Contents

[INTRODUCTION 1](#_Toc453098395)

[1. HISTORY OF PAC-MAN ORIGINAL GAME AND OBJECTIVES 1](#_Toc453098396)

[2- Project backbone 3](#_Toc453098397)

[2.0- ALGORITHM DESIGN 4](#_Toc453098398)

[**2.2- User Gui and user environment** 14](#_Toc453098399)

## INTRODUCTION

In 1980, Namco1 developed the now-ubiquitous Pac-Man, an arcade game starring an eponymous yellow circle who the player guides through a maze, eating dots, and avoiding enemy ghosts. The goal of this project is to fully

implement Pac-Man, from the graphical user interface to the artificially intelligent ghosts. The aim, in a lot of ways, will be to effectively emulate the Pac-Man experience with an AI .

## 1. HISTORY OF PAC-MAN ORIGINAL GAME AND OBJECTIVES

Pac-Man is a simple predator-prey style game, where the human player maneuvers an agent (i.e. Pac-Man) through a maze. The aim of the game is to score points, by eating dots initially distributed throughout the maze while attempting to avoid four “ghost” characters. If Pac-Man collides with a ghost, he loses one of his three lives (Whenever Pac-Man occupies the same tile as an enemy, he is considered to have collided with that ghost), and play resumes with the ghosts reassigned to their initial starting location (the “ghost cage” in the center of the maze). Four “PowerPills or PowerDots” are initially positioned near each corner of a maze: when Pac-Man eats a PowerPill he is able to turn the tables and eat the ghosts for a few seconds, The game ends when Pac-Man has lost all of his lives.  
  
In the original game there are no elements of randomness, each ghost makes deterministic decision about which direction to move at a given time step. Since the ghosts movement is dependent on the position and movement of the Pac-Man agent. As long as the Pac-Man agent displays variety in play, the game dynamics will unfold differently. This means that typical human players develop strategies for playing the game based on task prioritization, while planning and risk assessment have part in making any ghost agent AI the basic behaviors of the ghost agents can be described, the precise (programmed) strategies in the original arcade game appear to be unknown (reason is the short of translating machine code back into assembly language for the Z80microprocessor and reverse-engineering!).

Pac-Man (and variant) computer games have received some recent attention in Artificial Intelligence research. One reason is that the game provides a platform that is both simple enough to conduct experimental research and complex enough to require non-trivial strategies for successful gameplay.

The game is overly beaten and old and exists in too many variants to enumerate here. (Also One of the major challenges is making it playable and distinct). The development of a unique packman game has infiltrated many fields, for example some collages use the ghost AI for teaching purposes, Plus there is a version of the pacman game titled “MsPacman” which is officially used for competitions worldwide, The aim of this competition is to provide the best software automated controller for the game agent of MsPacMan. This is a great challenge for computational intelligence, machine learning, and AI in general.

Unlike Pac-Man, Ms. Pac-Man is a non-deterministic game, and rather difficult for most human players.  As far as we know, nobody really knows how hard it is to develop an AI player for the game.  The world record for a human player scoring (on the original arcade version) currently stands at 921,360 , Can anyone develop a software agent to beat that.. that is what this competition is all about, The Ms. Pac-Man competition will test the ability of computer-based players at the conference.  We are especially interested in players that use computational intelligence methods to address the problem, but the contest is open to any type of algorithm: you can hand-program it as much as you like. And the results are posted by IEEE reference [1][2]

## 2- Project backbone

This paper aims to describe the process of creating a unique playable and different Pacman game through our development of the “Ghost agents” AI, our algorithm aims to show the importance of the Randomness factor while navigating through a maze, and how some deterministic factors combined with randomness could create a well-balanced environment for a game, the implementation and the process of building an applet, This is an intelligent pacman game written in Java. The original goal of the project is to demonstrate how randomness contributes to the intelligent movements of the ghosts while dealing with the game environment with all of its variables and states, the movement of the ghosts is what makes our game project unique and interesting.

