count += 1
            elif not thisLevel.IsWall((self.nearestRow,self.nearestCol+1)):
                directions.append('R')
                count += 1
        elif self.orientation == 'L':
            if not thisLevel.IsWall((self.nearestRow,self.nearestCol-1)):
                directions.append('L')
                count += 1
            elif not thisLevel.IsWall((self.nearestRow+1,self.nearestCol)):
                directions.append('D')
                count += 1
            elif not thisLevel.IsWall((self.nearestRow-1,self.nearestCol)):
                directions.append('U')
                count += 1
        elif self.orientation == 'R':
            if not thisLevel.IsWall((self.nearestRow,self.nearestCol+1)):
                directions.append('R')
                count += 1
            elif not thisLevel.IsWall((self.nearestRow+1,self.nearestCol)):
                directions.append('D')
                count += 1
            elif not thisLevel.IsWall((self.nearestRow-1,self.nearestCol)):
                directions.append('U')
                count += 1
    elif Dead[0] < self.nearestRow:
        ...
```

**Listing A.16:** Código para implementar la comunicación de objetivo.

```


#main
ghostmove = {}
door = {}
visto = {}
objetivo = {}

if gameMode == 0:
```

```
10      for i in range(0, 2, 1):
11          ghostmove[i] = (-1,-1)
12          door[i] = (-1,-1)
13          visto[i] = (-1,-1)
14  else:
15      for i in range(0, 3, 1):
16          ghostmove[i] = (-1,-1)
17          door[i] = (-1,-1)
18          visto[i] = (-1,-1)
19          if ghostVersion > 1:
20              objetivo[i] = -1
21  ...
22
23  for i in range(0, 3, 1):
24      if ghostVersion<2:
25          ghosts[i].Move(thisLevel, player, ghostmove, door, visto, i)
26      else:
27          ghosts[i].Move(thisLevel, player, ghostmove, door, visto, objetivo, i)
28      ghosts[i].UpdateSound(10, thisLevel)
29
30  #ghost
31  ...
32  elif i==0:
33      if ghostmove[1] != (-1,-1) or ghostmove[2] != (-1,-1):
34          ghostmove[0] = (-1,-1)
35          if ghostmove[1] != (-1,-1):
36              g=1
37              ghostmove[2] = (-1,-1)
38          else:
39              g=2
40              ghostmove[1] = (-1,-1)
41          objetivo[0] = objetivo[g]
42          visto[i]= (-1,-1)
43          if s[0]!=0 and objetivo[0] == 0:
44              comunicado =2
45          elif s[1]!=0 and objetivo[0] == 1:
46              comunicado =2
47  ...
48  elif i==1:
49      if ghostmove[0] != (-1,-1) or ghostmove[2] != (-1,-1):
50          ghostmove[1] = (-1,-1)
51          if ghostmove[0]!= (-1,-1):
52              g = 0
53              ghostmove[2] = (-1,-1)
54          else:
55              g = 2
56              ghostmove[0] = (-1,-1)
57          visto[i]= (-1,-1)
58          objetivo[1] = objetivo[g]
59          if s[0]!=0 and objetivo[1] == 0:
60              comunicado =2
61          elif s[1]!=0 and objetivo[1] == 1:
```

```
            comunicado =2
...
elif i==2:
    if ghostmove[0] != (-1,-1) or ghostmove[1]!= (-1,-1):
        ghostmove[2] = (-1,-1)
        if ghostmove[0] != (-1,-1):
            g=0
            ghostmove[1] = (-1,-1)
        else:
            g=1
            ghostmove[0] = (-1,-1)
        objetivo[2] = objetivo[g]
        visto[i]= (-1,-1)
        if s[0]!=0 and objetivo[2] == 0:
            comunicado =2
        elif s[1]!=0 and objetivo[2] == 1:
            comunicado =2
...
if(comunicado==0):
    ghostmove[i]=(-1,-1)
    if v[0]!= 0 or v[1]!= 0:
        if self.nearestRow < 8:
            door[i] = (23,10)
        elif self.nearestRow > 16:
            door[i] = (1,10)
        elif self.nearestCol < 6:
            door[i] = (12,19)
        elif self.nearestCol > 13:
            door[i] = (12,1)
        if v[0]!= 0:
            j=0
        else:
            j=1
        objetivo[i] = j
        ghostmove[i] = v[j]
        visto[i] = v[j]
        ...
    else:
        if s[0] == 0 and s[1]==0:
            objetivo[i] = -1
...
else:
    visto[i] = (-1,-1)
    if s[0]!=0:
        j=0
    else:
        j=1
    objetivo[i] = j
```

**Listing A.17:** Código para implementar la mejora del trabajo en equipo.

