PONTIFICIA UNIVERSIDAD CATÓLICA DE VALPARAÍSO
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA INFORMÁTICA

Modelado y resolución del Ms. PacMan Problem utilizando programación con restricciones

Diego Nicolás González López
Francisco Lorenzo Lobos Ulloa

INFORME FINAL DE PROYECTO
PARA OPTAR AL TÍTULO PROFESIONAL DE
INGENIERO EN EJECUCIÓN INFORMÁTICA

Enero, 2012

Pontificia Universidad Católica de Valparaíso
Facultad de Ingeniería
Escuela de Ingeniería Informática

Modelado y resolución del Ms. PacMan Problem utilizando programación con restricciones

Diego Nicolás González López
Francisco Lorenzo Lobos Ulloa

Profesor Guía: Ricardo Soto De Giorgis
Profesor Co-referente: Broderick Crawford Labrín

Carrera: Ingeniería de Ejecución en Informática

Enero, 2012

Dedicatoria

A nuestras familias por ser el pilar fundamental de apoyo y energía, enseñándonos con sus ejemplos para ser perseverantes y darnos la fuerza que nos impulsara a conseguir nuestros sueños y metas.

Agradecimientos

A Dios, a nuestros grupos familiares que fueron el pilar más importante en nuestra instancia en la universidad, que siempre tuvieron sus corazones llenos de esperanzas y fe en nosotros y nos brindaron todo lo necesario para sentirnos cómodos y estar bien en nuestros estudios y vida universitaria.

A los docentes que nos han acompañado durante el largo camino, brindándonos siempre su orientación con profesionalismo ético en la adquisición de conocimientos y afianzando nuestra formación.

Igualmente a nuestro profesor guía quien nos ha orientado y brindado apoyo en todo momento en la realización de este proyecto.

A nuestros amigos, compañeros y personas que nos acompañaron en esta odisea universitaria, por apoyarnos, ayudarnos en momentos de necesidad, por su alegría y ánimos, comprensión y compañía en estos 5 años.

Resumen

Ms. PacMan es un videojuego arcade producido originalmente por Namco, cuyo objetivo es lograr que Ms. PacMan capture todas las píldoras de un laberinto evitando ser atrapada por un conjunto de fantasmas. El propósito de este proyecto es simular el comportamiento de Ms. PacMan de tal manera que supere el laberinto dentro del juego. Esto se realiza por medio de la programación con restricciones *Constraint Programming* (CP) donde cada movimiento de Ms. PacMan se modela como un problema de satisfacción de restricciones *Constraint Satisfaction Problem* (CSP). El sistema se implementó utilizando el solver Choco, una versión de código abierto de Ms. PacMan y una interfaz que permite la comunicación entre ambos componentes.

Palabras Clave: Programación con restricciones, Satisfacción, Ms. PacMan, Modelo.

Abstract

Ms. PacMan is an arcade game originally produced by Namco, whose goal is to achieve that Ms. PacMan captures the whole set of pills from a labyrinth by avoiding to be attacked by ghosts. The purpose of this project is to simulate the behavior of Ms. PacMan in order to pass the maze. This is done by using constraint programming (CP) where each Ms. PacMan move is modeled as a constraint satisfaction problem (CSP). The system has been implemented by using the Choco Solver, an open source version of Ms. PacMan and an interface that allow the communication between both components.

Keywords: Constraint Programming, Satisfaction, Ms. PacMan, Model.

Índice

1. Introducción **1**

2. Definición de Objetivos **2**
2.1. Objetivos Generales 2
2.2. Objetivos Específicos 2

3. Estado del Arte **3**

4. Programación con restricciones **5**
4.1. Problema de satisfacción de restricciones 5
4.2. Ejemplos CSP 6
4.2.1. Send + More = Money 6
4.2.2. N-Reinas 6
4.3. Algoritmos de búsqueda 7
4.3.1. Generate and Test 7
4.3.2. Backtracking 8
4.3.3. Backjumping 9
4.3.4. Forward checking 10
4.3.5. Maintaining Arc Consistency 11
4.4. Estrategias de Enumeración 11
4.4.1. Heurísticas de Ordenación de Variables 11
4.4.2. Heurísticas de ordenación de variables estáticas 12
4.4.3. Heurísticas de ordenación de variables dinámicas 12
4.4.4. Heurísticas de ordenación valores 13
4.5. Solver 14
4.5.1. Choco 14
4.5.2. Diseño de Choco 15
4.5.3. Sintaxis de Choco 15
4.5.4. Heurísticas de Choco 16
4.5.5. Heurísticas de Selección de Variable en Choco 17
4.5.6. Heurísticas de Selección de Valor en Choco 19

5. Formalización del problema como un CSP **20**
5.1. Presentación del problema 20
5.2. Modelo 21

6. Plataforma de implementación de Ms PacMan **40**

7. Estructuras de Unión **41**
7.1. Thing.java 42
7.2. Traductor.java 42
7.3. Solver.java 42
7.4. HumanInput.java 42

8. Codificación del problema en Choco **44**
8.1. Declaraciones en Choco 44
8.2. Modelo 45

9. Pruebas y Resultados **54**

10. Conclusión **56**

Lista de Figuras

1. Solución al problema Send + More = Money. 6
2. Problema de las N-Reinas. 7
3. Resolución al problema N-Reinas con Generate & test. 8
4. Resolución al problema N-Reinas con Backtracking. 9
5. Ejemplo Backjumping. 10
6. Resolución al problema N-Reinas con Forward Checking. . . . 10
7. Resolución al problema N-Reinas con Maintaining Arc Consistency. 11
8. Funcionamiento de Choco. 16
9. Laberinto Ms. PacMan. 20
10. Modelo Ms. PacMan. 21
11. Campo de visión de Ms. PacMan. 22
12. Restricción 1. 24
13. Restricción 2. 25
14. Restricción 3. 25
15. Restricción 4. 26
16. Restricción 5. 26
17. Restricción 6. 27
18. Restricción 7. 27
19. Restricción 30. 27
20. Restricción 33. 28
21. Restricción 103. 28
22. Restricción 105. 29
23. Restricción 78. 30
24. Restricción 82. 30
25. Restricción 86. 31
26. Restricción 94. 32
27. Restricción 96. 33
28. Restricción 112. 33
29. Restricción reloj2. 36
30. Restricción contra2. 37
31. Restricción especial3. 38
32. Loop de Movimiento. 39
33. Interfaz Ms. Pac-Man 2010. 40
34. Diagrama de Clases. 41
35. Restricción 1. 46
36. Restricción 2. 47
37. Restricción 3. 48

38. Restricción 4. 49
39. Restricción 5. 50
40. Restricción 6. 51
41. Restricción reloj2. 52
42. Resultados de la competencia 2011. 55
43. Resultados obtenidos por el Solver. 55
44. Gráfico Niveles Alcanzados. 56

1. Introducción

Tradicionalmente los videojuegos han proveído un marco para el estudio de la inteligencia artificial y aprendizaje de máquinas. Las técnicas evolucionadas de computación pueden en algunos casos desarrollar agentes competitivos de alto nivel; que pueden llegar a superar agentes codificados a mano. Los juegos son diseñados para desarrollar y desafiar las habilidades cognitivas y de aprendizaje de las personas. Asimismo, existe una serie de competencias internacionales donde se evalúan distintos métodos para simular la operación humana en videojuegos por medio de la inteligencia artificial. Ms. PacMan es un videojuego arcade producido por Midway que comenzó como versión no autorizada del tradicional PacMan, para luego ser oficialmente licenciado por Namco. El objetivo de este juego es lograr que Ms. PacMan capture todas las píldoras de un laberinto evitando ser atrapada por un conjunto de fantasmas.

El propósito de este proyecto es simular el comportamiento de Ms. PacMan de tal manera que supere el laberinto dentro del juego. Esto se realiza modelando cada movimiento de Ms. PacMan como un problema de satisfacción de restricciones, comúnmente conocido como Constraint Satisfaction Problem (CSP). Para la resolución de este problema se utiliza la programación con restricciones, Constraint Programming (CP) en inglés. Esta es una metodología de software utilizada para la resolución eficiente de problemas, en particular combinatorios, donde las relaciones entre las variables son expresadas en términos de ecuaciones o restricciones. En la actualidad, esta tecnología se utiliza exitosamente en diversas áreas de aplicación especialmente en planificación y programación de tareas.

Este documento se dispone de la siguiente manera. En una primera etapa, se plantean los objetivos del proyecto y el estado del arte de la problemática. En el capítulo 5 se realiza una introducción a CP seguido de la implementación de la solución al problema planteado. El capítulo 6 presenta la interfaz de Ms. PacMan que se usó para aplicar el modelo desarrollado. El capítulo 7 explica las estructuras desarrolladas y utilizadas para la interfaz del juego. En el capítulo 8 se presenta cómo se trabajo la resolución del Ms. PacMan problem utilizando un modelo de CSP desarrollado en Java bajo el Solver Choco. El extracto de resultados en la prueba del funcionamiento del Solver se muestran en el capítulo 9. Finalmente en el capítulo 10 se presentan las conclusiones del proyecto.

2. Definición de Objetivos

2.1. Objetivos Generales

Modelar y resolver el Ms. PacMan Problem utilizando programación con restricciones.

2.2. Objetivos Específicos

- Modelar el Ms. PacMan problem.
- Implementar el modelo CSP con Choco solver.
- Adaptar el solver a la interfaz gráfica del juego.
- Realizar experimentos con el fin de superar el laberinto.

3. Estado del Arte

Ms. PacMan es un videojuego arcade producido por Midway y empezó como una secuela no autorizada del PacMan. Fue lanzado en Norteamérica en 1981 y se volvió uno de los videojuegos más populares, llevándolo a ser adaptado por el licenciador de PacMan Namco como un título oficial. El juego introduce una protagonista, nuevos ambientes de juego y menos jugabilidad que antes [4].

La mayor diferencia con respecto al PacMan original, es que, al contrario que este, Ms. PacMan es un juego no determinista. Además, es bastante difícil para la mayoría de los jugadores. Esto hace que desarrollar un agente controlado por medio de la inteligencia artificial para este juego se convierta en un auténtico reto por lo que han surgido varios trabajos de investigación, los cuales han utilizado distintas versiones del juego (la versión original, Ms. PacMan y simuladores de costumbre).

Diferentes trabajos de investigación sobre PacMan han utilizado varias versiones del juego (la versión original, Ms. PacMan y simuladores de costumbre), lo que complica el problema de comparar las diferentes propuestas [2, 10, 11]. Los trabajos en este campo se pueden clasificar en dos grandes grupos de técnicas: los que hacen uso de la Inteligencia Artificial, en todo o en parte, y las técnicas que implementan agentes escritos en código. En general, el segundo enfoque se basa en la aplicación de conjuntos de reglas y produce mejores resultados que las soluciones dadas por el primer grupo. Pac-Man y Ms. Pac-Man han sido una herramienta de investigación en conjunto con muchos trabajos en el campo de los videojuegos permitiendo investigar y/o resolver problemas relacionados a la inteligencia artificial y resolución de problemas informáticos. También ha sido motivo de desarrollo de nuevas formas de resolución del mismo juego y hasta competencias por lograr un mejor performance en puntuaciones o mejoras al mismo juego.

Tanta fuerza ha tomado esto que Simon M. Lucas, doctor en informática y miembro de IEEE (*Institute of Electrical and Electronics Engineers*) Computational Intelligence Society lleva varios años organizando una competencia internacional denominada "Ms. PacMan Competition" [6] que tiene por objetivo desarrollar la inteligencia artificial capaz de controlar a Ms. PacMan con el fin de alcanzar la mayor cantidad de puntaje posible [5]. La competencia en si está enfocada a los jugadores que utilicen inteligencia computacional para solucionar el problema junto con la interacción de un

mapa de pixeles programado en Java que es entregado por los organizadores, y está abierta a cualquier tipo de algoritmo que sea capaz de lograr el objetivo. Cabe destacar que este juego ha sido implementado en una serie de plataformas de programación y ha utilizado muchos modelos de resolución, como por ejemplo optimización basada en Ant-Colonies.

En el caso de este proyecto se utilizó CP para la resolución del problema, y al ser un problema de satisfacción, el objetivo principal es simular los movimientos de Ms. PacMan con el fin de que tome decisiones correctas y con ello lograr superar el laberinto, pero a diferencia de las competencias, no está enfocado en obtener un puntaje alto.

4. Programación con restricciones

La Programación con restricciones (CP) es un paradigma de programación en informática, donde las relaciones entre las variables son expresadas en términos de restricciones o ecuaciones para un problema y posteriormente pretende encontrar soluciones que logren satisfacer cada una de ellas. La programación con restricciones puede resolver problemas de diversas áreas, no sólo los relacionados con la informática. Algunos de los campos que puede comprender esta tipología de problemas son la investigación de operaciones, inteligencia artificial, problemas matemáticos, etc. Incluso situaciones de la vida cotidiana están sumergidos en la problemática de las restricciones, asi como la organización de turnos de un hospital, la organización de eventos, distribución de recursos o el tipo de locomoción a tomar [9].

4.1. Problema de satisfacción de restricciones

Un problema de satisfacción de restricciones puede ser representado mediante una terna $\langle X,D,C \rangle$ donde:

- X es un conjunto de n variables $\{x_1,x_2,...,x_n\}$
- $D = \langle D_1, ...,D_n \rangle$ donde D_i es el dominio que contiene los posibles valores que pueden asignarse a la variable x_i.
- $C = \{c_1, c_2,...,c_p\}$ es un conjunto finito de restricciones. Cada restricción k-aria c_i está definida sobre un conjunto de k variables $var(c_i) \subseteq X$, denominado su ámbito, y restringe los valores que dichas variables pueden tomar a la vez.

