Universidad Autónoma de Yucatán
Facultad de matemáticas

Inteligencia Artificial
Proyecto final
MsPacMan

José Alexandro Uc Santos
LIC
alexandro_245@hotmail.com

Marcos Antonio Flores Echazarreta
LIC
Marcos_echazarret@hotmail.com

15 de diciembre de 2015

Resumen

En este documento hacemos una pequeña descripción del código que realizamos para la inteligencia artificial del agente Ms.PacMan mediante máquina de estado finito, Esta metodología es parte de los agentes reactivos. En este caso, la tarea que hay que resolver es el desarrollo de un controlador de software para el popular videojuego Pac - Man. Este tema es importante porque la inteligencia artificial ha cambiado la forma de hacer juegos de aplicación diferentes modelos de agentes autónomos.

1. Introduccion

La inteligencia artificial es acerca de hacer a los ordenadores capaces de realizar las tareas de pensamiento que un ser humano, es decir hacer a la computadora capaz de pensar y hacer cosas grandiosas, pero esto no es nada fácil de realizar y los investigadores cada día tienen un avance que nos acerca la meta. Un ejemplo del comportamiento de una inteligencia podría ser un programa de computadora con la capacidad de resolver problemas aritméticos. La tarea a realizar en este proyecto final en este es desarrollar una inteligencia artificial. En este reporte encontraras todo lo relacionado con el proyecto final que realizamos en el curso de inteligencia artificial, donde nosotros decidimos hacer la inteligencia artificial de Ms. PacMan donde nuestro agente jugara contra los fantasmas del juego, 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.

2. Investigacion

Ms.PacMan:

Ms. PacMan se trata de un videojuego lanzado en 1981 como una secuela no autorizada de PacMan. El juego es casi idéntico al original. El jugador tiene que ir comiendo píldoras mientras esquiva a los fantasmas. Las píldoras de mayor tamaño permiten comerse a los fantasmas y hacen que éstos huyan en la dirección contraria. A medida que se van completando niveles, se va incrementando la dificultad. El objetivo consiste en comer el mayor número de píldoras, fantasmas y frutas para obtener la mayor puntuación posible.

Ms. PacMan era originalmente una versión falsificada del PacMan llamada Crazy Otto, creada por programadores empleados del General Computes Corporation (GCC). Luego de que el juego se volviera popular, Midway y GCC iniciaron una batalla legal por la legalidad del juego. Pero como el juego fue acoplado a los contenidos de Namco, ambas compañías tuvieron que venderles los derechos a su rival, terminando el conflicto. De cualquier modo, Ms. PacMan fue la primera de una serie de secuelas no autorizadas que finalmente terminaron haciendo que la licencia pertenezca a Namco y Midway.

3. Desarrollo

Para poder realizar nuestro agente inteligente tuvimos que basarnos de un juego ya hecho(donde se notara que hay partes en ingles del codigo implementado en la IA pero no lo cambiamos por el hecho que nos basamos de eso ya que en el juego usamos las mismas variables ya declaradas) y tambien nos ayudamos de una página proporcionado por el profesor que imparte la clase donde nos dijo que en esa parte se encuentra todas las herramientas que necesitábamos para realizar nuestro agente inteligente, para esto empezamos por saber cuáles eran las percepciones del jugador.

La percepción que tenemos del entorno es la siguiente:

