PAC-MAN’S GHOSTS

THEORY OF COMPUTATION  
CASE STUDY – TERM 2  
GROUP-2

|  |  |
| --- | --- |
| NAME | ROLL NUMBER |
| Harini S | CB.EN.U4CSE19023 |
| Rithika Sri J | CB.EN.U4CSE19025 |
| Tarsana B | CB.EN.U4CSE19057 |

**THEORY OF COMPUTATION IN VIDEO GAMES:**Finite automata lend themselves to representing the behaviour of computer controller characters in video games. The states of the automata correspond to the character’s behaviours, which change according to various events. These changes are modelled by transitions in the state diagram. Automata are certainly not the most sophisticated means of implementing artificially intelligent agents in games, but many games include characters with simple, state-based behaviours that are easily and effectively modelled using automata/FSM.

**PROBLEM DEFINITION:**  
Pac-Man requires the player to navigate through a maze, eating pellets and avoiding the ghosts who chase him through the maze. Occasionally, Pac-Man can turn the tables on his pursuers by eating a power pellet, which temporarily grants him the power to eat the ghosts. When this occurs, the ghosts’ behaviour changes, and instead of chasing Pac-Man they try to avoid him. The ghosts in Pac-Man have four behaviours:

1. Randomly wander the maze   
2. Chase Pac-Man, when he is within line of sight  
3. Flee Pac-Man, after Pac-Man has consumed a power pellet  
4. Return to the central base to regenerate



**SPECIFICATIONS OF TOKENS IN PAC-MAN:**

1. **ALPHABETS:**

Any Finite set of symbols is called alphabets.

Each alphabet corresponds to the activity (current behaviour) of the Pac-Man and the ghosts.

Set of Alphabets for the Pac-Man game è  ∑= {l, s, e, p, n, d, r} where  
l à loses Pac-Man (out of sight)

s à spots Pac-Man

e à eats Pac-Man

p à Pac-Man eats power pellet

n à power pellet expires, Pac-Man returns to normal

d à eaten by Pac-Man and dies

r à regenerate from the central base

1. **STRINGS:**

Any finite sequence of alphabets is called a string.

Σ\* is set of all possible strings

In Pac-Man,

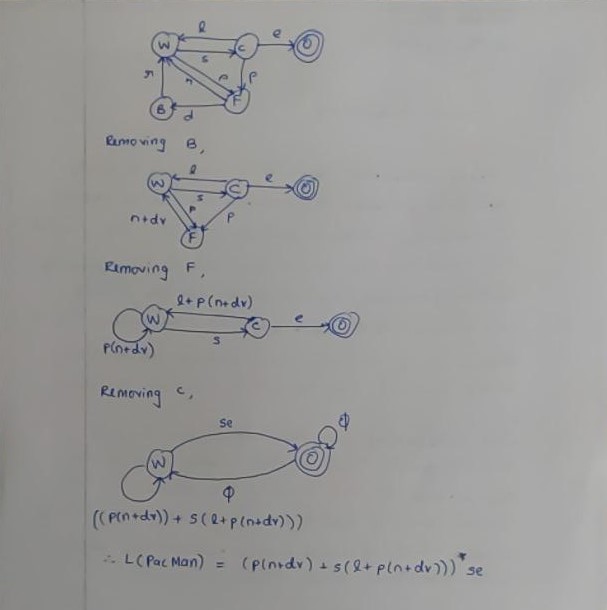
Σ\*={ ε, l, s, e, p, n, d, r, ls, se, ep, ndr, sepn, ppn, ddlsse, ...…. } are the set of some strings over the alphabets mentioned above.

Length of the string is the total number of occurrence of alphabets and denoted as |w| where ‘w’ is a string.

A string having no alphabets(a string of length zero) is known as an empty string and is denoted by ε (epsilon).

**LANGUAGE DESCRIPTION:**

A language is considered as a finite set of strings over some finite set of alphabets.  
(1) The language describing the Pac-Man game comprises of all possible   
 strings in which each string represents a trial of the game.  
(2) Each trial of the game must start with ghosts wandering and end with the   
 Pac-Man getting spotted(s) and eaten(e) up by the ghosts.  
(3) Also that the behaviors of the ghost must be ordered.   
(4) In Pac-Man game, the length of each string is not fixed because the game   
 is not time nor length bounded.  
  
Thus the language described by the Pac-Man game through ghost’s action is  
L = {x : x **∈** Σ+ and x starts with ‘s or p’, ‘s or p’ occurs after ‘n or l or r’, ‘p or e’ occurs after ‘s’, ’n or d’ occurs after ‘p’, ‘r’ occurs after ‘d’ and ends with ‘se’ always}  
L = {se, spdrse, pnse, pdrse, spnpdrse, slse ….}  
The language described by the Pac-Man game is a regular language because it does not need any memory and there is no need to store nor count the alphabets used. The regular grammar also is right-linear or left-linear (given below). Thus, Automata can be drawn.   
It is noted that it is countable infinite regular language because the game getting over is certain but no one knows when it comes to an end.  
  
The regular language described by the Pac-Man game is vague (given above). It can be denoted as regular expression. It is a much simple way to describe using algebraic expressions.  
L(Pac-Man) = (p (n + dr) + s (l + p (n + dr))) \* se   
(Regex was derived from NFA to regex conversion)

**NFA TO REGEX:**

**GRAMMAR:**G = (V, T, P, S)  
V: Set of states = {W, B, C, F, O}  
T: Inputs = {s, l, e, r, n, p, d}  
S: Initial state = {W}  
P: Production rules

{

W -> sC |pF

C -> pF | lW | e

F -> dB | nW

B -> rW

}

**AUTOMATA:**

**States**: W for “wander the maze”; C for “chase Pac-Man”; F for “Flee Pac-Man”; B for “return to base”; O for “game over”.

**Transitions**: s for “spots Pac-Man”; p for “Pac-Man eats power pellet; l for “loses Pac-Man”; e for “eats Pac-Man”; n for “power pellet expires, Pac-Man returns to normal”; d for “eaten by Pac-Man and dies”; r for “regenerate from the central base.”

**State Transition Table:**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | STATE | INPUTS | | | | | | |  |
|  |  | s | l | n | p | d | r | e |  |
|  | W | C | ~ | ~ | F | ~ | ~ | ~ |  |
|  | C | ~ | W | ~ | F | ~ | ~ | O |  |
|  | B | ~ | ~ | ~ | ~ | ~ | W | ~ |  |
|  | F | ~ | ~ | W | ~ | B | ~ | ~ |  |
|  | O | ~ | ~ | ~ | ~ | ~ | ~ | ~ |  |

**Deterministic Finite Automata (DFA):**

**M = (Q, Σ, δ, q0, F)**

**Q:** Finite set of internal states = {W,B,C,F,O}

**Σ:** Finite set of symbols, Inputs alphabets = {s, l, e, r, n, p, d}

**δ:** Total function, Transition function

**q0:** Initial state = W

**F:** Set of Final State = {O}

**M = ({W, B, C, F, O}, {s, l, e, r, n, p, d}, δ, W, {O})**

**δ** = {

δ(W, s) = C

δ(W, p) = F

δ(C, p) = F

δ(C, l) = W

δ(C, e) = O

δ(F, d) = B

δ(F, n) = W

δ (B, r) = W

}

**TRANSITION GRAPH**

**Deterministic Finite Automata (DFA)**  