El CSP está resuelto cuando se han satisfecho todas sus restricciones. Si el CSP tiene al menos una solución se dice que es un problema consistente, en caso contrario se dice que corresponde a un problema inconsistente [3].

Los CSPs se pueden dividir en categorías dependiendo de los valores de sus dominios, por ejemplo:

- Un CSP con dominio finito involucra sólo valores en enteros.
- Un CSP numérico (NCSP) involucra valores en reales.

4.2. Ejemplos CSP

4.2.1. Send + More = Money

Un ejemplo típico es el problema *"Send + More = Money"* el cual consiste en asignar un valor numérico distinto a cada letra *{S,E,N,D,M,O,R,Y}* dentro del dominio {0..9} de manera que satisfaga la ecuación [9]:

$$\begin{array}{r} S\ E\ N\ D \\ +\ M\ O\ R\ E \\ \hline M\ O\ N\ E\ Y \end{array}$$

- Variables: $\{S,E,N,D,M,O,R,Y\} \in [0,9]$
- Restricciones:
  - $10^3(S+M)+10^2(E+O)+10(N+R)+D+E=10^4M+10^3O+10^2N+10E+Y$
  - $\{S,E,N,D,M,O,R,Y\}$ todas las letras con diferentes valores
- Solución: La figura 1 muestra el resultado al problema CSP

$$\begin{array}{r} 9\ 5\ 6\ 7 \\ +\ 1\ 0\ 8\ 5 \\ \hline 1\ 0\ 6\ 5\ 2 \end{array}$$

Figura 1: Solución al problema Send + More = Money.

4.2.2. N-Reinas

Es un ejemplo clásico de formalización de CSP. Consiste en ubicar N Reinas dentro de un tablero de ajedrez de dimensiones $N \times N$ de tal manera de que ninguna reina pueda interferir en el movimiento de las demás reinas y tampoco puedan ser comidas. Para cumplir esto no puede haber 2 reinas en la misma fila, columna o diagonal. Con estos datos podemos formular el problema de la siguiente manera [9]:

- Variables: $\{Q_i\} i \in [1,N]$
- Dominio: $\{1,2,\ldots,N\}$ para todas las variables
- Restricciones: $(\forall\ Q_i,\ Q_j,\ i \neq j)$:
  - $Q_i \neq Q_j$ (filas)

- $Q_i + i \neq Q_j + j$ (diagonal 1)
- $Q_i - i \neq Q_j - j$ (diagonal 2)

La figura 2 muestra una representación de satisfacción para un tablero de ajedrez de 4×4 utilizando la fórmula anterior.

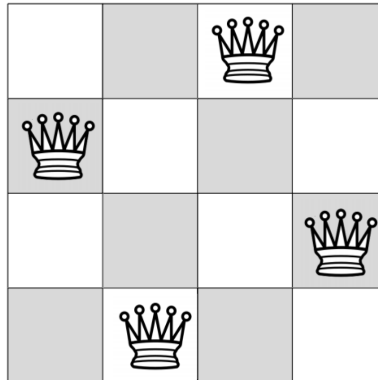

Figura 2: Problema de las N-Reinas.

4.3. Algoritmos de búsqueda

Los métodos de búsqueda se centran en explorar el espacio de estados del problema, también para ofrecer buenos, rápidos y/o eficientes resultados, ya sea para un CSP o Problema de optimización de Restricciones (Constraint Optimization Problem, COP), para ayudandar a que los tiempos de cómputo sean menores y ahorren recursos.

4.3.1. Generate and Test

La manera más sencilla, aunque poco eficiente, de encontrar todas las soluciones de un CSP es la que implementa Generate & test (GT), ya que esta genera de forma sistemática todas las posibles asignaciones completas. Cuando finaliza de generar una asignación completa, comprueba si esta asignación es una solución (es decir, comprueba si satisface todas las restricciones). En terminología de árboles, GT es un algoritmo que recorre el árbol de búsqueda en profundidad prioritaria (recorrido primero en profundidad). La ineficiencia en GT se debe a que genera muchas asignaciones completas que no cumplen con las restricciones del caso [1]; esto se puede ver ejemplificado en la figura 3.

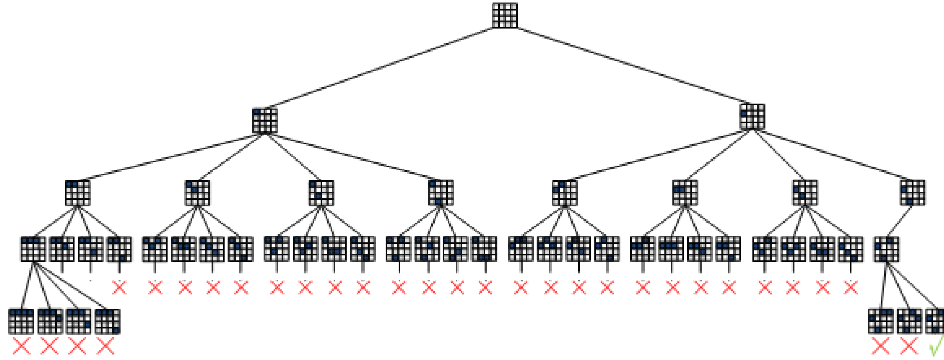

Figura 3: Resolución al problema N-Reinas con Generate & test.

4.3.2. Backtracking

El algoritmo Backtracking (BT) trabaja similar al algoritmo GT, pero lo mejora de la siguiente forma: cada vez que se asigna un nuevo valor a la variable actual *(*X_i*)*, se comprueba si es consistente con los valores que hemos asignado a las variables pasadas. Si no lo es, se abandona esta asignación parcial, y se asigna un nuevo valor a X_i. Si ya se ha agotado todos los valores de D_i, BT retrocede para probar otro valor para la variable X_i. Si se ha agotado D_{i-1} retrocede al nivel D_{i-2}, y así sucesivamente hasta encontrar una asignación de un valor a una variable que es consistente con las variables pasadas o hasta que se demuestra que no hay más soluciones. Es decir, BT recorre el árbol utilizando búsqueda primero en profundidad y, en cada nodo, comprueba si la variable actual es consistente con las variables pasadas. Si detecta inconsistencia, descarta la asignación parcial actual, puesto que no es parte de ninguna asignación completa que sea solución. De esta forma, se ahorra recorrer el subárbol que cuelga de esta asignación parcial [1]; esto se puede ver ejemplificado en la figura 4.

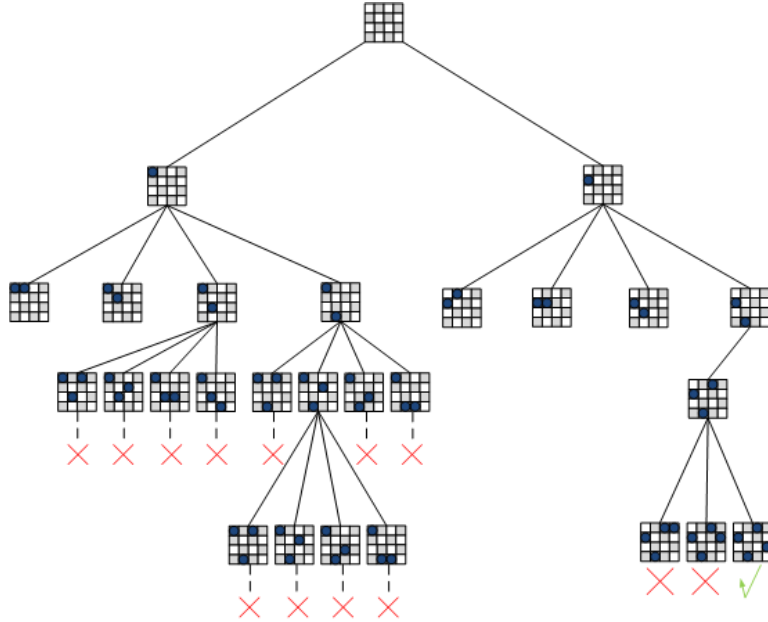

Figura 4: Resolución al problema N-Reinas con Backtracking.

4.3.3. Backjumping

El algoritmo Backjumping (BJ) es un algoritmo de backtracking no cronológico que mitiga el trashing. BJ recorre el árbol como BT, pero cuando detecta que la asignación que hemos hecho a la variable actual X_i es inconsistente con la asignación que tenemos, en la misma rama, para alguna variable pasada X_h, guarda el nivel de esta variable; es decir, el algoritmo recuerda en qué nivel se ha detectado una inconsistencia. En caso de que no se haya detectado ninguna inconsistencia, se guarda el valor i-1. A continuación, BJ asigna a X_i un nuevo valor de D_i, y también guarda el nivel en el que se ha detectado una inconsistencia. Cuando se agota D_i, BJ retrocede al máximo nivel X_i, en lugar de retroceder a la variable más reciente instanciada como en BT. El hecho de cambiar la instanciación de X_h nos puede permitir encontrar una instanciación que sea consistente con X_i. Sin embargo, cambiar la instanciación de cualquiera de las variables que hay entre X_i y X_h es inútil, puesto que no ha cambiado la razón por la que hemos detectado un bloqueo para X_i [1]; esto se puede ver ejemplificado en la figura 5.

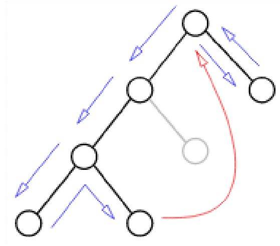

Figura 5: Ejemplo Backjumping.

4.3.4. Forward checking

El algoritmo Forward Checking (FC) es un algoritmo que fuerza a que, en cada nodo, no haya ninguna variable futura que tenga algún valor de su dominio que sea inconsistente con el valor que alguna variable pasada tiene asignado en la rama que estamos considerando. Para conseguirlo, FC elimina los valores de los dominios de variables futuras que no cumplen con esta condición; si alguno de estos dominios queda vacio, FC hace backtracking. Esta condición es equivalente a exigir que sea arco consistente en conjunto de restricciones en las que interviene la variable actual y una variable futura. Es decir, FC funciona como BT, pero realiza una consistencia de arcos entre restricciones que conectan la variable actual y una variable futura cada vez que se asigna un valor a la variable actual [1]; esto se puede ver ejemplificado en la figura 6.

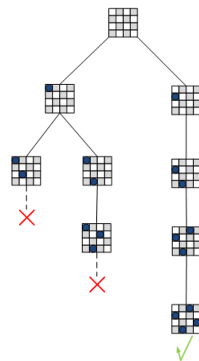

Figura 6: Resolución al problema N-Reinas con Forward Checking.

4.3.5. Maintaining Arc Consistency

Maintaining Arc Consistency (MAC) funciona como FC, pero en cada nodo del árbol de búsqueda se aplica al algoritmo Arc Consistency para alcanzar arco consistencia entre todas las restricciones del problema. Es decir, cada nodo, se simplifica mediante arco consistencia el CSP que estamos considerando. Mientras que en FC exigimos arco consistencia parcial, en MAC exigimos arco consistencia total en cada nodo. Existen otros algoritmos (por ejemplo, partial lookahead y full lookahead) que, en cada nodo, exigen un grado de consistencia local intermedio entre FC y MAC [1]; esto se puede ver ejemplificado en la figura 7.

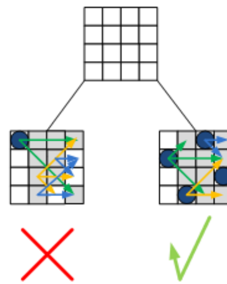

Figura 7: Resolución al problema N-Reinas con Maintaining Arc Consistency.

4.4. Estrategias de Enumeración

Encontrar una solución a un CSP que dé satisfacción plena a todas las restricciones involucra necesariamente llevar adelante un proceso de búsqueda bastante costoso, que en la actualidad está siendo abordado con las técnicas de propagación de restricciones y enumeración ya descritas. En este proceso de búsqueda, específicamente en la fase de enumeración, la estrategia utilizada como guía tiene un efecto importante en el rendimiento del proceso de resolución [9].

4.4.1. Heurísticas de Ordenación de Variables

El orden en el cual las variables son asignadas durante la búsqueda puede tener gran impacto significativo en el tamaño del espacio de búsqueda explorado. Generalmente las heurísticas de ordenación de variables tratan de seleccionar lo antes posible las variables que más restringen a las demás. La

intuición es tratar de asignar lo antes posible las variables más restringidas y de esa manera identificar las situaciones sin salida lo antes posible y así reducir el número de vueltas atrás. La literatura presenta diferentes caminos para el desarrollo de esta selección, la cual depende de la naturaleza del problema, para dirigir una eficiente propagación de restricciones. La ordenación de variables puede ser estática o dinámica.

4.4.2. Heurísticas de ordenación de variables estáticas

Las heurísticas de ordenación de variables estática generan un orden fijo de las variables antes de iniciar la búsqueda, se basan en la información global que deriva de la topología del grafo de restricciones original que representa el CSP. Este tipo de heurísticas de ordenación de variables estáticas no cambia su forma de analizar los posibles caminos de una manera total.