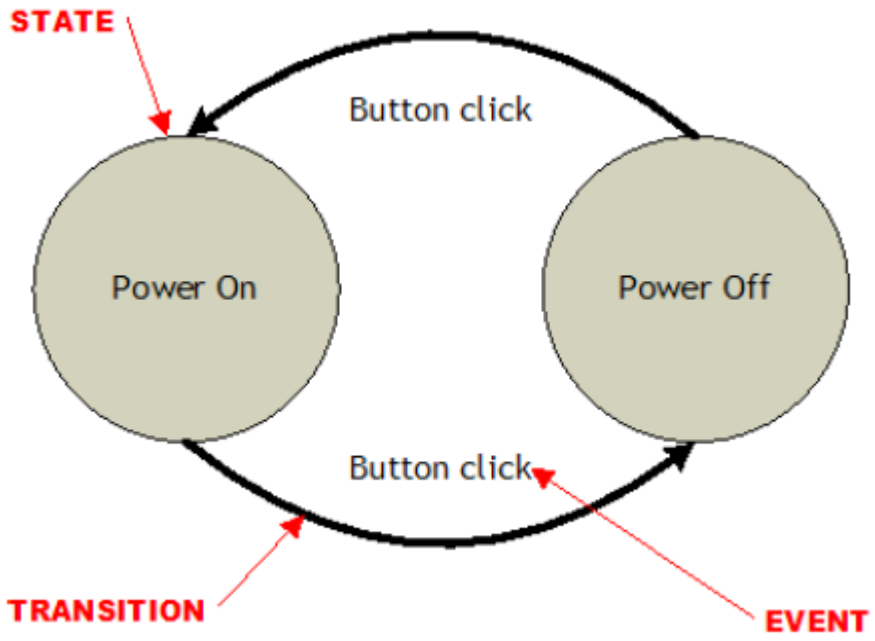
Imagen	*Percepción*	*Descripción*
	mapa del juego	Entorno por el que se mueve tanto Ms. PacMan como los fantasmas
	ms. PacMan	Personaje que maneja el jugador
	fantasmas	Fantasmas que intentarán comerse a Ms. PacMan
	fantasmas comestibles	Fantasmas que puede comerse Ms. PacMan para obtener puntuación adicional
	píldoras de poder	Píldoras que dan el poder a Ms. PacMan de hacer huir a los fantasmas
	túneles de escape	Si se pasa a través del túnel se aparece en el túnel simétrico del mapa

Acciones del juegador:

Si se tiene en cuenta estrictamente los controles a los que tiene acceso el jugador las acciones a realizar son las siguientes:
- No mover
- Mover hacia arriba
- Mover hacia abajo
- Mover hacia derecha
- Mover hacia izquierda

Nota: Si se produce un movimiento no permitido como mover a ms. Pacman hacia una pared el juego lo ignora.

Para la realización de nuestra inteligencia artificial usamos un código donde el juego ya estaba implementado y también las referencia de IEEE Computacional Inteligencie society para entender mejor como podría implementarlo: ya después de tanto investigar, decidimos usar el código donde había una inteligencia artificial hecha con el algoritmo de A* pero no era muy buena ya siempre tomaba el mismo camino, entonces lo que hicimos fue dejar todo el código del juego e implementar nuestra propia inteligencia artificial, donde usamos algunos tutoriales en YouTube donde realizaban un juego de PacMan en c++, pero nosotros nos basamos de esa para hacer un Ms.Pacman en Java con inteligencia artificial donde usábamos máquina de estados.
En una máquina de estado cada personaje tiene un estado. Por lo general, las acciones están asociadas con cada estado , siempre y cuando el personaje permanece en un estado que seguirá llevando a cabo la misma acción o comportamiento.
Los estados están conectados entre sí por las transiciones, en cada transición lleva de un estado a otro, el Estado de destino, y cada uno tiene un conjunto de condiciones asociadas.

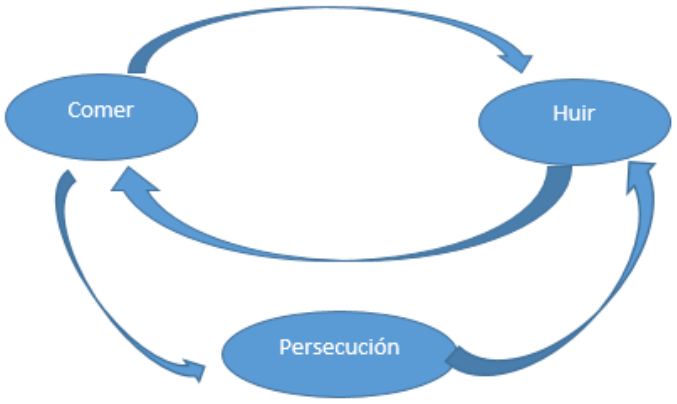


Estados de Ms.PacMan:

Huir
En este Estado de la máquina. Si Ms. PacMan ve un fantasma entonces huye Si no hay fantasmas cerca luego cambian para indicar que Coma.