```
1
2   ...
3   posicion[i] = (self.nearestRow, self.nearestCol)
4
5   thisLevel.CheckIfGhostHitSomething((self.x, self.y), (self.nearestRow, self.nearestCol),self,i,
        self.version)
6
7   directions = list()
8   count = 0
9   comunicado = 0
10  fuera = 0
11
12  camino = (-1,-1)
13
14  ...
15
16  elif i==0:
17      if ghostmove[1] != (-1,-1) or ghostmove[2] != (-1,-1):
18          if ghostmove[1] != (-1,-1) or ghostmove[2] != (-1,-1):
19          ghostmove[0] = (-1,-1)
20          if ghostmove[1] != (-1,-1):
21              g=1
22              j=2
23              ghostmove[2] = (-1,-1)
24          else:
25              g=2
26              j=1
27              ghostmove[1] = (-1,-1)
28          objetivo[0] = objetivo[g]
29          visto[i]= (-1,-1)
30          if s[0]!=0 and objetivo[0] == 0:
31              comunicado =2
32          elif s[1]!=0 and objetivo[0] == 1:
33              comunicado =2
34          else:
35              comunicado = 1
36          if comunicado == 1:
37              if (abs(door[g][0] - self.nearestRow)+abs(door[g][1] - self.nearestCol) < abs(door[g
        ][0] - posicion[j][0])+abs(door[g][1] - posicion[j][1])):
38                  camino = door[g]
39              else:
40                  camino = ghostmove[g]
41              if camino[0] < self.nearestRow:
42                  if (not thisLevel.IsWall((self.nearestRow-1,self.nearestCol)) or (self.pellet > 0
         and fuera == 0)) and self.orientation != 'D':
43                      if thisLevel.IsWall((self.nearestRow-1,self.nearestCol)):
44                          self.pellet -= 1
45                      directions.append('U')
46                      count += 1
47                  elif camino[1] < self.nearestCol:
48      ...
```

```
elif i==1:
    if ghostmove[0] != (-1,-1) or ghostmove[2] != (-1,-1):
        ghostmove[1] = (-1,-1)
        if ghostmove[0]!= (-1,-1):
            g = 0
            j = 2
            ghostmove[2] = (-1,-1)
        else:
            g = 2
            j = 0
            ghostmove[0] = (-1,-1)
        visto[i]= (-1,-1)
        objetivo[1] = objetivo[g]
        if s[0]!=0 and objetivo[1] == 0:
            comunicado =2
        elif s[1]!=0 and objetivo[1] == 1:
            comunicado =2
        else:
            comunicado = 1
        if comunicado==1:
            if (abs(door[g][0] - self.nearestRow)+abs(door[g][1] - self.nearestCol) < abs(door[g
    ][0] - posicion[j][0])+abs(door[g][1] - posicion[j][1])):
                camino = door[g]
            else:
                camino = ghostmove[g]
            if camino[0] < self.nearestRow:
                if (not thisLevel.IsWall((self.nearestRow-1,self.nearestCol)) or (self.pellet > 0
     and fuera == 0)) and self.orientation != 'D':
                    if thisLevel.IsWall((self.nearestRow-1,self.nearestCol)):
                        self.pellet -= 1
                    directions.append('U')
                    count += 1
                elif camino[1] < self.nearestCol:
    ...
elif i==2:
    if ghostmove[0] != (-1,-1) or ghostmove[1]!= (-1,-1):
        ghostmove[2] = (-1,-1)
        if ghostmove[0] != (-1,-1):
            g=0
            j=1
            ghostmove[1] = (-1,-1)
        else:
            g=1
            j=0
            ghostmove[0] = (-1,-1)
        objetivo[2] = objetivo[g]
        visto[i]= (-1,-1)
        if s[0]!=0 and objetivo[2] == 0:
            comunicado =2
        elif s[1]!=0 and objetivo[2] == 1:
            comunicado =2
        else:
```

```
99              comunicado = 1
100         if comunicado == 1:
101             if (abs(door[g][0] - self.nearestRow)+abs(door[g][1] - self.nearestCol) < abs(door[g
        ][0] - posicion[j][0])+abs(door[g][1] - posicion[j][1])):
102                 camino = door[g]
103             else:
104                 camino = ghostmove[g]
105             if camino[0] < self.nearestRow:
106                 if (not thisLevel.IsWall((self.nearestRow-1,self.nearestCol)) or (self.pellet > 0
         and fuera == 0)) and self.orientation != 'D':
107                     if thisLevel.IsWall((self.nearestRow-1,self.nearestCol)):
108                         self.pellet -= 1
109                     directions.append('U')
110                     count += 1
111                 elif camino[1] < self.nearestCol:
112 ...
```

# Bibliografía

[Atari, ] Atari. Asteroids description. https://www.arcade-history.com/?n=asteroids&page=detail&id=126/.

[Birch, ] Birch, C. Understanding pac-man ghost behavior. http://gameinternals.com/post/2072558330/understanding-pac-man-ghost-behavior/.

[Goldberg, ] Goldberg, M. Pac-man: The phenomenon. https://web.archive.org/web/20131029194218/http://classicgaming.gamespy.com/View.php?view=Articles.Detail&id=249/.

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