4.4.3. Heurísticas de ordenación de variables dinámicas

El problema de los algoritmos de ordenación de variables estáticos es que no tienen en cuenta los cambios en los dominios o relaciones de las variables causados por la propagación de las restricciones durante la búsqueda. Por ello, estas heurísticas generalmente se utilizan en algoritmos de comprobación hacia atrás donde no se lleva a cabo la propagación de las instanciaciones que se van realizando. Las heurísticas de ordenación de variables dinámica pueden cambiar el orden de selección de las variables de forma dinámica durante el proceso de búsqueda, cada vez que requiere la instanciación de una variable. La ordenación se basa en las instanciaciones ya realizadas y en el estado de la red en cada momento de la búsqueda. Se han propuesto varias heurísticas de ordenación de variables dinámicas. Las más comunes se basan en el principio de primer fallo que sugiere que "para tener éxito deberíamos intentar primero donde sea más probable que falle". De esta manera las situaciones sin salida pueden identificarse antes y además se ahorra espacio de búsqueda.

- **First-fail**: se selecciona la variable con el dominio más pequeño. Esta elección es inspirada por la suposición que el suceso puede ser logrado por probar primeramente las variables que tienen una gran posibilidad de fallar, en éste caso, los valores con un pequeño número de alternativas disponibles. Esta heurística es conocida por su aplicación mucho más adaptable en dominios discretos.

- **Most-constrained variable**: esta alternativa puede ser justificada por el hecho que la instanciación de dicha variable debe conducir a

un árbol de dominios más grande a través de la propagación de la restricción.

- **Reduce-first**: la selección de la variable con el dominio más grande. Esta heurística es conocida por ser la más adoptada para los dominios continuos.

- **Round robin**: es usada para seleccionar alguna variable en orden racional y equitativo, para instanciar desde la primera variable definida en el modelo hasta la última la última [9].

4.4.4. Heurísticas de ordenación valores

Estas heurísticas tienen como objetivo seleccionar el valor más prometedor para cada variable en su dominio de instanciación. La idea básica es seleccionar el valor de la variable actual que más probabilidad tenga de llevarnos a una solución, es decir, identificar la rama del árbol de búsqueda que sea más probable que obtenga una solución. La mayoría de las heurísticas propuestas tratan de seleccionar el valor menos restringido de la variable actual, es decir, el valor que menos reduce el número de valores útiles para las futuras variables, o alternativamente, el que deja los dominios mayores. Esto sigue la intuición de que un subproblema es más probable que tenga solución cuantos más valores tengan las variables que quedan por instanciar en sus dominios. Entre las más utilizadas podemos mencionar:

- **min-conflicts**: es una de las heurísticas de ordenación de valores más conocida. Esta heurística ordena los valores de acuerdo a los conflictos que generan con las variables aún no instanciadas. Esta heurística asocia a cada valor de la variable actual, el número total de valores en los dominios de las futuras variables adyacentes con la actual que son incompatibles con a. El valor seleccionado es el asociado a la suma más baja.

- **max-domain-size**: alternativamente a la anterior, esta heurística selecciona el valor de la variable actual que deja el máximo dominio en las variables futuras.

- **weighted-max-domain-size**: esta heurística especifica una manera de romper empates en el método anterior, en el caso de que existan varios valores de la variable actual que dejen el mismo máximo dominio en las variables futuras. Entonces, se basa en el número de futuras variables que tienen la mayor talla de dominio.

- **point-domain-size**: esta heurística asigna un peso (unidades) a cada valor de la variable actual dependiendo del número de variables futuras que se quedan con ciertas tallas de dominios [9].

4.5. Solver

Los lenguajes y sistemas para el modelado y resolución de un CSP han sido desarrollados bajo diferentes principios. Por ejemplo, el uso de la programación lógica como el soporte para el paradigma Programacion Lógica con Restricciones (CLP) o el uso de objetos para la simulación de problemas sujetos a restricciones. Desde el punto de vista de la implementación, diferentes métodos han sido propuestos, por ejemplo, la integración de librerías sobre un lenguaje de programación o la construcción de un nuevo lenguaje de programación con soporte para restricciones. Recientemente se ha propuesto la idea de desarrollar un lenguaje de programación puro en vez de un lenguaje de programación común buscando esencialmente proveer un lenguaje más entendible para el usuario.

Las librerías para CP proveen un lenguaje para la declaración de problemas sujeto a restricciones. Estas normalmente son implementadas mediante clases y métodos específicos en un determinado lenguaje de programación, por ejemplo, una determinada clase puede ser usada para declarar variables y métodos que definen la relación entre ellos. La gran particularidad de estos métodos es la no necesidad de implementación de un nuevo lenguaje para resolver un problema sujeto a restricciones. A continuación se presentan algunos ejemplos de Solvers.

A continuación se presenta el solver que se ocupó para resolver el MS. PacMan Problem.

4.5.1. Choco

El Solver Choco fue desarrollado en Francia por Caseau Yves y Laburthe Francois. Este es un lenguaje de alto nivel funcional y orientado a objetos que contiene en reglas de avanzada capacidad. Claire se propuso por objetivo permitir al programador expresar en pocas y cortas líneas de código algoritmos complejos siempre de una manera elegante y sobre todo legible. Su diseño fue pensado para aplicaciones avanzadas que involucran modelos de datos complejos, con procesos y reglas.

La biblioteca de Choco se utiliza siempre en el mismo escenario: en primer lugar un problema se crea. A continuación, las variables relacionadas con el problema se agregan y finalmente las restricciones son declaradas para finalmente ser resuelto. Una vez terminado, el problema puede ser manipulado, lo que permite la profunda exploración de un problema en una variedad de maneras. Debido a su corta edad, Choco, en la actualidad sólo es compatible con variables de tipo entero y hay límites en las operaciones disponibles en el manejo de esos números enteros.

A diferencia de algunos solvers de programación con restricciones como por ejemplo Sicstus Prolog, Choco Solver es una herramienta de código abierto en inglés conocido como open-source distribuido bajo licencia BSD y auspiciado y alojado en sourceforge.net. Fue especialmente diseñado para la investigación y el aprendizaje académico [9].

4.5.2. Diseño de Choco

Desde el punto de vista de los usuarios ya sean académicos o investigadores en su mayoría, la gran atracción hacia el uso de esta librería es su facilidad de programar en ella, lo que lo hace cercano y simple de comprender; los usuarios expresan que su lenguaje es claro, legible y poderoso, haciendo que los programas que se implementan bajo su apoyo sean concisos, minimizando la probabilidad de errores o simplemente evitando que estos se propaguen en la creación de un código. Choco es una librería de Java, lo que provee una clara separación entre el modelado y la resolución. La figura 8 presenta la arquitectura de la librería Choco. La primera parte, desde el punto de vista del usuario, es dedicada para expresar el problema. La idea de esto es manipular las variables y las restricciones de estas últimas con el objeto de abordar un problema de la manera más fácil posible. La segunda parte es dedicada para resolver el problema realmente. En la figura se muestra también como la solución gestiona la memoria para la búsqueda basada en árbol.

4.5.3. Sintaxis de Choco

Dentro de las características de Choco como Solver, se encuentran la de modelador de problemas y también la de un Solver de programación con restricciones habilitado como una librería Java. Sin embargo, esta arquitectura permite adicionar complementos de solvers no necesariamente basados en CP. Choco es un modelador de problemas que permite manipular una

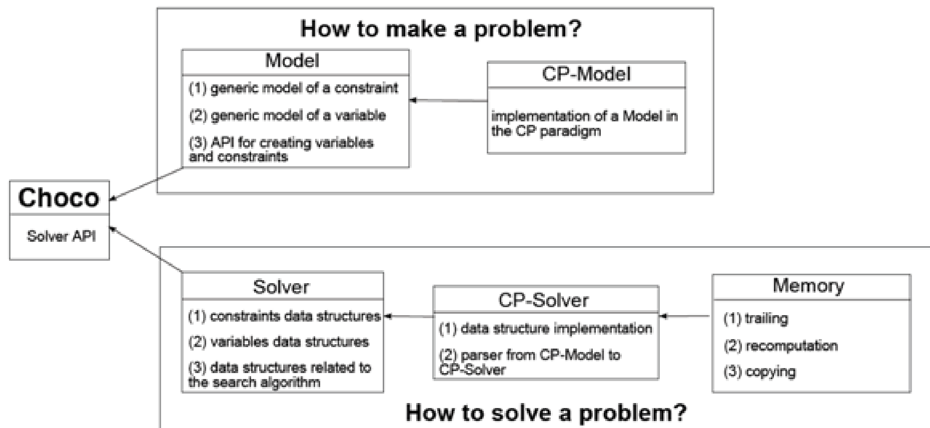

Figura 8: Funcionamiento de Choco.

amplia variedad de tipos de datos como lo son:

- Variables enteras
- Conjuntos de variables representadas como variables enteras
- Variables reales, representando variables que toman ciertos valores en intervalos de números flotantes.
- Expresiones que representan un entero o real basado en una expresión que utiliza operadores como *plus, mult, minus, scalar, sum,* etc.

Los modelos de Choco aceptan más de 70 restricciones, siempre que la llamada del Solver sea un programa basado en CP:

- Restricciones clásicas aritméticas en reales o enteros como: *equal, les sor equal, greater or equal,* etc.
- Operación de números booleanos
- Tablas de restricciones definidas como un conjunto de tuplas

4.5.4. Heurísticas de Choco

Un ingrediente clave de cualquier enfoque de una estrategia de búsqueda inteligente es el uso de heurísticas de selección de variable y valor. En los

algoritmos backtracking y Branch and Bound los enfoques la búsqueda se organizan como un árbol de enumeración, en la que cada nodo corresponde a un sub-espacio de la búsqueda, y cada nodo hijo es una subdivisión del espacio de su nodo padre. El árbol de búsqueda se construye progresivamente mediante la aplicación de una serie de estrategias de ramificación que determinan cómo se subdivide el espacio en cada nodo creado. Básicamente, según lo anterior, una estrategia de búsqueda en Choco es una composición de objetos en una estrategia de ramificación, cada uno definido en un conjunto de variables de decisión. Las estrategias más comunes de ramificación se basan en la asignación de una variable seleccionada por medio de una heurística de selección de variable, a uno o varios valores seleccionados por medio de una heurística de selección de valor.

4.5.5. Heurísticas de Selección de Variable en Choco

Una heurística de selección de variable define básicamente, dependiendo de las restricciones involucradas, la manera de elegir una variable no instanciada como la siguiente variable de decisión. Las heurísticas de selección de variable disponibles en la actualidad en Choco son las siguientes:

- CompositiveIntVarSelector: Básicamente es una composición de heurísticas de variable, que tiene por objeto seleccionar una restricción acorde a la primera de las heurísticas de selección de valor para luego asignar un valor entero involucrado en una restricción de la segunda de las heurísticas de variable. Esta composición de heurísticas es aplicable a las heurísticas de selección de dominio más pequeño como también las heurísticas de selección de dominios más amplios, todas estas explicitadas más adelante en este documento.

- CyclicRealVarSelector: Selecciona las variables reales en el orden en que aparecen en un arreglo x, en un camino cíclico hasta que todas ellas sean instanciadas.

- LexIntVarSelector: Selecciona la variable entera acorde a una primera heurística de variable, hasta que esa razón de selección se rompa con la elección de una segunda heurística de selección de variable.

- MaxDomain: Selecciona una variable de tipo entero con el dominio más grande.

- MaxDomSet: Selecciona un conjunto de variables con el dominio más grande.

- MaxRegret: Selecciona una variable de tipo entero con la diferencia más grande entre los dos valores más pequeños en esos dominios.
- MaxRegretSet: Selecciona un conjunto de variables con la diferencia más grande entre los dos valores más pequeños de ese conjunto.
- MaxValueDomain: Selecciona una variable de tipo entero con el valor más grande en su dominio.
- MaxValueDomSet: Selecciona un conjunto de variables con el valor más grande.
- MinDomain: Selecciona la variable de tipo entero con el dominio más pequeño.
- MinDomSet: Selecciona un conjunto de variables con el dominio más pequeño.
- MinValueDomain: Selecciona una variable de tipo entero con el valor más pequeño de su dominio.
- MinValueDomSet: Selecciona una variable de tipo entero con el valor más pequeño de un grupo seleccionado.
- MostConstrained: Selecciona una variable de tipo entero involucrada en el más grande número de restricciones posibles en el Solver.
- MostConstrainedSet: Selecciona un conjunto de variables involucradas en el más grande número de restricciones en el Solver.
- RandomIntVarSelector: Selecciona una variable de tipo entero de forma aleatoria.
- RandomSetVarSelector: Selecciona un conjunto de variables de tipo entero de manera aleatoria.
- StaticSetVarOrder: Selecciona un conjunto de variables en orden que aparezcan en un arreglo predefinido por el Solver.
- StaticVarOrder: Selecciona una variable de tipo entero en orden que aparezcan en un arreglo predefinido por el Solver.

4.5.6. Heurísticas de Selección de Valor en Choco

Básicamente las heurísticas de selección de valor se caracterizan, en una primera instancia, por ser partes de una estrategia de búsqueda y de computo, donde su principal tarea es instanciar las variables dependiendo de sus dominios conformes a los valores consultados en las llamadas a las variables. Actualmente las heurísticas de selección de valor disponibles en Choco son las siguientes:

- MaxVal: La heurística de selección de valor MaxVal selecciona el valor más grande en el dominio de una variable de tipo entero.
- MidVal: La heurística de selección de valor MidVal selecciona el valor más cercano (igual o mayor) al punto medio de una variable de tipo entero.
- MinVal: La Heurística de selección de valor MinVal selecciona el valor más pequeño en el dominio de una variable de tipo entero.