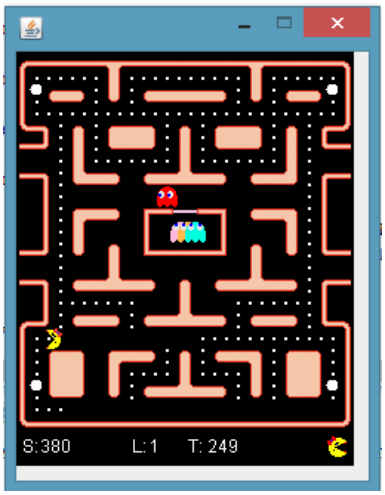
Persecución
En este estado Ms. PacMan, ya ha comido una de sus píldoras entonces perseguirá a los fantasmas para así comerlas y mejorar su score, y si no sigue comiendo.

Comer
Ms. PacMan va comiendo y si ve un fantasma entonces cambia al estado de Huir. Y si come una de sus píldoras cambia al estado de Persecución.



Método comer:

En este estado como ya habíamos explicado nuestro Ms. PacMan está comiendo y si detecta la amenaza de algún fantasma, entonces Huira hasta encontrar una píldora.

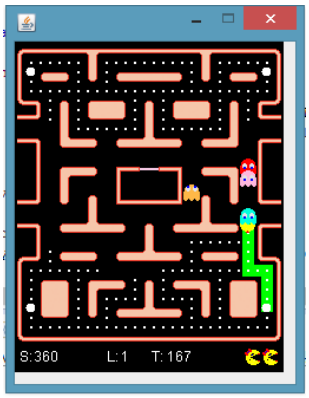


esta es una pequeña parte del codigo comer:

```
public MiPacMan() {
    root = new SubMaquinaEstado();

    SubMaquinaEstado normalState = new SubMaquinaEstado();
    root.addState(normalState);
    root.initialState = normalState;

    PacManState eatPillState = new PacManState(new CercaPillAvoidPowerAction(), "eat pill");
    normalState.addState(eatPillState);
    normalState.initialState = eatPillState;

    PacManState runAwayState = new PacManState(new AccionHuir(), "run away");
    normalState.addState(runAwayState);

    PacManState eatGhostState = new PacManState(new AccionPersecusion(), "eat ghost");
    root.addState(eatGhostState);

    final Condicion closeCondition = new Condicion() {
        @Override
        public boolean test() {
            if (closestNonBlueGhost < 0) {
                return false;
            }
            return (ghostDist[closestNonBlueGhost] < CLOSE_DIST);
        }
```

Método Huir:

Para este método como ya habíamos explicado brevemente, lo que hace es si detecta a un fantasma cerca entonces huira hasta perderlos de vista o encontrar una píldora para comer a los fantasmas y así mejorar su score.



