Vidyavardhaka Sangha®, Mysore
VIDYAVARDHAKA COLLEGE OF ENGINEERING
Autonomous Institute, Affiliated to Visvesvaraya Technological University, Belagavi
(Approved by AICTE, New Delhi & Government of Karnataka)
Accredited by NBA | NAAC with 'A' Grade
Department of Computer Science & Engineering
Phone: +91 821-4276230, Email: hodcs@vvce.ac.in
Web: http://www.vvce.ac.in
@vvceofficial

AUTOMATA THEORY (20CS53)

ACTIVITY BASED ASSESSMENT

TOPIC: DETERMINISTIC FINITE AUTOMATA (DFA) FOR VIDEO GAMES

BY,

1. **ECHCHITHA S SHETTY (4VV20CS036)**
2. **ESHANYE SRINIVAS (4VV20CS038)**
3. **HARSHITHA M M (4VV20CS049)**

DFA FOR VIDEO GAMES (PAC-MAN GAME)

DFA (Deterministic finite automata)

Deterministic finite automata (or DFA) are finite state machines that accept or reject strings of characters by parsing them through a sequence that is uniquely determined by each string.

The term “deterministic” refers to the fact that each string, and thus each state sequence, is unique. In a DFA, a string of symbols is parsed through a DFA automata, and each input symbol will move to the next state that can be determined.

These machines are called finite because there are a limited number of possible states which can be reached. A finite automaton is only called deterministic if it can fulfill both conditions. DFAs differ from non-deterministic automata in that the latter are able to transition to more than one state at a time and be active in multiple states at once.

In practice, DFAs are made up of five components (and they’re often denoted by a five-symbol set known as a 5-tuple). These components include:

- A finite number of states
- A set of symbols known as the alphabet, also finite in number
- A function that operates the transition between states for each symbol
- An initial start state where the first input is given or processed
- A final state or states, known as accepting states.

PAC-MAN GAME

ABSTRACT

To emulate the gameplay of Pac-Man, a popular maze arcade game, with the help of Finite State Machines. Artificial intelligence plays an important role in making video games more interactive and mathematical models like finite state machines have provided a backbone for implementation of game AI.

GAMEPLAY OF PAC-MAN:

The game requires the player to control the eponymous character through an enclosed maze, eating dots (or pellets) and avoid the ghosts. Larger "power" pellets are scattered around the corners of the maze, which when eaten grant Pac-

Man the power to eat the ghosts, so they start to avoid it until the pellet's effect lasts. The game is won when Pac-Man eats all the pellets.

THE MACHINE MODEL FOR THE GHOSTS OF PAC-MAN

A finite state machine will be created for the non-player characters for the game, i.e., the ghosts. The ghosts can be in one of the following states at any point of time:

1) **WANDER**: Default state from the beginning of game, the ghost would exit their base and wander around the maze randomly.

2**) CHASE**: If the player (Pac-Man) is in range of a ghost, the ghost would start chasing Pac-Man, but only if the power pellet is not in effect.

3) **FLEE**: If Pac-Man eats a power pellet, it would be able to eat the ghosts to gain extra points, so the ghosts start to flee away from Pac-Man while the effect of the pellet lasts.

4) **DEAD**: The ghost "dies" if Pac-Man eats (touches) it. Only its eyes remain which flee back to the base and the ghost respawns.

5)**GWON**: The player loses if a ghost touches (eats) Pac-Man.

6)**PWON**: The player wins when Pac-Man has eaten all the pellets in the maze before being eaten by a ghost.

So, we have the following set of internal states as:

Q = {WANDER, CHASE, FLEE, DEAD, GWON, PWON} with

q_0 = WANDER being our initial state and F = {PWON} being the set of final states.

To define our input alphabet Σ, we have the following actions:

1) S: Stands for SPOT. This happens when Pac-Man is in the close range of a ghost and the ghosts start chasing it.

2) L: Stands for LOSE. This happens when Pac-Man is farther from the ghosts.

3) EP: Stands for EAT PELLET. This means Pac-Man has eaten a larger power pellet.

4) PEX: Stands for PELLET EXPIRED. This means the effects of the power pellet have worn off.

5) PEG: Stands for Pac-Man Eats Ghost. Pac-Man eats a ghost under the effects of the power pellet.

6) RET: Stands for RETURN. The "eyes" of the ghost return to the base.