5. Formalización del problema como un CSP

5.1. Presentación del problema

En el juego Ms. PacMan, el jugador debe comer puntos (pills o píldoras) y esquivar a los fantasmas que entorpecen su camino (que al contacto con uno, se pierden vidas) para evitar que este complete su misión, que es comer todas las píldoras de la interfaz. Con esto como base la investigación debe basarse en buscar una solución de satisfacción mediante la programación con restricciones para lograr que Ms PacMan capture todas las píldoras del laberinto (véase figura 9), y así para facilitar el trabajo, los valores del laberinto se han rescatado y traspasado a una matriz de 31x28 en un primer momento de la investigación, donde cada uno de los elementos tendrán un valor representativo para las acciones del Solver, igualando los valores del código de la interfaz en la matriz mencionada. En el transcurso de la investigación se decisión de agrandar la matriz global a 37x34, que contiene la matriz del laberinto rodeada de un borde de 3x3 con valores 0 (véase figura 10). Estos valores serán descritos a continuación en conjunto con las restricciones preliminares que se ocuparan para resolver el Ms. PacMan Problem.

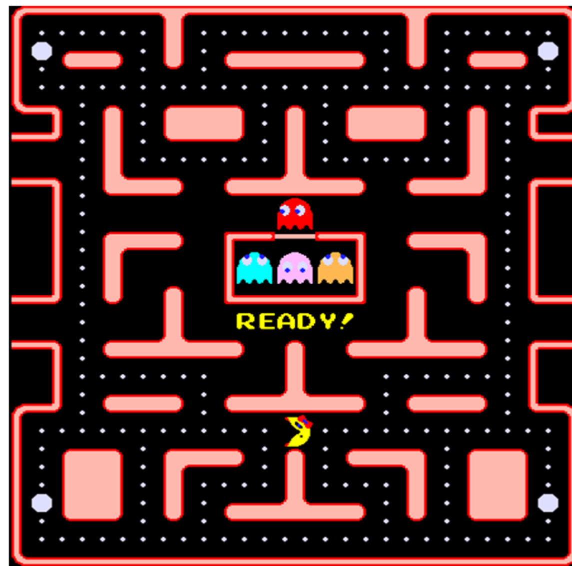

Figura 9: Laberinto Ms. PacMan.

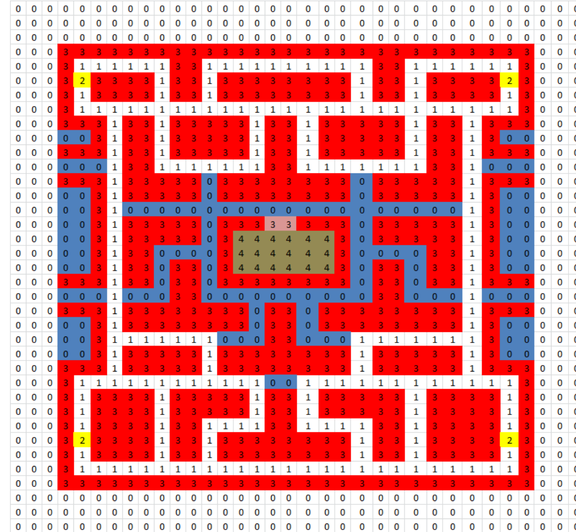

Figura 10: Modelo Ms. PacMan.

5.2. Modelo

Como se ha mencionado anteriormente, el modelo aplicado al Ms. PacMan Problem, consiste en traspasar el laberinto del juego a una matriz de 31x28, en donde se especifican cada una de las constantes que posee el modelo (murallas, pills, fantasmas, etc.). Además de esto se implementó una matriz de visión de 7x7 (véase figura 11) para efectuar las decisiones de los movimientos que realizará, cuando se necesita buscar una píldora lejana si no existe ninguna cerca, por ejemplo si el Solver está recorriendo la base del laberinto y no existen más pills, se ocupa la matriz copia del juego para “ver” donde se encuentran otras pills faltantes dentro del laberinto; antes con el laberinto de dimensión 31x28 Ms. PacMan no podía cruzar los túneles que existen en el laberinto de esta manera la matriz creció a 37x34 correspondiente a la visión global de Ms. PacMan final, esta matriz le permitirá evaluar la mejor ruta posible en conjunto con la matriz de visión. La declaración de variables, constantes y restricciones sigue a continuación.

Cabe agregar que el modelo desarrollado para el Ms PacMan Problem, no busca terminar las etapas del juego como objetivo principal, si no que el Solver desarrollado debe encontrar una solución que satisfaga para cada búsqueda de caminos posibles, es decir, que Ms PacMan elija un camino

correcto (avanzar en alguna dirección ó no ser atrapada por un fantasma) a cada ejecución del Solver, es decir, el Solver funciona de una manera recursiva; primero evalúa los valores a su alrededor toma una decisión y la ejecuta, luego después del movimiento, nuevamente se ejecuta el Solver y así sucesivamente. La ejecución correcta del Solver sucesivamente provoca que Ms PacMan pueda realizar los movimientos para alcanzar el término de la etapa o el juego.

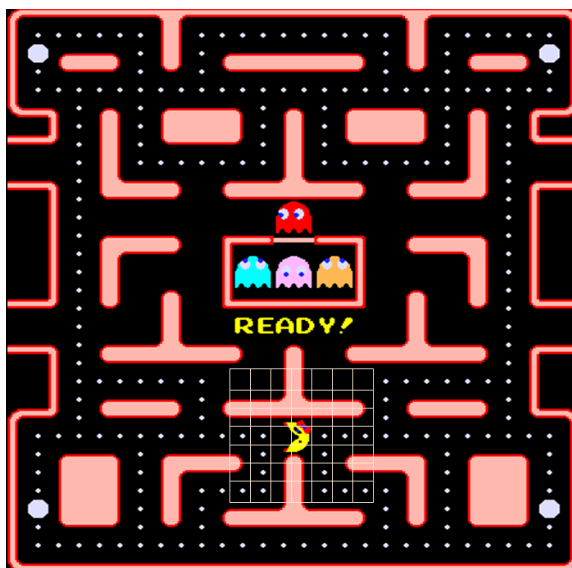

Figura 11: Campo de visión de Ms. PacMan.

- Variables
  - Up $\in$ [*0..1*]
  - Down $\in$ [*0..1*]
  - Left $\in$ [*0..1*]
  - Right $\in$ [*0..4*]
    Los valores del dominio corresponden a los valores que pueden tomar las variables. Se asignará 1 a la variable en caso de que

Ms. PacMan tenga que moverse en esa dirección y 0 en caso de lo contrario.

  - Loop $\in$ [*0..1*]
    Se asignara 1 a la variable en caso de que Ms. PacMan se este moviendo por camino sin píldoras y 0 en caso de lo contrario.

- Constantes

  - NoPills(NP)=0
  - Pills=1
  - PowerPills(PP)=2
  - Wall=3
  - Ghost=4
  - CurrentX
    Esta variable representa la posición X de Ms. PacMan dentro de la matriz de visión.
  - CurrentY
    Esta variable representa la posición Y de Ms. PacMan dentro de la matriz de visión.
  - mv
    Esta es la matriz que representa la visión o campo de evaluación de Ms. PacMan.
  - mov_ant$\in$ [*0..4*]
    Esta variable guarda el movimiento anterior realizado por Ms. PacMan, tendrá valor 0 si no tiene movimiento anterior, 1 en el caso de ser arriba, 2 en el caso de derecha, 3 en el caso de abajo y 4 en el caso de izquierda.
  - cont
    Esta variable es un contador del camino recorrido sin píldoras, sumara 1 al cont mientras la variable Loop tenga valor 1.
  - loop
    Esta variable tendra valor 1 al obtenerse el número pedido a la variable cont, lo que provocará el cambio de la toma de desición de movimiento. La variable tendrá valor 0 en caso contrario.

- Restricciones
  Para poder solucionar el Ms. PacMan Problem se creó un universo de 128 restricciones, de las cuales 32 son para cada una de las 4 direcciones de movimiento posibles en el juego (arriba, abajo, izquierda, derecha) y solo se diferencian entre ellas en las coordenadas respectivas a cada dirección. Las restricciones se pueden dividir en 5 grandes grupos:

  - Restricciones simples
    restricciones que no tienen prioridad de movimiento, preferentemente utilizadas para la toma de decisión de movimiento en caminos vacíos (sin píldoras). Estas restricciones se presentan a continuación:

    - Restricción 1:
      Esta restricción aplica si Ms. PacMan tiene píldoras, píldoras de poder o no píldoras en los 3 casilleros siguientes a la derecha, puede avanzar a la derecha. Esto se ve reflejado en la figura 12.

```
c1:if((mv[CurrentX][CurrentY+1]=(0 or 1 or 2)
or(mv[CurrentX][CurrentY+1]≠4))
and(mv[CurrentX][CurrentY+2]=(0 or 1 or 2)
or(mv[CurrentX][CurrentY+2]≠4))
and(mv[CurrentX][CurrentY+3]=(0 or 1 or 2)
or(mv[CurrentX][CurrentY+3]≠4)))
```

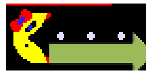

Figura 12: Restricción 1.

    - Restricción 2:
      Esta restricción aplica si Ms. PacMan tiene píldoras, píldoras de poder o no píldoras situadas en los 3 casilleros siguientes en forma de "L" hacia la derecha y abajo, puede moverse a la derecha. Esto se ve reflejado en la figura 13.

```
c2:if((mv[CurrentX][CurrentY+1]=(1 or 2 or 3)
or(mv[CurrentX][CurrentY+1]≠4))
```

```
and(mv[CurrentX][CurrentY+2]=(0 or 1 or 2)
or(mv[CurrentX][CurrentY+2]≠4))
and(mv[CurrentX+1][CurrentY+2]=(0 or 1 or 2)
or (mv[CurrentX+1][CurrentY+2]≠4)))
```

Figura 13: Restricción 2.

- Restricción 3:
  Esta restricción aplica si Ms. PacMan tiene píldoras, píldoras de poder o no píldoras situadas en los 3 casilleros siguientes en forma de "L" hacia la derecha y arriba, puede moverse a la derecha. Esto se ve reflejado en la figura 14.

```
c3:if((mv[CurrentX][CurrentY+1]=(0 or 1 or 2)
or(mv[CurrentX][CurrentY+1]≠4))
and(mv[CurrentX][CurrentY+2]=(0 or 1 or 2)
or(mv[CurrentX][CurrentY+2]≠4))
and(mv[CurrentX-1][CurrentY+2]=(0 or 1 or 2)
or(mv[CurrentX-1][CurrentY+2]≠4)))
```

Figura 14: Restricción 3.

- Restricción 4:
  Esta restricción aplica si Ms. PacMan tiene píldoras, píldoras de poder o no píldoras en las 2 posiciones siguientes y muro en la tercera posición a la derecha, puede moverse hacia esa dirección. Esto se ve reflejado en la figura 15.

```
c4:if((mv[CurrentX][CurrentY+1]=(0 or 1 or 2)
or(mv[CurrentX][CurrentY+1]≠4))
and(mv[CurrentX][CurrentY+2]=(0 or 1 or 2)
```

```
or(mv[CurrentX][CurrentY+2]≠4))
and(mv[CurrentX][CurrentY+3]=3)
and(mv[CurrentX-1][CurrentY+2]≠4
or(mv[CurrentX+1][CurrentY+2]≠4)))
```

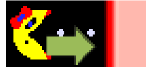

Figura 15: Restricción 4.

- Restricción 5:
  Esta restricción aplica si Ms. PacMan tiene un casillero con píldora, píldora de poder o no píldora y los 2 siguientes con muro hacia la derecha, puede avanzar hacia esa dirección. Esto se ve reflejado en la figura 16.

```
c5:if(mv[CurrentX][CurrentY+1]=(0 or 1 or 2)
or(mv[CurrentX][CurrentY+1]≠4))
and(mv[CurrentX][CurrentY+2]=3)
and(mv[CurrentX][CurrentY+3]=3)
```

Figura 16: Restricción 5.

- Restricción 6:
  Esta restricción aplica si Ms. PacMan tiene píldoras, píldoras de poder o no píldoras y fantasma situados en los 3 casilleros siguientes en forma de "L"hacia la izquierda y abajo, no puede avanzar y debe moverse en la dirección contraria. Esto se ve reflejado en la figura 17.

```
c6:if(mv[CurrentX][CurrentY-1]=(0 or 1 or 2)
or(mv[CurrentX][CurrentY-1]≠4)
and(mv[CurrentX][CurrentY-2]=(0 or 1 or 2)
or(mv[CurrentX][CurrentY-2]≠4))
and(mv[CurrentX+1][CurrentY-2=4)))
```

Figura 17: Restricción 6.

- Restricción 7:
  Esta restricción aplica si Ms. PacMan tiene píldoras, píldoras de poder o no píldoras y fantasma situados en los 3 casilleros siguientes en forma de "L"hacia la izquierda y arriba, no puede avanzar y debe moverse en la dirección contraria. Esto se ve reflejado en la figura 18.

```
c7:if(mv[CurrentX][CurrentY-1]=(0 or 1 or 2)
or(mv[CurrentX][CurrentY-1]≠4))
and(mv[CurrentX][CurrentY-2]=(0 or 1 or 2)
or(mv[CurrentX][CurrentY-2]≠4))
and((mv[CurrentX-1][CurrentY-2)=4)
```

Figura 18: Restricción 7.