aqui esta el codigo implementado para huir:

```
package Juego.Entradas.pacman;

import java.awt.Color;
import java.util.ArrayList;

import Juego.core.Game;
import Juego.core.GameView;
import ia.arboldecomportamiento.Selector;
import ia.arboldecomportamiento.Labor;

public class AccionHuir extends AccionPacMan {

        class RunForPowerPill extends Labor {

                @Override
                public boolean run() {

                        int current = game.getCurPacManLoc();

                        int[] targets = game.
                            getPowerPillIndicesActive();
```

```
                        if (targets.length > 0) {

                                int[][] paths = new int[4][];
                                int[] ghosts = new int[4];
                                int[] ghostDists = new int[4];


                                for (int i = 0; i < targets.
                                    length; i++) {
                                        paths[i] = game.getPath(
                                            current, targets[i]);
                                }


                    for (int i = 0; i < ghosts.length; i++) {
                            ghosts[i] = game.getCurGhostLoc(i);
                    ghostDists[i] = game.getPathDistance(current,
                        ghosts[i]);
                                }

                                boolean skipPath = false;
                                int targetIndex = -1;

                                int dist = Integer.MAX_VALUE;


for (int path = 0; (path < paths.length) && paths[path] != null;
     path++) {

                    // for each step of path
for (int step = 0; (step < paths[path].length) && !skipPath;
    step++) {


for (int ghost = 0; ghost < ghosts.length; ghost++) {


if (ghostDists[ghost] >= 0 && ghostDists[ghost] < MiPacMan.
    CLOSE_DIST) {

//int pathDist = game.getPathDistance(paths[path][step], ghosts[
    ghost]);
int myDist = game.getPathDistance(current, paths[path][step]);
int ghostDist = game.getGhostPathDistance(ghost, paths[path][
    step]);
if (ghostDist > 0 && ghostDist < MiPacMan.POWER_DIST && (
    ghostDist < myDist + 1)) {
//if (pathDist > 0 && pathDist < MyPacMan.POWER_DIST) {
                                                        skipPath
```

```
                                                                =
                                                               true;
                                                            break;
                                                    }
                                                     }

                                                        }
                                                }


                                if (!skipPath && paths[path].length <
                                    dist) {
                                        dist = paths[path].length;
                                        targetIndex = path;
                                                }

                                        skipPath = false;
                                        }

                                if (targetIndex != -1) {

GameView.addPoints(game, Color.GREEN, game.getPath(current,
    targets[targetIndex]));
//return game.getNextPacManDir(targets[targetIndex], true, Game.
    DM.PATH);
setTarget(game.getNextPacManDir(targets[targetIndex], true, Game
    .DM.PATH));

                                visitedJunctions.clear();
                                        return true;
                                        }
                                }
                                return false;
                        }

                }

                class RunForPill extends Labor {
                        @Override
                        public boolean run() {
                        int current = game.getCurPacManLoc();
                        int[] activePills = game.getPillIndicesActive();
                        int target = game.getTarget(current, activePills
                            , true, Game.DM.PATH);
                                int[] path = game.getPath(current,
                                    target);

                                boolean ghostExists = false;
```

```
// find path to nearest pill and check if ghost is on it
                        for (int step = 0; (step < path.length)
                            && !ghostExists; step++) {
                                for (int i = 0; i < Game.
                                    NUM_GHOSTS; i++) {
                                        if (path[step] == game.
                                            getCurGhostLoc(i)) {
                                                ghostExists =
                                                    true;
                                                break;
                                        }
                                }
                        }


                        if (!ghostExists) {
                                visitedJunctions.clear();

        GameView.addPoints(game, Color.MAGENTA, game.getPath(
            current, target));

        setTarget(game.getNextPacManDir(target, true, Game.DM.
            PATH));
                                return true;
                        }
                        return false;
                }

        }

        class RunForJunction extends Labor {

                @Override
                public boolean run() {
                        int i;
                        int j = -1;
                        int k;
                        int[] path;
                        int step;

                        int current = game.getCurPacManLoc();

                        if (game.isJunction(current)) {
                                visitedJunctions.add(current);
                        }

                        int[] junction = game.getJunctionIndices
                            ();
                        int[] array;
```