7) GEP: Stands for Ghost Eats Pac-Man.

8) OVER: Pac-Man has eaten all the pellets. So, we have the following input alphabet:

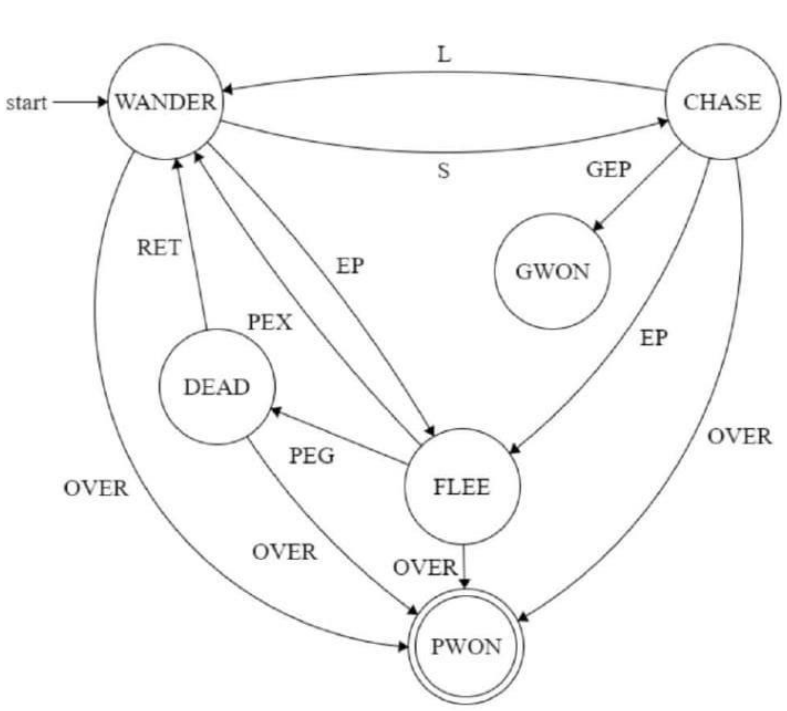
Σ = {S, L, EP, PEX, PEG, RET, GEP, OVER} Hence, we get the following

Deterministic Finite Acceptor: M = (Q, Σ, δ, q0, F)

TRANSITION TABLE:

	S	L	EP	PEX	PEG	RET	GEP	OVER
WANDER	CHASE		FLEE	-	-	-	-	PWON
CHASE	-	WANDER	FLEE	-	-	-	GWON	PWON
FLEE	-	-	-	WANDER	DEAD	-	-	PWON
DEAD	-	-	-	-	-	WANDER	-	-
GWON	-	-	-	-	-	-	-	-
PWON	-	-	-	-	-	-	-	-

TRANSITION DIAGRAM:



PYTHON PROGRAM TO IMPLEMENT PAC-MAN:

```
import pygame._view

black = (0,0,0)
white = (255,255,255)
blue = (0,0,255)
green = (0,255,0)
red = (255,0,0)
purple = (255,0,255)
yellow   = ( 255, 255,   0)

Trollicon=pygame.image.load('images/Trollman.png')
pygame.display.set_icon(Trollicon)

#Add music
pygame.mixer.init()
pygame.mixer.music.load('pacman.mp3')
pygame.mixer.music.play(-1, 0.0)

# This class represents the bar at the bottom that the player controls
class Wall(pygame.sprite.Sprite):
    # Constructor function
    def __init__(self,x,y,width,height, color):
        # Call the parent's constructor
        pygame.sprite.Sprite.__init__(self)

        # Make a blue wall, of the size specified in the parameters
        self.image = pygame.Surface([width, height])
        self.image.fill(color)

        # Make our top-left corner the passed-in location.
        self.rect = self.image.get_rect()
        self.rect.top = y
        self.rect.left = x

# This creates all the walls in room 1
def setupRoomOne(all_sprites_list):
    # Make the walls. (x_pos, y_pos, width, height)
    wall_list=pygame.sprite.RenderPlain()

    # This is a list of walls. Each is in the form [x, y, width, height]
    walls = [ [0,0,6,600],
              [0,0,600,6],
              [0,600,606,6],
              [600,0,6,606],
              [300,0,6,66],
              [60,60,186,6],
              [360,60,186,6],
              [60,120,66,6],
              [60,120,6,126],
              [180,120,246,6],
              [300,120,6,66],
              [480,120,66,6],
              [540,120,6,126],
              [120,180,126,6],
              [120,180,6,126],
              [360,180,126,6],
              [480,180,6,126],
              [180,240,6,126],
              [180,360,246,6],
              [420,240,6,126],
              [240,240,42,6],
              [324,240,42,6],
              [240,240,6,66],
              [240,300,126,6],
```

```
                [120,180,6,126],
                [360,180,126,6],
                [480,180,6,126],
                [180,240,6,126],
                [180,360,246,6],
                [420,240,6,126],
                [240,240,42,6],
                [324,240,42,6],
                [240,240,6,66],
                [240,300,126,6],
                [360,240,6,66],
                [0,300,66,6],
                [540,300,66,6],
                [60,360,66,6],
                [60,360,6,186],
                [480,360,66,6],
                [540,360,6,186],
                [120,420,366,6],
                [120,420,6,66],
                [480,420,6,66],
                [180,480,246,6],
                [300,480,6,66],
                [120,540,126,6],
                [360,540,126,6]
              ]

    # Loop through the list. Create the wall, add it to the list
    for item in walls:
        wall=Wall(item[0],item[1],item[2],item[3],blue)
        wall_list.add(wall)
        all_sprites_list.add(wall)

    # return our new list
    return wall_list

def setupGate(all_sprites_list):
      gate = pygame.sprite.RenderPlain()
      gate.add(Wall(282,242,42,2,white))
      all_sprites_list.add(gate)
      return gate

# This class represents the ball
# It derives from the "Sprite" class in Pygame
class Block(pygame.sprite.Sprite):

    # Constructor. Pass in the color of the block,
    # and its x and y position
    def __init__(self, color, width, height):
        # Call the parent class (Sprite) constructor
        pygame.sprite.Sprite.__init__(self)

        # Create an image of the block, and fill it with a color.
        # This could also be an image loaded from the disk.
        self.image = pygame.Surface([width, height])
        self.image.fill(white)
        self.image.set_colorkey(white)
        pygame.draw.ellipse(self.image,color,[0,0,width,height])

        # Fetch the rectangle object that has the dimensions of the image
        # image.
        # Update the position of this object by setting the values
        # of rect.x and rect.y
        self.rect = self.image.get_rect()