- Restricción 30:
  Esta restricción aplica si Ms. PacMan tiene fantasmas en alguna de las siguientes posiciones hacia la izquierda, no puede avanzar y debe moverse en la dirección contraria. Esto se ve reflejado en la figura 19.

```
c30:if(mv[CurrentX][CurrentY-1]≠4)
and(mv[CurrentX][CurrentY-2]≠4)
and(mv[CurrentX][CurrentY-3]≠4)
```

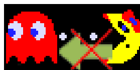

Figura 19: Restricción 30.

- Restricción 33:
  Esta restricción aplica si Ms. PacMan tiene píldora, píldora de poder o no píldora, muralla y fantasma o no píldora, respectivamente en las 3 posiciones siguientes hacia la derecha puede moverse hacia la derecha. Esto se ve reflejado en la figura 20.

```
c33:if(mv[CurrentX][CurrentY+1]=(0 or 1 or 2)
and(mv[CurrentX][CurrentY+1]≠4))
and(mv[CurrentX][CurrentY+2]=3)
and(mv[CurrentX][CurrentY+3]=(0 or 4))
```

Figura 20: Restricción 33.

- Restricción 103:
  Esta restricción aplica si Ms. PacMan tiene píldora, píldora de poder o no píldora un casillero a la izquierda y fantasma en cualquiera de los 2 casilleros siguientes hacia abajo de ese casillero, debe moverse hacia la derecha. Esto se ve reflejado en la figura 21.

```
c103:if((mv[CurrentX][CurrentY-1]=(0 or 1 or 2)
and(mv[CurrentX+1][CurrentY-1]=4)
and(c1 or c2 or c3))
or(mv[CurrentX][CurrentY-1]=(0 or 1 or 2)
and(mv[CurrentX+1][CurrentY-1]=(0 or 1 or 2))
and(mv[CurrentX+2][CurrentY-1]=4)
and(c1 or c2 or c3)))
```

Figura 21: Restricción 103.

- Restricción 105:
  Esta restricción aplica si Ms. PacMan tiene píldora, píldora de poder o no píldora un casillero a la izquierda y fantasma en cualquiera de los 2 casilleros siguientes hacia arriba de ese casillero, debe moverse hacia la derecha. Esto se ve reflejado en la figura 22.

```
c105:if((mv[CurrentX][CurrentY-1]=(0 or 1 or 2)
and(mv[CurrentX-1][CurrentY-1]=4)
and(c1 or c2 or c3))
or(mv[CurrentX][CurrentY-1]=(0 or 1 or 2)
and(mv[CurrentX-1][CurrentY-1]=(0 or 1 or 2))
and(mv[CurrentX-2][CurrentY-1]=4)
and(c1 or c2 or c3)))
```

Figura 22: Restricción 105.

- Restricciones de prioridad píldora
  restricciones idénticas a las simples, pero variando en que se evalúa sólo si en el camino hay píldoras o píldoras de poder, es decir no se evalúan los caminos vacíos para dar prioridad a lo anterior mencionado. Para este tipo de restriccion se agregaron las siguientes:

  - Restricción 78:
    Esta restricción aplica si Ms. PacMan tiene píldora o píldora de poder en el primer casillero y los dos siguientes hacia la derecha píldoras, no píldoras o píldora de poder, puede avanzar a la derecha. Esto se ve reflejado en la figura 23.

```
c78:if((mv[CurrentX][CurrentY+1]=(1 or 2)
or(mv[CurrentX][CurrentY+1]≠4))
and(mv[CurrentX][CurrentY+2]=(0 or 1 or 2)
or(mv[CurrentX][CurrentY+2]≠4))
and(mv[CurrentX][CurrentY+3]=(0 or 1 or 2)
or(mv[CurrentX][CurrentY+3]≠4)))
```

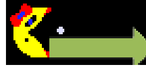

Figura 23: Restricción 78.

- Restricción 82:
  Esta restricción aplica si Ms. PacMan tiene en el primer casillero hacia la derecha píldora, píldora de poder o no píldora, en el segundo casillero píldora o píldora de poder y en el tercer casillero píldora, píldora de poder o no píldora, puede avanzar a la derecha. Esto se ve reflejado en la figura 24.

```
c82:if((mv[CurrentX][CurrentY+1]=(0 or 1 or 2)
or(mv[CurrentX][CurrentY+1]≠4))
and(mv[CurrentX][CurrentY+2]=(1 or 2)
or(mv[CurrentX][CurrentY+2]≠4))
and(mv[CurrentX][CurrentY+3]=(0 or 1 or 2)
or(mv[CurrentX][CurrentY+3]≠4)))
```

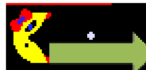

Figura 24: Restricción 82.

- Restricción 86:
  Esta restricción aplica si Ms. PacMan tiene en los dos primeros casilleros hacia la derecha píldoras, píldoras de poder o no píldoras, y en el último casillero píldora o píldora de poder, puede avanzar a la derecha. Esto se ve reflejado en la figura 25.

```
c86:if((mv[CurrentX][CurrentY+1]=(0 or 1 or 2)
or(mv[CurrentX][CurrentY+1]≠4))
and(mv[CurrentX][CurrentY+2]=(0 or 1 or 2)
or(mv[CurrentX][CurrentY+2]≠4))
and(mv[CurrentX][CurrentY+3]=(1 or 2)
or(mv[CurrentX][CurrentY+3]≠4)))
```

Figura 25: Restricción 86.

- Restricción 94:
  Esta restricción aplica si Ms. PacMan tiene en el primer casillero hacia la derecha píldora, píldora de poder o no píldora, y los 3 casilleros arriba de este último píldoras, píldoras de poder o no píldoras, puede avanzar a la derecha. En este caso se van evaluando distintas posibilidades de espacios vacíos dentro de los 3 casilleros hacia arriba. Esto se ve reflejado en la figura 26.

```
c94:if(((mv[CurrentX][CurrentY+1]=(0 or 1 or 2)
or(mv[CurrentX][CurrentY+1]≠4))
and(mv[CurrentX-1][CurrentY+1]=(1 or 2)
or(mv[CurrentX-1][CurrentY+1]≠4))
and(mv[CurrentX-2][CurrentY+1]=(1 or 2)
or(mv[CurrentX-2][CurrentY+1]≠4))
and(mv[CurrentX-3][CurrentY+1]=(0 or 1 or 2)
or(mv[CurrentX-3][CurrentY+1]≠4))
and(mv[CurrentX][CurrentY+2]≠4)
and(mv[CurrentX][CurrentY+3]≠4)
and(mv[CurrentX][CurrentY-1]≠1)
and(mv[CurrentX][CurrentY-1]≠4))
or...
or((mv[CurrentX][CurrentY+1]=(0 or 1 or 2)
or(mv[CurrentX][CurrentY+1]≠4))
and(mv[CurrentX-1][CurrentY+1]=(0 or 1 or 2)
or(mv[CurrentX-1][CurrentY+1]≠4))
and(mv[CurrentX-2][CurrentY+1]=(0 or 1 or 2)
or(mv[CurrentX-2][CurrentY+1]≠4))
and(mv[CurrentX-3][CurrentY+1]=(1 or 2)
or(mv[CurrentX-3][CurrentY+1]≠4))
and(mv[CurrentX][CurrentY+2]≠4)
and(mv[CurrentX][CurrentY+3]≠4)
and(mv[CurrentX][CurrentY-1]≠1)
and(mv[CurrentX][CurrentY-1]≠4)))
```

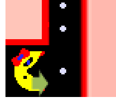

Figura 26: Restricción 94.

- Restricción 96:
  Esta restricción aplica si Ms. PacMan tiene en el primer casillero hacia la derecha píldoras, píldoras de poder o no píldoras, y los 3 casilleros abajo de este último píldoras, píldoras de poder o no píldoras, puede avanzar a la derecha. Al igual que la restricción anterior se evalúan distintas posibilidades de espacios vacíos, pero ahora hacia abajo. Esto se ve reflejado en la figura 27.

```
c96:if(((mv[CurrentX][CurrentY+1]=(0 or 1 or 2)
or(mv[CurrentX][CurrentY+1]≠4))
and(mv[CurrentX+1][CurrentY+1]=(1 or 2)
or(mv[CurrentX+1][CurrentY+1]≠4))
and(mv[CurrentX+2][CurrentY+1]=(1 or 2)
or(mv[CurrentX+2][CurrentY+1]≠4))
and(mv[CurrentX+3][CurrentY+1]=(0 or 1 or 2)
or(mv[CurrentX+3][CurrentY+1]≠4))
and(mv[CurrentX][CurrentY+2]≠4)
and(mv[CurrentX][CurrentY+3]≠4)
and(mv[CurrentX][CurrentY-1]≠1)
and(mv[CurrentX][CurrentY-1]≠4))
or...
or((mv[CurrentX][CurrentY+1]=(0 or 1 or 2)
or(mv[CurrentX][CurrentY+1]≠4))
and(mv[CurrentX+1][CurrentY+1]=(0 or 1 or 2)
or(mv[CurrentX+1][CurrentY+1]≠4))
and(mv[CurrentX+2][CurrentY+1]=(0 or 1 or 2)
or(mv[CurrentX+2][CurrentY+1]≠4))
and(mv[CurrentX+3][CurrentY+1]=(1 or 2)
or(mv[CurrentX+3][CurrentY+1]≠4))
and(mv[CurrentX][CurrentY+2]≠4)
and(mv[CurrentX][CurrentY+3]≠4)
```

```
and(mv[CurrentX][CurrentY-1]≠1)
and(mv[CurrentX][CurrentY-1]≠4)))
```

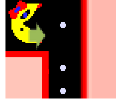

Figura 27: Restricción 96.

- Restricción 112:
  Esta restricción aplica si Ms. PacMan tiene no píldoras los 3 casilleros siguientes a la derecha, y en el tercer casillero a su derecha tiene píldora hacia arriba o abajo de éste, entonces debe moverse a la derecha. Esto se ve reflejado en la figura 28.

```
c112:if((mv[CurrentX][CurrentY+1]=0)
and(mv[CurrentX][CurrentY+2]=0)
and(mv[CurrentX][CurrentY+3]=0)
and(mv[CurrentX-1][CurrentY]≠1)
and(mv[CurrentX+1][CurrentY]≠1)
and(mv[CurrentX-1][CurrentY+1]≠1)
and(mv[CurrentX+1][CurrentY+1]≠1)
and(mv[CurrentX-1][CurrentY+2]≠1)
and(mv[CurrentX+1][CurrentY+2]≠1)
and((mv[CurrentX-1][CurrentY+3]=1)
or(mv[CurrentX+1][CurrentY+3]=1)))
```

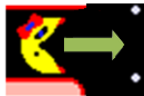

Figura 28: Restricción 112.

- Restricciones de unión
  restricciones que unen el conjunto de restricciones simples de las 4 direcciones, y restricciones que unen el conjunto de restricciones de prioridad píldora para todas las direcciones, estos casos se presentan a continuación:

- Restricción 37 (unión de restricciones simples)
  Si se cumple una de las restricciones simples de moverse a la derecha, retorna verdadero.

  ```
  c37:if(c1 or c2 or c3 or c4 or c5 or c6 or c7
  or c30 or c33 or c103 or c105)
  ```

- Restricción 90 (unión de restricciones con prioridad píldora)
  Si se cumple una de las restricciones simples de moverse a la derecha, retorna verdadero.

  ```
  c90:if(c42 or c43 or c44 or c45 or c46 or c47
  or c48 or c70 or c75 or c78 or c82 or c86
  or c94 or c96 or c103 or c105 or c112)
  ```

- Restricciones de toma de decisión de movimiento
  Estas restricciones son árboles de decisiones que incluyen las restricciones de unión y que van priorizando los movimientos hacia los caminos con píldora por sobre los caminos vacíos, además de priorizar el movimiento hacia la dirección del movimiento anterior. De no poder continuar en esa dirección se debe evaluar las demás direcciones para hacer el siguiente movimiento, y si no encuentra ningún camino con píldoras comienza a evaluar las restricciones de unión normales para evaluar movimiento. Para tener una prioridad de que dirección debe evaluarse, si los casos anteriores no cumplen con importancia mayor existen 3 grupos de restricciones de toma de decisión de movimiento, las en sentido reloj, contrarreloj y especial. Cada uno de estos tipos de toma de decisión posee 4 casos, uno para cada movimiento anterior. Las restricciones en sentido reloj siguen la prioridad de dirección de movimiento como dice su nombre, entregando importancia al movimiento anteriormente realizado, pero con la salvedad de dejar en última instancia la dirección contraria a la realizada anteriormente por Ms. PacMan, es decir si el movimiento anterior fue hacia arriba, el orden de prioridades de movimiento sería arriba, derecha, izquierda y abajo, evaluando solo las restricciones de unión de las restricciones de prioridad píldora y de no cumplirse sigue el árbol de decisión repitiendo las mismas direcciones arriba, derecha, izquierda y abajo, pero ahora evaluando las restricciones de unión

simples. Las en sentido contrarreloj hacen lo mismo que la anterior pero en sentido contrario y las restricciones de toma de decisión de movimiento especial, no tienen un patrón definido de orden. Ms. PacMan sólo se mueve en sentido reloj, y puede usar los otros 2 tipos cuando entra en estado de Loop de movimiento.