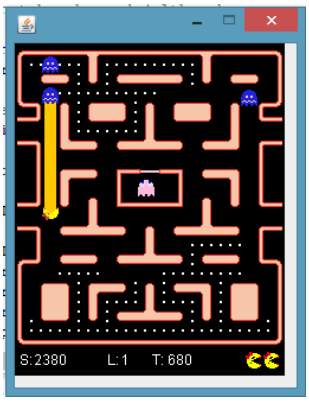
```
                        ArrayList<Integer> list = new ArrayList<
                            Integer>();

                        for (i = 0; i < junction.length; i++) {
                                list.add(junction[i]);
                        }

                        list.removeAll(visitedJunctions);

                        boolean foundPath = false;
                        boolean foundGhost = false;

                        while (!foundPath && !list.isEmpty()) {
                                array = new int[list.size()];

                                for (i = 0; i < array.length; i
                                    ++) {
                                        array[i] = list.get(i);
                                }

                                j = game.getTarget(current,
                                    array, true, Game.DM.PATH);
                                path = game.getPath(current, j);

                                foundGhost = false;
        for (step = 0; step < path.length && !foundGhost; step
            ++) {
                for (k = 0; k < Game.NUM_GHOSTS; k++) {
                        int ghostDist = game.
                            getGhostPathDistance(k, path[step]);
                int myDist = game.getPathDistance(current, path[
                    step]);

        //if (distance > 0 && distance < MyPacMan.JUNC_DIST) {
if (ghostDist > 0 && ghostDist < MiPacMan.JUNC_DIST && (
    ghostDist < myDist)) {
                                        foundGhost = true;
                                                list.remove(new
                                                    Integer(j));
                                                break;
                                                }
                                        }
                                }



                                foundPath = !foundGhost;
                        }
```

```
        if (foundPath) {
        GameView.addPoints(game, Color.LIGHT_GRAY, game.getPath(
            current, j));
                        setTarget(game.getNextPacManDir(j, true,
                            Game.DM.PATH));
                        return true;
                        }

                        return false;
                }

        }

        class NaiveRunAway extends Labor {

                @Override
        public boolean run() {
                int current = game.getCurPacManLoc();
                int[] ghostLocations = new int[Game.NUM_GHOSTS];

                        for (int i = 0; i < Game.NUM_GHOSTS; i
                            ++) {
                                ghostLocations[i] = game.
                                    getCurGhostLoc(i);
                        }

int target = game.getTarget(current, ghostLocations, true, Game.
    DM.PATH);
//GameView.addPoints(game, Color.RED, game.getPath(current,
    target));
setTarget(game.getNextPacManDir(target, false, Game.DM.PATH));
        visitedJunctions.clear();
                        return true;
                }

        }

        public static ArrayList<Integer> visitedJunctions = new
            ArrayList<Integer>();

        private int target = -1;
        Labor taskTree;
        Game game;

        public AccionHuir() {

                taskTree = new Selector();

                taskTree.children.add(new RunForPowerPill());
```

```
                //taskTree.children.add(new RunForPill());
                taskTree.children.add(new RunForJunction());
                taskTree.children.add(new NaiveRunAway());
        }

        public void setTarget(int target) {
                this.target = target;
        }

        @Override
        int act(Game game) {
                this.game = game;
                this.target = -1;

                if (!taskTree.run()) {
                        throw new RuntimeException("No Selection
                            ");
                }

                return target;
        }

}
```

Método persecución:

En esta método o estado, sucede cuando Ms. PacMan se come a una píldora de poder entonces es cuando entra en el estado de persecución, si detecta un fantasma cerca entonces empieza a perseguir a los fantasmas y los come, y luego que sale de esa estado entonces empieza a huir, esto se puede observar la máquina de estados proporcionado al principio.