## 2.1- ALGORITHM DESIGN

The game logic of original Pacman game is straightforward and simple.

In order to make it more interesting, game logic design is another important

design part we concentrated on this project.



**Figure** 2.1: States of a Robot Ghost

in Figure 2.1 we show that Our ghosts are built over the uses of a Finite State Machine , The ghosts have three states and three transitions in the game:

\_ S0: Robot Ghost is walking randomly in Maze

\_ S1: Robot Ghost meets the Pacman (chase state)

\_ S2: Robot Ghost dies (in the state where pacman eats a PowerDot)

\_ T0: Robot Ghost transitions from S0 to S1

\_ T1: Robot Ghost transitions from S1 to S2

\_ T2: Reset Robot Ghost to S0

At the stare we show the initial state of the maze in figure 2.2

Where there are four PowerDots at the corners of the maze, and all the ghosts are in a cage, pacman cat go inside the cage, and any ghost inside that cage won’t be affected by the power pill effect (won’t have any of the main states till they leave outside of it)

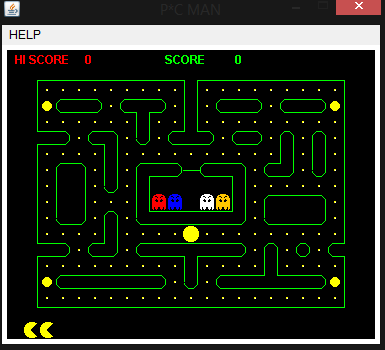


Figure 2.2

When building the ghosts, we programmed them to start as blind for a set numbers of frames, we set their speed in that state (while walking randomly in state S0) as full speed which is same as the speed of pacman, the ghosts have knowledge of the maze at all times, and move through iX and iY axes inside the maze (i.e. always knows its place inside the maze), the ghost in this state will be choosing one of the available directions randomly at every intersect.

once the number of the blind frames is zero, the ghost will change states into the second state of “ghost chasing pacman S1” this is when the behavior of ghost takes smart AI movement, where they have priorities while choosing the random directions, the ghosts will go roam around the power dots, till it spots the pacman (the PowerDots are located in one of the four corner blocks of the maze, one in each corner, Four in total, its score will be added to the calculations that affect the second state path choice (descried more later).

Once the number of the blind frames is zero, The ghosts will change into the third state which is “Flee state S3” as shown in figure 2.3, in that state they will change color, and instead of chasing pacman, they will flee to a changeable fixed point outside the maze where they will always keep roaming, the speed of the ghosts will be at half of the normal speed and pacman will be able to eat them at that state, resetting them back to the initial starting position (inside the cage in the middle of the maze) where they will be reset, after a fixed frame time, the powerDot effect will fade, and the ghosts will be back to the initial random state to restart the loop of states.

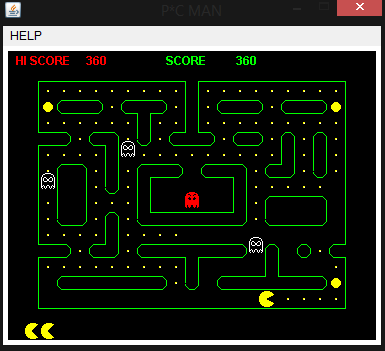


Figure 2.3: showing the flee state S3

**2.2- User Guide and Game Environment**

Our process of java coding started with a base idea to build an applet first which will contain all the features separately while the build process should be in blocks and separate classes for each function, because such projects are always made with the purpose of expansion and development, as we see in figure 3.1 the applet window which holds a play button that runs the app we built, and if at any point we decide to add more functions, or different apps, we can just add them separately to the applet page interface, the applet provided by eclipse contain a debugging function that will show any errors, the applet is implemented within “applet.class” which implements “java.applet.Applet” library provided by java