- Restricción reloj2
  Esta restricción aplica si el movimiento anterior fue derecha, entonces evalúa si tiene píldoras en esa dirección, de no encontrar alguna o de no ser posible evalúa moverse hacia abajo, arriba o izquierda siguiendo el orden. Si no hay píldoras cercanas en ninguna dirección evalúa el mismo orden, derecha, abajo, arriba e izquierda pero con caminos vacíos para moverse. Esto se ve reflejado en la figura 29.

```
reloj2:if((90)then(Up=0 and Down=0 and Right=1
and Left=0 and Loop=0)
else(if(93)then(Up=0 and Down=1 and Right=0
and Left=0 and Loop=0)
else(if(92)then(Up=1 and Down=0 and Right=0
and Left=0 and Loop=0)
else(if(91)then(Up=0 and Down=0 and Right=0
and Left=1 and Loop=0)
else(if(37)then(Up=0 and Down=0 and Right=1
and Left=0 and Loop=1)
else(if(40)then(Up=0 and Down=1 and Right=0
and Left=0 and Loop=1)
else(if(39)then(Up=1 and Down=0 and Right=0
and Left=0 and Loop=1)
else(if(38)then(Up=0 and Down=0 and Right=0
and Left=1 and Loop=1))))))))
```

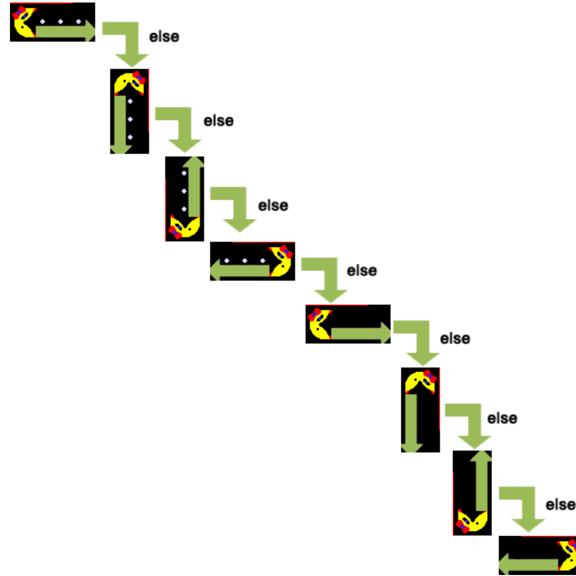

Figura 29: Restricción reloj2.

- Restricción contra2
  Esta restricción aplica si el movimiento anterior fue derecha, entonces evalúa si tiene píldoras en esa dirección, de no encontrar alguna o de no ser posible evalúa moverse hacia arriba, abajo o izquierda siguiendo el orden. Si no hay píldoras cercanas en ninguna dirección evalúa el mismo orden, derecha, arriba, abajo e izquierda pero con caminos vacíos para moverse. Esto se ve reflejado en la figura 30.

```
contra2:if((90)then(Up=0 and Down=0 and Right=1
and Left=0 and Loop=0)
else(if(92)then(Up=1 and Down=0 and Right=0
and Left=0 and Loop=0)
else(if(93)then(Up=0 and Down=1 and Right=0
and Left=0 and Loop=0)
else(if(91)then(Up=0 and Down=0 and Right=0
and Left=1 and Loop=0)
else(if(37)then(Up=0 and Down=0 and Right=1
and Left=0 and Loop=1)
else(if(39)then(Up=1 and Down=0 and Right=0
```

```
and Left=0 and Loop=1)
else(if(40)then(Up=0 and Down=1 and Right=0
and Left=0 and Loop=1)
else(if(38)then(Up=0 and Down=0 and Right=0
and Left=1 and Loop=1))))))))
```

Figura 30: Restricción contra2.

- Restricción especial3
  Esta restricción aplica si el movimiento anterior fue derecha, entonces evalúa si tiene píldoras en esa dirección, de no encontrar alguna o de no ser posible evalúa moverse hacia la izquierda, arriba o abajo siguiendo el orden. Si no hay píldoras cercanas en ninguna dirección evalúa el mismo orden, derecha, izquierda, arriba y abajo pero con caminos vacíos para moverse. Esto se ve reflejado en la figura 31.

```
especial3:if((90)then(Up=0 and Down=0 and Right=1
and Left=0 and Loop=0)
else(if(91)then(Up=0 and Down=0 and Right=0
and Left=1 and Loop=0)
else(if(92)then(Up=1 and Down=0 and Right=0
```

```
and Left=0 and Loop=0)
else(if(93)then(Up=0 and Down=1 and Right=0
and Left=0 and Loop=0)
else(if(37)then(Up=0 and Down=0 and Right=1
and Left=0 and Loop=1)
else(if(38)then(Up=0 and Down=0 and Right=0
and Left=1 and Loop=1)
else(if(39)then(Up=1 and Down=0 and Right=0
and Left=0 and Loop=1)
else(if(40)then(Up=0 and Down=1 and Right=0
and Left=0 and Loop=1))))))))
```

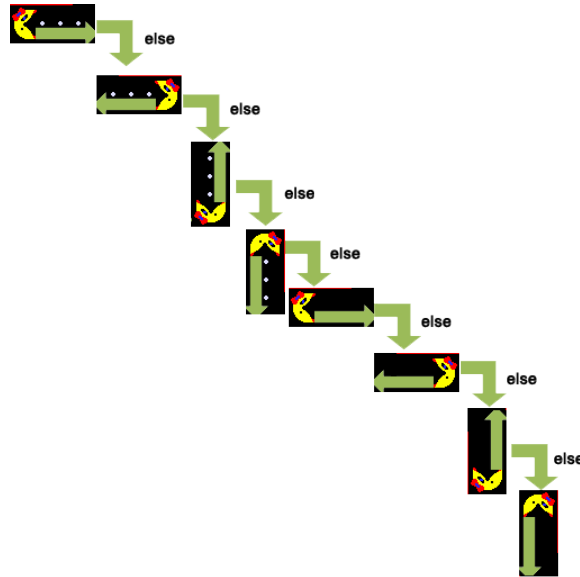

Figura 31: Restricción especial3.

- Loop de movimiento
  Los loop de movimiento son instancias en que Ms. PacMan empieza a seguir un trazo de camino repetitivamente, esto ocurre generalmente al no tener ninguna píldora cercana a su alrededor dentro de todo este trayecto (Véase figura 32), es por ello que se creó un constructor de búsqueda que anida los movimientos reloj, contrarreloj y especial. Este constructor se activa al alcanzar la variable cont un límite definido, estos son 180 en el caso que que-

den más de 50 píldoras en el laberinto o 80 en el caso contrario. Al alcanzar dicho límite la variable loop obtiene valor 1 activándose el cambio de sentido. El laberinto se dividió en 8 hemisferios, que son superior izquierdo, superior derecho, inferior izquierdo e inferior derecho, arriba, abajo, derecho e izquierdo (estos últimos 4 en el caso de que la píldora este en la misma columna o fila que Ms. PacMan). Se hace una búsqueda por todo el tamaño del tablero y al encontrar un casillero con valor 1 (Pills) para de buscar y guarda las coordenadas, y empieza a buscar en que hemisferio esta situada la píldora. Al encontrar el objetivo Ms. PacMan cambiara sus prioridades de movimiento dependiendo de su posición actual y de que hemisferio sea el que posee la píldora encontrada, por ejemplo si la píldora esta situada en el hemisferio superior izquierdo y la posición actual de Ms. PacMan es el hemisferio inferior derecho, sus prioridades de movimiento serán arriba e izquierda. Ms.PacMan finaliza este estado de loop al encontrarse con un fantasma o al comer una píldora.

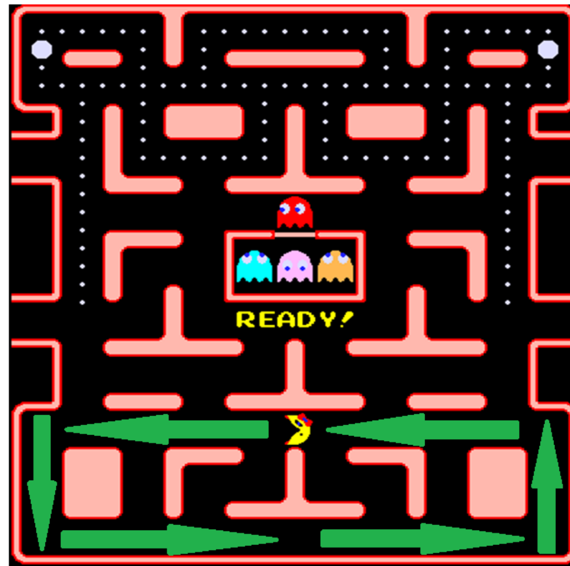

Figura 32: Loop de Movimiento.

6. Plataforma de implementación de Ms PacMan

En 1979 nace uno de los videojuegos más populares de la historia, llamado PacMan, creado por Toru Iwatani para la compañía Namco, el cual rápidamente logró gran éxito en todo el mundo. En este juego el jugador controla a PacMan dentro de un laberinto. Dentro de este laberinto están distribuídas las píldoras que PacMan debe comer para poder completar el nivel. Para agregarle dificultad dentro del laberinto existen 4 fantasmas que trataran de impedir que PacMan logre comer todas las píldoras [2].

En 1981 Ms. PacMan es desarrollado como sucesor del primer juego. Ambos juegos comparten el mismo funcionamiento, teniendo como grandes diferencias el que el jugador ahora controla a Ms. PacMan y el comportamiento de los fantasmas es distinto. En la versión original el comportamiento de los fantasmas es determinista, lo que quiere decir que si un jugador repite movimientos durante varios juegos, el movimiento de los fantasmas se repetirá también, lo que hacía posible determinar agentes para aprender rutas óptimas. En Ms. PacMan los fantasmas poseen un componente pseudo aleatorio, que previene su aprendizaje de rutas óptimas, lo cual incrementa la dificultad de desarrollar un agente. Se han desarrollado un gran número de versiones del juego para distintas plataformas hasta la fecha. La plataforma de Ms. PacMan escogida para trabajar en este proyecto es una interfaz de código abierto desarrollada en JAVA por **meatfighter.com** [8] denominada Ms.Pac-Man 2010 (figura 33). Esta interfaz está inspirada por el trabajo de las distintas empresas desarrolladoras de versiones del juego, que son Namco, Midway, General Computer Corporation, Nintendo y Capcom. A pesar de esto la versión utilizada no es una copia exacta del juego original, pero contiene la estructura original del juego.

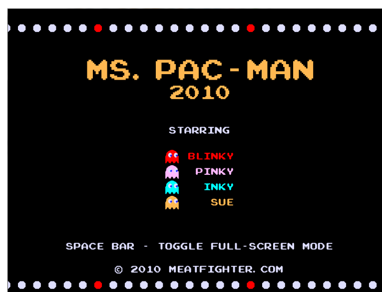

Figura 33: Interfaz Ms. Pac-Man 2010.

7. Estructuras de Unión

Para lograr que Ms. PacMan pueda moverse de forma autónoma por el laberinto del juego, se necesitaron realizar una serie de estructuras (clases en java) que complementaran la comunicación entre el modelo de CP y la interfaz, estas son clase Solver y el Traductor, que son las encargadas de realizar el manejo de datos y la transformación de ellos a una solución de CSP; junto a estos dos se necesito utilizar clases nativas del juego como la clase Thing y HumanInput. Estas serán explicadas en orden de uso para que sea más entendible el comportamiento de estas, acompañado de un diagrama de clases (Figura 34) al final de la descripción de estas cuatro clases.

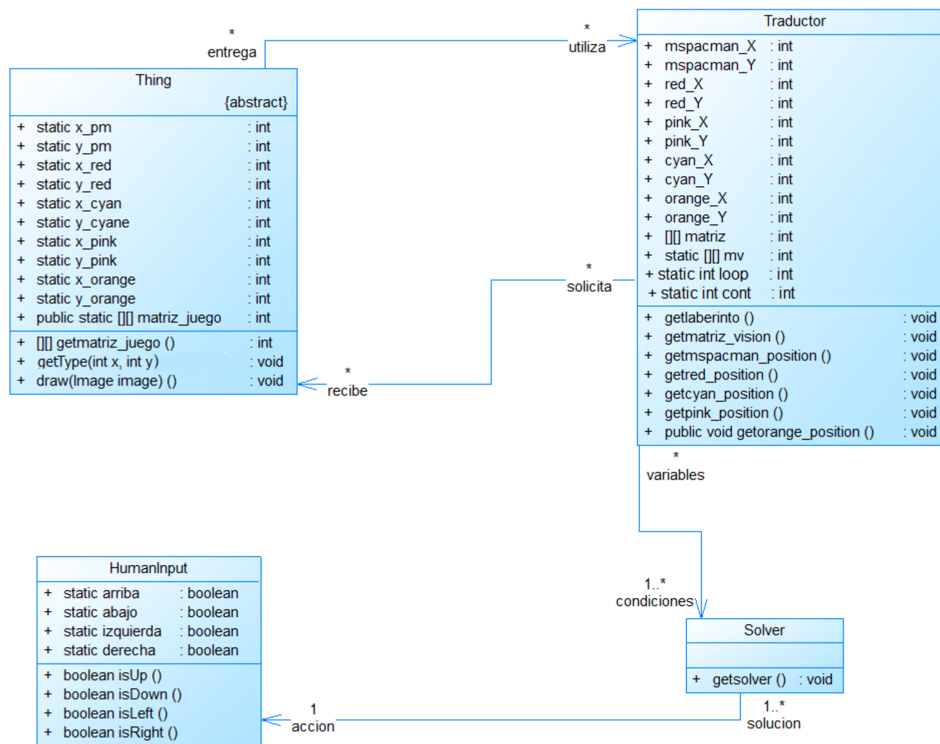

Figura 34: Diagrama de Clases.