aqui tenemos el codigo que implementamos:

```
package Juego.Entradas.pacman;

import java.awt.Color;
import java.util.ArrayList;

import Juego.core.Game;
import Juego.core.GameView;
import ia.arboldecomportamiento.Selector;
import ia.arboldecomportamiento.Labor;

public class AccionHuir extends AccionPacMan {

        class RunForPowerPill extends Labor {

                @Override
                public boolean run() {

                        int current = game.getCurPacManLoc();
```

```
                        int[] targets = game.
                            getPowerPillIndicesActive();

                        // existe alguna pildora de poder
                        if (targets.length > 0) {

                                int[][] paths = new int[4][];
                                int[] ghosts = new int[4];
                                int[] ghostDists = new int[4];

                        // llenar caminos
                        for (int i = 0; i < targets.length; i++)
                             {
                                paths[i] = game.getPath(current,
                                     targets[i]);
                                }


                        for (int i = 0; i < ghosts.length; i++)
                            {
                                ghosts[i] = game.getCurGhostLoc(
                                    i);
                                ghostDists[i] = game.
                                    getPathDistance(current,
                                    ghosts[i]);
                            }

                                boolean skipPath = false;
                                int targetIndex = -1;

                                int dist = Integer.MAX_VALUE;


for (int path = 0; (path < paths.length) && paths[path] != null;
     path++) {

                    // para cada paso del camino
for (int step = 0; (step < paths[path].length) && !skipPath;
    step++) {

        //campara nuevamente a los fantasmas
        for (int ghost = 0; ghost < ghosts.length; ghost++) {

        // fantasma en el rango
        if (ghostDists[ghost] >= 0 && ghostDists[ghost] <
            MiPacMan.CLOSE_DIST) {

        // fantasma en el camino
//int pathDist = game.getPathDistance(paths[path][step], ghosts[
```

```
    ghost]);
        int myDist = game.getPathDistance(current, paths[path][
            step]);
        int ghostDist = game.getGhostPathDistance(ghost, paths[
            path][step]);
        if (ghostDist > 0 && ghostDist < MiPacMan.POWER_DIST &&
            (ghostDist < myDist + 1)) {
                //if (pathDist > 0 && pathDist < MyPacMan.
                    POWER_DIST) {
                                                        skipPath
                                                            =
                                                        true;
                                                    break;
                                                    }
                                                }

                                            }
                                        }


                                        if (!skipPath && paths[path].
                                            length < dist) {
                                                dist = paths[path].
                                                    length;
                                                targetIndex = path;
                                        }

                                        skipPath = false;
                                }

                                if (targetIndex != -1) {

GameView.addPoints(game, Color.GREEN, game.getPath(current,
    targets[targetIndex]));

setTarget(game.getNextPacManDir(targets[targetIndex], true, Game
    .DM.PATH));

                                        visitedJunctions.clear();
                                        return true;
                                }
                        }
                        return false;
        }

        }

        class RunForPill extends Labor {
                @Override
```

```
                public boolean run() {
                        int current = game.getCurPacManLoc();
                        int[] activePills = game.
                            getPillIndicesActive();
                        int target = game.getTarget(current,
                            activePills, true, Game.DM.PATH);
                        int[] path = game.getPath(current,
                            target);

                        boolean ghostExists = false;


 for (int step = 0; (step < path.length) && !ghostExists; step
    ++) {
                        for (int i = 0; i < Game.NUM_GHOSTS; i
                            ++) {
                                if (path[step] == game.
                                    getCurGhostLoc(i)) {
                                        ghostExists = true;
                                        break;
                                }
                        }
                }

                // si no hay fantasma
                if (!ghostExists) {
                        visitedJunctions.clear();

 GameView.addPoints(game, Color.MAGENTA, game.getPath(current,
    target));

                setTarget(game.getNextPacManDir(target, true,
                    Game.DM.PATH));
                        return true;
                }
                return false;
        }

}

        class RunForJunction extends Labor {

                @Override
                public boolean run() {
                        int i;
                        int j = -1;
                        int k;
                        int[] path;
                        int step;
```

```
                        int current = game.getCurPacManLoc();

                        if (game.isJunction(current)) {
                                visitedJunctions.add(current);
                        }

                        int[] junction = game.getJunctionIndices
                            ();
                        int[] array;
                        ArrayList<Integer> list = new ArrayList<
                            Integer>();

                        for (i = 0; i < junction.length; i++) {
                                list.add(junction[i]);
                        }

                        list.removeAll(visitedJunctions);

                        boolean foundPath = false;
                        boolean foundGhost = false;

                        while (!foundPath && !list.isEmpty()) {
                                array = new int[list.size()];

                                for (i = 0; i < array.length; i
                                    ++) {
                                        array[i] = list.get(i);
                                }

                                j = game.getTarget(current,
                                    array, true, Game.DM.PATH);
                                path = game.getPath(current, j);

                        foundGhost = false;
                        for (step = 0; step < path.length &&
                            !foundGhost; step++) {
                                for (k = 0; k < Game.NUM_GHOSTS;
                                    k++) {
          int ghostDist = game.getGhostPathDistance(k, path[step
              ]);
                        int myDist = game.getPathDistance(
                            current, path[step]);

                //if (distance > 0 && distance < MyPacMan.
                    JUNC_DIST) {
if (ghostDist > 0 && ghostDist < MiPacMan.JUNC_DIST && (
    ghostDist < myDist)) {
                                        foundGhost = true;
```