```

```
121
122 # This class represents the bar at the bottom that the player controls
123 class Player(pygame.sprite.Sprite):
124
125     # Set speed vector
126     change_x=0
127     change_y=0
128
129     # Constructor function
130     def __init__(self,x,y, filename):
131         # Call the parent's constructor
132         pygame.sprite.Sprite.__init__(self)
133
134         # Set height, width
135         self.image = pygame.image.load(filename).convert()
136
137         # Make our top-left corner the passed-in location.
138         self.rect = self.image.get_rect()
139         self.rect.top = y
140         self.rect.left = x
141         self.prev_x = x
142         self.prev_y = y
143
144     # Clear the speed of the player
145     def prevdirection(self):
146         self.prev_x = self.change_x
147         self.prev_y = self.change_y
148
149     # Change the speed of the player
150     def changespeed(self,x,y):
151         self.change_x+=x
152         self.change_y+=y
153
154     # Find a new position for the player
155     def update(self,walls,gate):
156         # Get the old position, in case we need to go back to it
157
158         old_x=self.rect.left
159         new_x=old_x+self.change_x
160         prev_x=old_x+self.prev_x
161         self.rect.left = new_x
162
163         old_y=self.rect.top
164         new_y=old_y+self.change_y
165         prev_y=old_y+self.prev_y
166
167         # Did this update cause us to hit a wall?
168         x_collide = pygame.sprite.spritecollide(self, walls, False)
169         if x_collide:
170             # Whoops, hit a wall. Go back to the old position
171             self.rect.left=old_x
172             # self.rect.top=prev_y
173             # y_collide = pygame.sprite.spritecollide(self, walls, False)
174             # if y_collide:
175             #     # Whoops, hit a wall. Go back to the old position
176             #     self.rect.top=old_y
177             #     print('a')
178         else:
179
180             self.rect.top = new_y
181
182             # Did this update cause us to hit a wall?
183             y_collide = pygame.sprite.spritecollide(self, walls, False)
184             if y_collide:
```

```
                # Whoops, hit a wall. Go back to the old position
                self.rect.top=old_y
                # self.rect.left=prev_x
                # x_collide = pygame.sprite.spritecollide(self, walls, False)
                # if x_collide:
                #     # Whoops, hit a wall. Go back to the old position
                #     self.rect.left=old_x
                #     print('b')

        if gate != False:
          gate_hit = pygame.sprite.spritecollide(self, gate, False)
          if gate_hit:
            self.rect.left=old_x
            self.rect.top=old_y

#Inheritime Player klassist
class Ghost(Player):
    # Change the speed of the ghost
    def changespeed(self,list,ghost,turn,steps,l):
      try:
        z=list[turn][2]
        if steps < z:
          self.change_x=list[turn][0]
          self.change_y=list[turn][1]
          steps+=1
        else:
          if turn < l:
            turn+=1
          elif ghost == "clyde":
            turn = 2
          else:
            turn = 0
          self.change_x=list[turn][0]
          self.change_y=list[turn][1]
          steps = 0
        return [turn,steps]
      except IndexError:
         return [0,0]

Pinky_directions = [
[0,-30,4],
[15,0,9],
[0,15,11],
[-15,0,23],
[0,15,7],
[15,0,3],
[0,-15,3],
[15,0,19],
[0,15,3],
[15,0,3],
[0,15,3],
[15,0,3],
[0,-15,15],
[-15,0,7],
[0,15,3],
[-15,0,19],
[0,-15,11],
[15,0,9]
]

Blinky_directions = [
[0,-15,4],
[15,0,9],
[0,15,11],
```

```
[15,0,3],
[0,15,7],
[-15,0,11],
[0,15,3],
[15,0,15],
[0,-15,15],
[15,0,3],
[0,-15,11],
[-15,0,3],
[0,-15,11],
[-15,0,3],
[0,-15,3],
[-15,0,7],
[0,-15,3],
[15,0,15],
[0,15,15],
[-15,0,3],
[0,15,3],
[-15,0,3],
[0,-15,7],
[-15,0,3],
[0,15,7],
[-15,0,11],
[0,-15,7],
[15,0,5]
]

Inky_directions = [
[30,0,2],
[0,-15,4],
[15,0,10],
[0,15,7],
[15,0,3],
[0,-15,3],
[15,0,3],
[0,-15,15],
[-15,0,15],
[0,15,3],
[15,0,15],
[0,15,11],
[-15,0,3],
[0,-15,7],
[-15,0,11],
[0,15,3],
[-15,0,11],
[0,15,7],
[-15,0,3],
[0,-15,3],
[-15,0,3],
[0,-15,15],
[15,0,15],
[0,15,3],
[-15,0,15],
[0,15,11],
[15,0,3],
[0,-15,11],
[15,0,11],
[0,15,3],
[15,0,1],
]

Clyde_directions = [
[-30,0,2],
[0,-15,4],
```

```
[15,0,15],
[0,-15,3],
[-15,0,11],
[0,-15,7],
[15,0,3],
[0,-15,11],
[15,0,9],
]

pl = len(Pinky_directions)-1
bl = len(Blinky_directions)-1
il = len(Inky_directions)-1
cl = len(Clyde_directions)-1

# Call this function so the Pygame library can initialize itself
pygame.init()

# Create an 606x606 sized screen
screen = pygame.display.set_mode([606, 606])

# This is a list of 'sprites.' Each block in the program is
# added to this list. The list is managed by a class called 'RenderPlain.'


# Set the title of the window
pygame.display.set_caption('Pacman')