7.1. Thing.java

Esta clase es usada para crear el ambiente de juego, entre estos tamaños y componentes, así como las posiciones de Ms. PacMan, los fantasmas, frutas, píldoras, murallas etc.; también encargada de dar las formas a la interfaz como los dibujos, distribuciones de los elementos. De esta clase se obtienen los valores para poder inicializar el traductor (explicado a continuación), gracias a que acá se forma la matriz con la que trabajaremos y se inicializan todas las posiciones de los personajes del juego.

7.2. Traductor.java

Esta es la estructura más importante entre la interfaz del juego y el Solver realizado en choco (Solver) ya que este se encarga de transformar los datos obtenidos del Thing a los que se usaran en la evaluación de las decisiones en el siguen paso por el Solver. El traductor crea una copia de la matriz original del juego para saber cuáles son los valores con los que se trabajarán, así mismo obtiene los valores de las posiciones iníciales de Ms. PacMan y gracias a esto se puede construir la "matriz de visión" que se utiliza para distinguir entre que opciones se debe mover el personaje; obtenemos también las posiciones de los fantasmas para poder reconocerlos y tenerlos de manera visible dentro de la matriz con la que se está evaluando. Por último obtendremos el cambio de desición de movimiento de Ms. PacMan de esta clase si es que la variable `loop` cambia a 1 al cumplirse el valor pedido al contador de camino sin píldora recorrido.

7.3. Solver.java

Podríamos decir de alguna manera que este es el cerebro de nuestro proyecto; el Solver es donde se encuentran todas las condiciones y restricciones en Choco que conforman nuestro modelo de CSP, ya sean si existen píldoras, fantasmas, murallas o caminos libres con el cual podría Ms. PacMan encontrarse, el cual nos permite distinguir y definir cuál será el comportamiento que debe cumplir Ms. PacMan para que pueda cumplir con su objetivo y este así mismo es el encargado de dar la señal del movimiento siguiente que se debe realizar, enviando la decisión elegida a HumanInput.

7.4. HumanInput.java

Encargado de realizar la acción enviada por el Solver. En esta clase se encuentran los procedimientos que interactúan con el movimiento en sí de

Ms. PacMan, entre estos procedimientos están derecha, izquierda, arriba, abajo, enter, pausa, entre otros; los cuales en un inicio sólo eran para el ingreso de valores por teclado en esta instancia se han ocupado para recibir los valores enviados por el Solver.

8. Codificación del problema en Choco

8.1. Declaraciones en Choco

A continuación se explican las declaraciones utilizadas en el código en choco:

- `CPModel m= new CPModel();`
  Creación del modelo en Choco.

- `IntegerVariable X = Choco.makeIntVar("X",Y,Z);`
  Creación de una variable entera para el modelo, donde X es el nombre de la variable e Y y Z sus dominios.

- `m.addVariables(X);`
  Añade la variable X al modelo.

- `IntegerVariable X = Choco.constant(Y);`
  Creación de la constante X de valor Y.

- `Choco.or(X,Y);`
  Establece que uno o ambos argumentos, X e Y se cumplan.

- `Choco.and(X,Y);`
  Establece que los dos argumentos, X e Y se cumplan.

- `Choco.eq(X,Y);`
  Establece que los dos argumentos, X e Y sean iguales.

- `Choco.neq(X,Y);`
  Establece que los dos argumentos, X e Y sean distintos.

- `Choco.ifThenElse(X,Y,Z);`
  Si satisface X entonces retorna Y, de no ser así retorna Z.

- `Choco.ifOnlyIf(X,Y);`
  Si satisface X entonces retorna Y.

- `m.addConstraint(X);`
  Añade la restricción X al modelo.

- `Solver s = new CPSolver();`
  Creación del Solver en Choco.

- `s.read(m);`
  El Solver s hace la Lectura del modelo m en Choco.
- `s.solve();`
  El Solver s resuelve el problema.

8.2. Modelo

Para esta instancia se ha implementado el Solver en Choco, mediante las estructuras de unión con el código del juego. La declaración de nuestro modelo escrito en Choco, sus variables, constantes y un extracto de las restricciones más importantes sigue a continuación.

- Variables
  - `IntegerVariable Left = Choco.makeIntVar("Left",0,1);`
  - `IntegerVariable Right = Choco.makeIntVar("Right",0,1);`
  - `IntegerVariable Up = Choco.makeIntVar("Up",0,1);`
  - `IntegerVariable Down = Choco.makeIntVar("Down",0,1);`
  - `IntegerVariable Loop = Choco.makeIntVar("Down",0,1);`
- Constantes
  - `IntegerVariable NP = Choco.constant(0);`
  - `IntegerVariable Pills = Choco.constant(1);`
  - `IntegerVariable PP = Choco.constant(2);`
  - `IntegerVariable Wall = Choco.constant(3);`
  - `IntegerVariable Ghost = Choco.constant(4);`
  - `IntegerConstantVariable[][] mv;`
  - `mv=Choco.constantArray(Traductor.mv);`
  - `int CurrentX=Traductor.mspacman_Y+3;`
  - `int CurrentY=Traductor.mspacman_X+3;`
  - `Traductor.mov_ant;`
  - `Traductor.cont;`
  - `Traductor.loop;`

- Restricciones
  - Restricción 1:
    Esta restricción aplica si Ms. PacMan tiene píldoras, píldoras de poder o no píldoras en los 3 casilleros siguientes a la derecha, puede avanzar a la derecha. Esto se ve reflejado en la figura 35.

    Pseudo Código:

```
c1:if((mv[CurrentX][CurrentY+1]=(0 or 1 or 2)
or(mv[CurrentX][CurrentY+1]≠4))
and(mv[CurrentX][CurrentY+2]=(0 or 1 or 2)
or(mv[CurrentX][CurrentY+2]≠4))
and(mv[CurrentX][CurrentY+3]=(0 or 1 or 2)
or(mv[CurrentX][CurrentY+3]≠4)))
```

    Código en Choco:

```
c1=Choco.and(Choco.and(Choco.or((
Choco.eq(mv[CurrentX][CurrentY+1],Pills)),
(Choco.eq(mv[CurrentX][CurrentY+1],NP)),
(Choco.eq(mv[CurrentX][CurrentY+1],PP))),
(Choco.neq(mv[CurrentX][CurrentY+1],Ghost))),
Choco.and(Choco.or((Choco.eq(mv[CurrentX][CurrentY+2],Pills)),
(Choco.eq(mv[CurrentX][CurrentY+2],NP)),
(Choco.eq(mv[CurrentX][CurrentY+2],PP))),
(Choco.neq(mv[CurrentX][CurrentY+1],Ghost))),
Choco.and(Choco.or((Choco.eq(mv[CurrentX][CurrentY+3],Pills)),
(Choco.eq(mv[CurrentX][CurrentY+3],NP)),
(Choco.eq(mv[CurrentX][CurrentY+3],PP))),
(Choco.neq(mv[CurrentX][CurrentY+3],Ghost))));
```

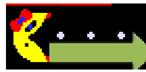

Figura 35: Restricción 1.

- Restricción 2:
  Esta restricción aplica si Ms. PacMan tiene píldoras, píldoras de poder o no píldoras situadas en los 3 casilleros siguientes en forma de “L” hacia la derecha y abajo, puede moverse a la derecha. Esto se ve reflejado en la figura 36.

  Pseudo Código:

```
c2:if((mv[CurrentX][CurrentY+1]=(1 or 2 or 3)
or(mv[CurrentX][CurrentY+1]≠4))
and(mv[CurrentX][CurrentY+2]=(0 or 1 or 2)
or(mv[CurrentX][CurrentY+2]≠4))
and(mv[CurrentX+1][CurrentY+2]=(0 or 1 or 2)
or (mv[CurrentX+1][CurrentY+2]≠4)))
```

  Código en Choco:

```
c2=Choco.and(Choco.and(Choco.or((
Choco.eq(mv[CurrentX][CurrentY+1],Pills)),
(Choco.eq(mv[CurrentX][CurrentY+1],NP)),
(Choco.eq(mv[CurrentX][CurrentY+1],PP))),
(Choco.neq(mv[CurrentX][CurrentY+1],Ghost))),
Choco.and(Choco.or((Choco.eq(mv[CurrentX][CurrentY+2],Pills)),
(Choco.eq(mv[CurrentX][CurrentY+2],NP)),
(Choco.eq(mv[CurrentX][CurrentY+2],PP))),
(Choco.neq(mv[CurrentX][CurrentY+2],Ghost))),
Choco.and(Choco.or((Choco.eq(mv[CurrentX+1][CurrentY+2],Pills)),
(Choco.eq(mv[CurrentX+1][CurrentY+2],NP)),
(Choco.eq(mv[CurrentX+1][CurrentY+2],PP))),
(Choco.neq(mv[CurrentX+1][CurrentY+2],Ghost))),
(Choco.neq(mv[CurrentX][CurrentY+3],Ghost)));
```

Figura 36: Restricción 2.

- Restricción 3:
  Esta restricción aplica si Ms. PacMan tiene píldoras, píldoras de poder o no píldoras situadas en los 3 casilleros siguientes en forma de “L” hacia la derecha y arriba, puede moverse a la derecha. Esto se ve reflejado en la figura 37.

  Pseudo Código:

```
c3:if((mv[CurrentX][CurrentY+1]=(0 or 1 or 2)
or(mv[CurrentX][CurrentY+1]≠4))
and(mv[CurrentX][CurrentY+2]=(0 or 1 or 2)
or(mv[CurrentX][CurrentY+2]≠4))
and(mv[CurrentX-1][CurrentY+2]=(0 or 1 or 2)
or(mv[CurrentX-1][CurrentY+2]≠4)))
```

  Código en Choco:

```
c3=Choco.and(Choco.and(Choco.or((
Choco.eq(mv[CurrentX][CurrentY+1],Pills)),
(Choco.eq(mv[CurrentX][CurrentY+1],NP)),
(Choco.eq(mv[CurrentX][CurrentY+1],PP))),
(Choco.neq(mv[CurrentX][CurrentY+1],Ghost))),
Choco.and(Choco.or((Choco.eq(mv[CurrentX][CurrentY+2],Pills)),
(Choco.eq(mv[CurrentX][CurrentY+2],NP)),
(Choco.eq(mv[CurrentX][CurrentY+2],PP))),
(Choco.neq(mv[CurrentX][CurrentY+2],Ghost))),
Choco.and(Choco.or((Choco.eq(mv[CurrentX-1][CurrentY+2],Pills)),
(Choco.eq(mv[CurrentX-1][CurrentY+2],NP)),
(Choco.eq(mv[CurrentX-1][CurrentY+2],PP))),
(Choco.neq(mv[CurrentX-1][CurrentY+2],Ghost))),
(Choco.neq(mv[CurrentX][CurrentY+3],Ghost)));
```

Figura 37: Restricción 3.

- Restricción 4:
  Esta restricción aplica si Ms. PacMan tiene píldoras, píldoras de poder o no píldoras en las 2 posiciones siguientes y muro en la tercera posición a la derecha, puede moverse hacia esa dirección. Esto se ve reflejado en la figura 38.

  Pseudo Código:

```
c4:if((mv[CurrentX][CurrentY+1]=(0 or 1 or 2)
or(mv[CurrentX][CurrentY+1]≠4))
and(mv[CurrentX][CurrentY+2]=(0 or 1 or 2)
or(mv[CurrentX][CurrentY+2]≠4))
and(mv[CurrentX][CurrentY+3]=3)
and(mv[CurrentX-1][CurrentY+2]≠4
or(mv[CurrentX+1][CurrentY+2]≠4)))
```

  Código en Choco:

```
c4=Choco.and(Choco.and(Choco.or((
Choco.eq(mv[CurrentX][CurrentY+1],Pills)),
(Choco.eq(mv[CurrentX][CurrentY+1],NP)),
(Choco.eq(mv[CurrentX][CurrentY+1],PP))),
(Choco.neq(mv[CurrentX][CurrentY+1],Ghost))),
Choco.and(Choco.or((Choco.eq(mv[CurrentX][CurrentY+2],Pills)),
(Choco.eq(mv[CurrentX][CurrentY+2],NP)),
(Choco.eq(mv[CurrentX][CurrentY+2],PP))),
(Choco.neq(mv[CurrentX][CurrentY+2],Ghost))),
(Choco.eq(mv[CurrentX][CurrentY+3],Wall)),
Choco.or((Choco.neq(mv[CurrentX-1][CurrentY+2],Ghost)),
(Choco.neq(mv[CurrentX+1][CurrentY+2],Ghost))));
```

Figura 38: Restricción 4.

- Restricción 5:
  Esta restricción aplica si Ms. PacMan tiene un casillero con píldora, píldora de poder o no píldora y los 2 siguientes con muro hacia la derecha, puede avanzar hacia esa dirección. Esto se ve reflejado en la figura 39.