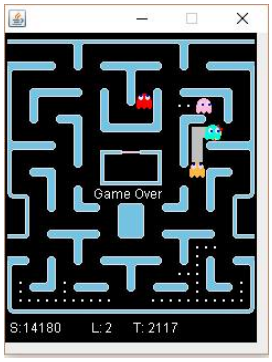
```
                                        list.remove(new Integer
                                            (j));
                                                break;
                                        }
                                }
                        }



                                foundPath = !foundGhost;
                        }

                        if (foundPath) {
                                GameView.addPoints(game, Color.
                                    LIGHT_GRAY, game.getPath(
                                    current, j));
                        setTarget(game.getNextPacManDir(j, true,
                             Game.DM.PATH));
                        return true;
                        }

                        return false;
                }

        }

        class NaiveRunAway extends Labor {

                @Override
                public boolean run() {
                int current = game.getCurPacManLoc();
                int[] ghostLocations = new int[Game.NUM_GHOSTS];

                for (int i = 0; i < Game.NUM_GHOSTS; i++) {
                                ghostLocations[i] = game.
                                    getCurGhostLoc(i);
                        }

int target = game.getTarget(current, ghostLocations, true, Game.
    DM.PATH);
        //GameView.addPoints(game, Color.RED, game.getPath(
            current, target));
                        setTarget(game.getNextPacManDir(target,
                            false, Game.DM.PATH));
                        visitedJunctions.clear();
                        return true;
                }

        }
```

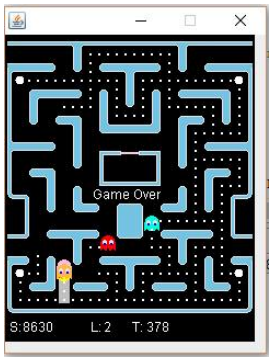
```
public static ArrayList<Integer> visitedJunctions = new
    ArrayList<Integer>();

        private int target = -1;
        Labor taskTree;
        Game game;

        public AccionHuir() {

                taskTree = new Selector();

                taskTree.children.add(new RunForPowerPill());

                taskTree.children.add(new RunForJunction());
                taskTree.children.add(new NaiveRunAway());
        }

        public void setTarget(int target) {
                this.target = target;
        }

        @Override
        int act(Game game) {
                this.game = game;
                this.target = -1;

                if (!taskTree.run()) {
                        throw new RuntimeException("No Selection
                            ");
                }

                return target;
        }

}
```

4. Resultados obtenidos

1. Ms. PacMan huye los fantasmas cercanos.
2. Ms. PacMan se come las pastillas y píldoras de poder más cercanas.
3. Ms. PacMan persigue a los fantasmas más cercanos si el tiempo de la píldora de energía es aun durable.

En los resultado obtenidos tenemos una inteligencia artificial que realiza las acciones proporcionadas por las líneas de código y después de haber ejecutado varias veces el código obtenemos que ha pasado hasta el nivel dos del juego, pero de ahí no ha pasado a más.
nota: para poder ver lo que hace el IA, tienes que juegar primero Tu, y despues juega la IA.





5. Conclusión

La realización de este proyecto no fue nada fácil ya que había que programar un agente inteligente y esto no es fácil de hacer ya que tenemos que considerar muchos casos y hacer que más a menos se vea que está tomando decisiones por sí mismo, tuvimos muchos problemas en su realización ya que no teníamos idea de cómo realizarlo, pero al final lo pudimos hacer con la ayuda de algunas páginas de internet y también nos basamos de un código ya implementado donde al final le cambiamos y pusimos nuestro propia IA, en las referencias pusimos la dirección donde sacamos el codigo que al final modificamos, pero en conclusión este proyecto me gustó mucho porque aprendimos más o menos como se hace un video juego cosa que no habíamos visto en toda la carrera.

6. Bibliografias

Referencias

I. A. para Jugadores No Humanos de Videojuegos,
title = https://sites.google.com/site/pfcgonzalotirado/inicio/proyecto/introduccion

ms-pacman-ai,
title = https://code.google.com/p/ms-pacman-ai/source/browse/src/pacman/entries/pacman/

Programando Pacman en c++ ,
title = https://www.youtube.com/watch?v=iA8zn62T7yw

pacman en java codigo tutorial ,
title = http://javayotros.blogspot.mx/2015/07/pacman-en-java-codigo-tutorial.html