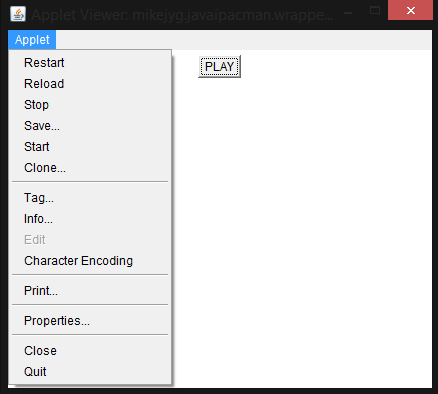


Figure 3.1 applet interface

Once Play is pressed the compiled app will be runned and it will show a window provided in figure 3.2 which is our core game that will start running and taking keyboard input once the user press the arrow Up key

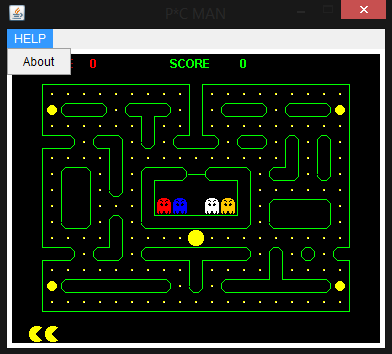


Figure 3.2 application window

1. Ghost AI Implementation

Code Implementation Examples:

First we start with a ghost movement rules and decision making which we set as follows:

1. All states are bound to a fixed timer which is programmed as a set of frames (which we can decrease if we want the game Level to be harder, ex: scatter 7 frames, chase 7 frames, then scatter again etc.)

// remain blind for ??? frames

Final int INIT\_BLIND\_COUNT=600;

//at the start state blind counter

blindCount=INIT\_BLIND\_COUNT/((round+1)/2);

They can never (in all states) go back the same direction path as the one they came from.

// random calculation factors

final int DIR\_FACTOR=2;

final int POS\_FACTOR=10;

// the maze the ghosts knows

cmaze maze;

1. The ghost in random scatter mode will move based on available paths, but will have the ability to scatter in different directions to cover most of the maze and not leave many openings

// randomly select a direction

if (iDirTotal!=0)

{

iRand=cuty.RandSelect(iDirTotal);

if (iRand>=iDirTotal)

throw new Error("iRand out of range");

// exit(2);

for (i=0; i<4; i++)

{

iM=maze.iMaze[iY/16+ ctables.iYDirection[i]]

[iX/16+ ctables.iXDirection[i]];

if (iM!=cmaze.WALL && i!= ctables.iBack[iDir] && iM!= cmaze.DOOR )

{

iRand-=iDirCount[i];

if (iRand<0)

// the right selection

{

iDir=i; break;

}

}

}

}

else

throw new Error("iDirTotal out of range");

// exit(1);

return(iDir);

}

1. They will try and choose different paths when in chase mode (the ghost won’t just follow you, they will try and surround you from different directions) this would really show when the game is almost over, where the Ai will use calculations and accounting factors to make a surrounding move of the pacman agent, some of the factors affecting our decision making process were implemented as followed:
   1. Ghost status.
   2. Ghost position to each other and in the maze.
   3. Ghost movement direction and choice when chasing.
   4. Dot positions.
   5. Only the closest dot is goal.
   6. PowerDot position.
   7. The closer the ghost to a pacman, more weight to go to power dot.
   8. Direction based on score (disperse to cover around all of the dots that are not yet eaten by pacman) They will disperse in different numbers based on the location score (cover all the maze at all times), the location with higher score will have more ghosts.
   9. If the location scores is not different, or not available, the opposite score will be chosen and the ghost will change priorities to catch pacman while moving between the four corners of the maze Then they will compare and calculate for a new decision again.

public int GetNextDir()

throws Error

{

int i;

// first, init to 0

for (i=0; i<4; i++)

iDirScore[i]=0;

// add score for dot

AddDotScore();

// add score for ghosts

AddGhostScore();