  Pseudo Código:

  ```
  c5:if(mv[CurrentX][CurrentY+1]=(0 or 1 or 2)
  or(mv[CurrentX][CurrentY+1]≠4))
  and(mv[CurrentX][CurrentY+2]=3)
  and(mv[CurrentX][CurrentY+3]=3)
  ```

  Código en Choco:

  ```
  c5=Choco.and(Choco.and(Choco.or((
  Choco.eq(mv[CurrentX][CurrentY+1],Pills)),
  (Choco.eq(mv[CurrentX][CurrentY+1],NP)),
  (Choco.eq(mv[CurrentX][CurrentY+1],PP))),
  (Choco.neq(mv[CurrentX][CurrentY+1],Ghost))),
  (Choco.eq(mv[CurrentX][CurrentY+2],Wall)),
  (Choco.eq(mv[CurrentX][CurrentY+3],Wall)));
  ```

Figura 39: Restricción 5.

- Restricción 6:
  Esta restricción aplica si Ms. PacMan tiene píldoras, píldoras de poder o no píldoras y fantasma situados en los 3 casilleros siguientes en forma de "L"hacia la izquierda y abajo, no puede avanzar y debe moverse en la dirección contraria. Esto se ve reflejado en la figura 40.

  Pseudo Código:

  ```
  c6:if(mv[CurrentX][CurrentY-1]=(0 or 1 or 2)
  or(mv[CurrentX][CurrentY-1]≠4)
  ```

```
and(mv[CurrentX][CurrentY-2]=(0 or 1 or 2)
or(mv[CurrentX][CurrentY-2]≠4))
and(mv[CurrentX+1][CurrentY-2=4)))
```

Código en Choco:

```
c6=Choco.and(Choco.and(Choco.or((
Choco.eq(mv[CurrentX][CurrentY-1],Pills)),
(Choco.eq(mv[CurrentX][CurrentY-1],NP)),
(Choco.eq(mv[CurrentX][CurrentY-1],PP))),
(Choco.neq(mv[CurrentX][CurrentY-1],Ghost))),
Choco.and(Choco.or((Choco.eq(mv[CurrentX][CurrentY-2],Pills)),
(Choco.eq(mv[CurrentX][CurrentY-2],NP)),
(Choco.eq(mv[CurrentX][CurrentY-2],PP))),
(Choco.neq(mv[CurrentX][CurrentY-2],Ghost))),
(Choco.eq(mv[CurrentX+1][CurrentY-2],Ghost)),Choco.or(c1,c2,c3));
```

Figura 40: Restricción 6.

- Restricción reloj2
  Esta restricción aplica si el movimiento anterior fue derecha, entonces evalúa si tiene píldoras en esa dirección, de no encontrar alguna o de no ser posible evalúa moverse hacia abajo, arriba o izquierda siguiendo el orden. Si no hay píldoras cercanas en ninguna dirección evalúa el mismo orden, derecha, abajo, arriba e izquierda pero con caminos vacíos para moverse. Esto se ve reflejado en la figura 41.

  Pseudo Código:

```
reloj2:if((90)then(Up=0 and Down=0 and Right=1
and Left=0 and Loop=0)
else(if(93)then(Up=0 and Down=1 and Right=0
and Left=0 and Loop=0)
else(if(92)then(Up=1 and Down=0 and Right=0
```

```
and Left=0 and Loop=0)
else(if(91)then(Up=0 and Down=0 and Right=0
and Left=1 and Loop=0)
else(if(37)then(Up=0 and Down=0 and Right=1
and Left=0 and Loop=1)
else(if(40)then(Up=0 and Down=1 and Right=0
and Left=0 and Loop=1)
else(if(39)then(Up=1 and Down=0 and Right=0
and Left=0 and Loop=1)
else(if(38)then(Up=0 and Down=0 and Right=0
and Left=1 and Loop=1))))))))
```

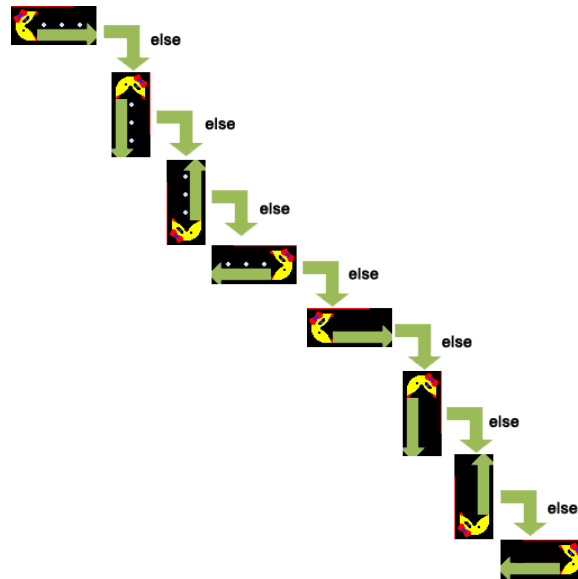

Figura 41: Restricción reloj2.

Código en Choco:

```
reloj2=Choco.ifThenElse(c90,Choco.and(Choco.eq(Down,0),
Choco.eq(Up,0),Choco.eq(Right,1),Choco.eq(Left,0),Choco.eq(Loop,0)),
Choco.ifThenElse(c93,Choco.and(Choco.eq(Down,1),Choco.eq(Up,0),
Choco.eq(Right,0),Choco.eq(Left,0),Choco.eq(Loop,0)),
Choco.ifThenElse(c92,Choco.and(Choco.eq(Down,0),Choco.eq(Up,1),
Choco.eq(Right,0),Choco.eq(Left,0),Choco.eq(Loop,0)),
```

```
Choco.ifThenElse(c91,Choco.and(Choco.eq(Down,0),Choco.eq(Up,0),
Choco.eq(Right,0),Choco.eq(Left,1),Choco.eq(Loop,0)),
Choco.ifThenElse(c37,Choco.and(Choco.eq(Down,0),Choco.eq(Up,0),
Choco.eq(Right,1),Choco.eq(Left,0),Choco.eq(Loop,1)),
Choco.ifThenElse(c40,Choco.and(Choco.eq(Down,1),Choco.eq(Up,0),
Choco.eq(Right,0),Choco.eq(Left,0),Choco.eq(Loop,1)),
Choco.ifThenElse(c39,Choco.and(Choco.eq(Down,0),Choco.eq(Up,1),
Choco.eq(Right,0),Choco.eq(Left,0),Choco.eq(Loop,1)),
Choco.ifOnlyIf(Choco.and(Choco.eq(Down,0),Choco.eq(Up,0),
Choco.eq(Right,0),Choco.eq(Left,1),Choco.eq(Loop,1)),c38))))))));
```

9. Pruebas y Resultados

En la fase más inicial del desarrollo, paralelo mientras se resolvía el problema de obtener los datos de la GUI y unir esta al Solver, se analizaron y crearon las restricciones y condiciones que conformarían el primer y básico conjunto del Solver (mover arriba, abajo, izquierda, derecha) evaluando solo el casillero contiguo a la posición del Ms. PacMan. Cada una de estas se probaron y testearon su rendimiento en una matriz de 7x7 inicializada con valores de prueba dentro del código del mismo Solver, para comprobar si las restricciones daban un resultado esperado al evaluar las variables en la matriz de prueba y que entre estas no existieran choques.

Cuando las restricciones iniciales fueron testeadas y resultaban satisfactorias para las pruebas, empezó el proceso de creación de restricciones más complejas evaluando no solo el casillero contiguo a Ms. PacMan, sino dos o tres casilleros más lejanos. También creando movimientos compuestos, por ejemplo arriba + izquierda, derecha + abajo, derecha + arriba, etc. Finalmente durante el período de desarrollo del Solver se llegó al universo de restricciones y condiciones mencionadas anteriormente y los resultados de las pruebas de ejecución (GUI + Solver) se pueden apreciar en el siguiente explicación y resumen de pruebas.

Se ha comparado los resultados de las pruebas del Solver con la competencia existente sobre este problema; las reglas de esta competencia se explican en el siguiente párrafo: El ganador será el programa que se puede lograr la puntuación más alta, dado el número permitido de intentos de jugar el juego. Los controladores presentados se llevará a cabo 10 veces cada uno antes de la final. Los finalistas serán los cinco controladores con los mejores de mejores puntuaciones en las carreras. Durante la final, cada controlador se llevará a cabo otras 3 veces. El controlador que gana es el que tiene la puntuación más alta en todas las 13 carreras (las 10 de las series más 3 de la de la final) [6].

Los resultados y ganadores de la competencia del 2011 [7] bajo las reglas anteriores se muestran en la figura 42.

Entry / Author(s)	Affiliation	High Score
Nozomu Ikehata and Takeshi Ito	The University of Electro-Communications, Tokyo	36,280
ICE Pambush 5 (Ruck) Takeru Miyama, Asami Yamada, Yuki Okunishi, Takashi Ashida, and **Ruck** Thawonmas	Ritsumeikan University, Kyoto	27,240
Kuan-Wei Chen, Chu-Ming Wang, and **Tsung-Che** Chiang	National Taiwan Normal University, Taiwan	20,300
SungUk Jeong and **Kyung-Joong** Kim	Sejong University, Seoul	19,900
Bruce Tong	City University of Hong Kong	13,700

Figura 42: Resultados de la competencia 2011.

Las pruebas se basaron en un universo de 40 ejecuciones del código controlando el juego de Ms. PacMan como muestra el gráfico; comparándolo con los resultados de la competencia, los resultados obtenidos en 13 ejecuciones, el mayor puntaje obtenido sería 11560 alcanzando el nivel 3 de la plataforma. El puntaje promedio de las 40 ejecuciones es de 7428 y el puntaje máximo alcanzado es de 13210 dejando al Solver dentro de los 10 lugares. La figura 43 muestra el gráfico de los puntajes obtenidos y niveles alcanzados en las 40 ejecuciones y la figura 44 muestrala etapa donde a Ms. PacMan queda sin vidas.

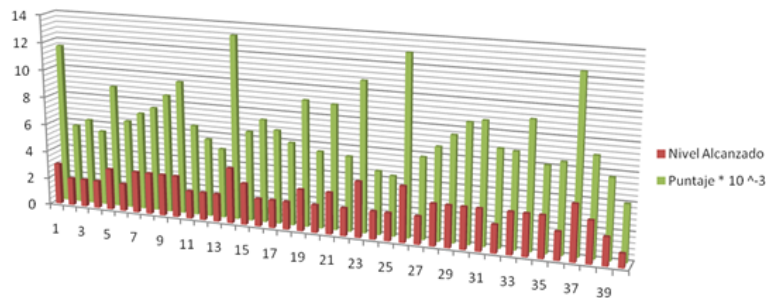

Figura 43: Resultados obtenidos por el Solver.

Figura 44: Gráfico Niveles Alcanzados.

10. Conclusión

En este proyecto, se ha introducido la programación con restricciones para resolver el Ms. PacMan Problem. Con variadas técnicas se ha tratado de resolver este problema, pero aún sin considerar la programación con restricciones. Los algoritmos de búsqueda, técnicas de consistencia y heurísticas contenidas en el Solver, han ayudado concretar los resultados esperados en un principio.

Se ha trabajado en la resolución de este problema formalizado como un CSP para la implementación en el Solver Choco, con los cuales se llegó a la resolución del problema. Actualmente, se han obtenido resultados prometedores, básicamente debido a la experimentación en la fase de desarrollo del modelo del problema para obtener finalmente una decisión de movimientos satisfactorios que pueden llegar a hacer que Ms. PacMan termine el juego. Para esto se han analizado y comparado los resultados obtenidos por el Solver Choco después de cada mejora o idea nueva para el Modelo desarrollado en el Solver.

Finalmente aunque la investigación del Ms. PacMan Problem utilizando programación con restricciones se muestra enfocada a este problema específico, pero puede ser aplicada nuevamente para en un futuro poder encontrar resultados óptimos u ser utilizado este tipo de programación en otros problemas de la misma índole.

Referencias

[1] Felip Manyà and Carla P. Gomes. Técnicas de resolución de problemas de satisfacción de restricciones. *Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial*, 7(19):169–180, 2003.

[2] Emilio Martín, Moisés Martínez, Gustavo Recio, and Yago Sáez. Pacmant: Optimization based on ant colonies applied to developing an agent for ms. pac-man. In *CIG*, pages 458–464, 2010.

[3] Problema Satisfacción de Restricciones. http://users.dsic.upv.es/ msalido/papers/capitulo.pdf, visited 04/2011.

[4] Ms. PacMan Wikipedia. http://es.wikipedia.org/wiki/Ms._Pac-Man, visited 04/2011.

[5] Agente software que utiliza inteligencia computacional para controlar a Ms. Pacman. http://code.google.com/p/pfc-gtg-mspacman/wiki/Introduccion, visited 04/2012.

[6] Ms. PacMan Competition. http://cswww.essex.ac.uk/staff/sml/pacman/PacManContest.html, visited 04/2012.

[7] Resultados competencia 2011. http://cswww.essex.ac.uk/staff/sml/pacman/CIG2011Results.html, visited 04/2012.

[8] Interfaz Ms Pacman. http://meatfighter.com/mspacman2010/, visited 04/2011.

[9] Walsh Rossi, van Beek. Handbook of Constraint Programming, 1st Edition. 2006.

[10] Istvan Szita and András Lörincz. Learning to play using low-complexity rule-based policies: Illustrations through ms. pac-man. *J. Artif. Intell. Res. (JAIR)*, 30:659–684, 2007.

[11] Ruck Thawonmas and Takashi Ashida. Evolution strategy for optimizing parameters in ms pac-man controller ice pambush 3. In *CIG*, pages 235–240, 2010.