// add score for powerdot

AddPowerDotScore();

// determine the direction based on scores

for (i=0; i<4; i++)

iValid[i]=1;

int iHigh, iHDir;

while (true)

{

iHigh=-1000000;

iHDir=-1;

for (i=0; i<4; i++)

{

if (iValid[i] == 1 && iDirScore[i]>iHigh)

{

iHDir=i;

iHigh=iDirScore[i];

}

}

if (iHDir == -1)

{

throw new Error("cpacmove: can't find a way?");

}

if ( cPac.iX%16 == 0 && cPac.iY%16==0)

{

if ( cPac.mazeOK(cPac.iX/16 + ctables.iXDirection[iHDir],

cPac.iY/16 + ctables.iYDirection[iHDir]) )

return(iHDir);

}

else

return(iHDir);

iValid[iHDir]=0;

iDirScore[ctables.iBack[iHDir]] = iDirScore[iHDir];

}

//calculating score example

void AddGhostScore()

{

int iXDis, iYDis; // distance

double iDis; // distance

int iXFact, iYFact;

// calculate ghosts one by one

for (int i=0; i<4; i++)

{

iXDis=cGhost[i].iX - cPac.iX;

iYDis=cGhost[i].iY - cPac.iY;

iDis=Math.sqrt(iXDis\*iXDis+iYDis\*iYDis);

if (cGhost[i].iStatus == cGhost[i].BLIND)

{

}

else

{

// adjust iDis into decision factor

iDis=18\*13/iDis/iDis;

iXFact=(int)(iDis\*iXDis);

iYFact=(int)(iDis\*iYDis);

if (iXDis >= 0)

{

iDirScore[ctables.LEFT] += iXFact;

}

else

{

iDirScore[ctables.RIGHT] += -iXFact;

}

if (iYDis >= 0)

{

iDirScore[ctables.UP] += iYFact;

}

else

{

iDirScore[ctables.DOWN] += -iYFact;

}

}

}

}

1. When the powerDot is taken by pacman, the ghosts will be in S3, they will scatter at half of pacman speed, and will have a changeable unreachable point to go to, which is located outside the maze frame (meaning the ghost will keep scattering but not to a predictable point, because random moving agent would be presented on the iX IY access)
2. The win state

// return 1 if caught the pac!

// return 2 if being caught by pac

int testCollision(int iPacX, int iPacY)

{

if (iX<=iPacX+2 && iX>=iPacX-2

&& iY<=iPacY+2 && iY>=iPacY-2)

{

switch (iStatus)

{

case OUT:

return(1);

case BLIND:

iStatus=EYE;

iX=iX/4\*4;

iY=iY/4\*4;

return(2);

}

}

// nothing

return(0);

}

1. Special case movements and conditions for example dealing with the home of the ghosts (the box where they start, and go back to when they get eaten by pacman, and where pacman cant inter)

if (iStatus==BLIND || iStatus==OUT)

{

iStatus=BLIND;

iBlindCount=blindCount;

iBlink=0;

// reverse

if (iX%16!=0 || iY%16!=0)

{

iDir= ctables.iBack[iDir];

// a special condition where the ghosts always leave the home (the box in the middle of the maze where they start, and where pacman cant inter) they leave blind (in random scatter mode)

int iM;

iM=maze.iMaze[iY/16+ ctables.iYDirection[iDir]]

[iX/16+ ctables.iXDirection[iDir]];

if (iM == cmaze.DOOR)

iDir=ctables.iBack[iDir];

}

}

1. Game Environment implementation

Here we show some of the core classes as a walkthrough of the building blocks and features:

1. cpcman: the main class of our pacman game containing the base of our build the game variables such as score pacman lives (number of time you can restart before game over) and the key components for a healthy initial start of a level

// the timer

Thread timer;

int timerPeriod=12; // in miliseconds

// the timer will increment this variable to signal a frame

int signalMove=0;

// the canvas starting point within the frame

int topOffset;

int leftOffset;

// the draw point of maze within the canvas

final int iMazeX=16;

final int iMazeY=16;

// the off screen canvas for the maze

Image offScreen;

Graphics offScreenG;

// the objects

cmaze maze;

cpac pac;

cpowerdot powerDot;

cghost [] ghosts;

// game counters

final int PAcLIVE=3;

int pacRemain;

int changePacRemain; // to signal redraw remaining pac

block listing some of the key objects initialization

// initialize maze object

maze = new cmaze(this, offScreenG);

// initialize ghosts object

// 4 ghosts

ghosts = new cghost[4];

for (int i=0; i<4; i++)

{

Color color;

if (i==0)

color=Color.red;

else if (i==1)

color=Color.blue;

else if (i==2)

color=Color.white;

else

color=Color.orange;

ghosts[i]=new cghost(this, offScreenG, maze,color);

}

// initialize power dot object

powerDot = new cpowerdot(this, offScreenG, ghosts);

// initialize pac object

pac = new cpac(this, offScreenG, maze, powerDot, ghosts);

pac = new cpac(this, offScreenG, maze, powerDot);

the block which is setting the maze dimension and painting everything

// set the proper size of canvas

Insets insets=getInsets();

topOffset=insets.top;

leftOffset=insets.left;

// updating the frame

powerDot.draw();

for (int i=0; i<4; i++)

ghosts[i].draw();

pac.draw();

// display extra information

if (changeHiScore==1)

{

imgHiScoreG.setColor(Color.black);

imgHiScoreG.fillRect(70,0,80,16);

imgHiScoreG.setColor(Color.red);

imgHiScoreG.drawString(Integer.toString(hiScore), 70,14);

g.drawImage(imgHiScore, 8+ leftOffset, 0+ topOffset, this);

changeHiScore=0;

}

if (changeScore==1)

{

imgScoreG.setColor(Color.black);

imgScoreG.fillRect(70,0,80,16);

imgScoreG.setColor(Color.green);

imgScoreG.drawString(Integer.toString(score), 70,14);

g.drawImage(imgScore,

158+ leftOffset, 0+ topOffset, this);

changeScore=0;

}

// update pac life info

if (changePacRemain==1)

{

int i;

for (i=1; i<pacRemain; i++)

{

g.drawImage(pac.imagePac[0][0],

16\*i+ leftOffset,

canvasHeight-18+ topOffset, this);

}

g.drawImage(powerDot.imageBlank,

16\*i+ leftOffset,

canvasHeight-17+ topOffset, this);

changePacRemain=0;

}

There is also some key factors for a game environment that we wont be listing the code for such as “Control and moves” block which is the routine running at the background of drawings and the routine that draw each frame (updating the maze objects as game progress), followed by” process key input” (keyboard input for controlling the pacman agent), followed by the handler of the menu event (buttons and such), handling the timer (stopping the clock at death, restarting the game etc.)

### Future plans for our smart pacman project :

We have started working on an automation factor for the pacman, the computer controlled auto play of the pacman agent feature is work in progress and does not work properly yet, we are building it while taking some factors into considerations and are set to follow these steps:

1. Building a pacman automation that will learn through trying and is able to beat our current project ghosts.
2. A pacman automation that is able to beat the original pacman game with a high score.
3. A pacman that have a unique changeable AI that can adapt to hard mode (ghost state timers are set to have chase at most times).
4. Having the ability to implementing a smarter ghost AI that is able to chase the pacman itself in a set of coordinated states unique to each of the ghosts.

[Pac Man papers in IEEE CIG](http://www.csse.uwa.edu.au/cig08/Proceedings/search.html?cx=000061790505554880205:oanpsvp2jsm&cof=FORID:10&ie=UTF-8&q=pacman&sa=Search#393) [1]

[IEEE Symposium on Computational Intelligence and Games](http://cigames.org/) [2]