# Create a surface we can draw on
background = pygame.Surface(screen.get_size())

# Used for converting color maps and such
background = background.convert()

# Fill the screen with a black background
background.fill(black)



clock = pygame.time.Clock()

pygame.font.init()
font = pygame.font.Font("freesansbold.ttf", 24)

#default locations for Pacman and monstas
w = 303-16 #Width
p_h = (7*60)+19 #Pacman height
m_h = (4*60)+19 #Monster height
b_h = (3*60)+19 #Binky height
i_w = 303-16-32 #Inky width
c_w = 303+(32-16) #Clyde width

def startGame():

   all_sprites_list = pygame.sprite.RenderPlain()

   block_list = pygame.sprite.RenderPlain()

   monsta_list = pygame.sprite.RenderPlain()

   pacman_collide = pygame.sprite.RenderPlain()

   wall_list = setupRoomOne(all_sprites_list)

   gate = setupGate(all_sprites_list)
```

```
    p_turn = 0
    p_steps = 0

    b_turn = 0
    b_steps = 0

    i_turn = 0
    i_steps = 0

    c_turn = 0
    c_steps = 0


    # Create the player paddle object
    Pacman = Player( w, p_h, "images/Trollman.png" )
    all_sprites_list.add(Pacman)
    pacman_collide.add(Pacman)

    Blinky=Ghost( w, b_h, "images/Blinky.png" )
    monsta_list.add(Blinky)
    all_sprites_list.add(Blinky)

    Pinky=Ghost( w, m_h, "images/Pinky.png" )
    monsta_list.add(Pinky)
    all_sprites_list.add(Pinky)

    Inky=Ghost( i_w, m_h, "images/Inky.png" )
    monsta_list.add(Inky)
    all_sprites_list.add(Inky)

    Clyde=Ghost( c_w, m_h, "images/Clyde.png" )
    monsta_list.add(Clyde)
    all_sprites_list.add(Clyde)

    # Draw the grid
    for row in range(19):
        for column in range(19):
            if (row == 7 or row == 8) and (column == 8 or column == 9 or column ==
                continue
            else:
              block = Block(yellow, 4, 4)

              # Set a random location for the block
              block.rect.x = (30*column+6)+26
              block.rect.y = (30*row+6)+26

              b_collide = pygame.sprite.spritecollide(block, wall_list, False)
              p_collide = pygame.sprite.spritecollide(block, pacman_collide, Fals
              if b_collide:
                continue
              elif p_collide:
                continue
              else:
                # Add the block to the list of objects
                block_list.add(block)
                all_sprites_list.add(block)

    bll = len(block_list)

    score = 0

    done = False

    i = 0
```

```
452     while done == False:
453         # ALL EVENT PROCESSING SHOULD GO BELOW THIS COMMENT
454         for event in pygame.event.get():
455             if event.type == pygame.QUIT:
456                 done=True
457 
458             if event.type == pygame.KEYDOWN:
459                 if event.key == pygame.K_LEFT:
460                     Pacman.changespeed(-30,0)
461                 if event.key == pygame.K_RIGHT:
462                     Pacman.changespeed(30,0)
463                 if event.key == pygame.K_UP:
464                     Pacman.changespeed(0,-30)
465                 if event.key == pygame.K_DOWN:
466                     Pacman.changespeed(0,30)
467 
468             if event.type == pygame.KEYUP:
469                 if event.key == pygame.K_LEFT:
470                     Pacman.changespeed(30,0)
471                 if event.key == pygame.K_RIGHT:
472                     Pacman.changespeed(-30,0)
473                 if event.key == pygame.K_UP:
474                     Pacman.changespeed(0,30)
475                 if event.key == pygame.K_DOWN:
476                     Pacman.changespeed(0,-30)
477 
478         # ALL EVENT PROCESSING SHOULD GO ABOVE THIS COMMENT
479 
480         # ALL GAME LOGIC SHOULD GO BELOW THIS COMMENT
481         Pacman.update(wall_list,gate)
482 
483         returned = Pinky.changespeed(Pinky_directions,False,p_turn,p_steps,pl)
484         p_turn = returned[0]
485         p_steps = returned[1]
486         Pinky.changespeed(Pinky_directions,False,p_turn,p_steps,pl)
487         Pinky.update(wall_list,False)
488 
489         returned = Blinky.changespeed(Blinky_directions,False,b_turn,b_steps,bl)
490         b_turn = returned[0]
491         b_steps = returned[1]
492         Blinky.changespeed(Blinky_directions,False,b_turn,b_steps,bl)
493         Blinky.update(wall_list,False)
494 
495         returned = Inky.changespeed(Inky_directions,False,i_turn,i_steps,il)
496         i_turn = returned[0]
497         i_steps = returned[1]
498         Inky.changespeed(Inky_directions,False,i_turn,i_steps,il)
499         Inky.update(wall_list,False)
500 
501         returned = Clyde.changespeed(Clyde_directions,"clyde",c_turn,c_steps,cl)
502         c_turn = returned[0]
503         c_steps = returned[1]
504         Clyde.changespeed(Clyde_directions,"clyde",c_turn,c_steps,cl)
505         Clyde.update(wall_list,False)
506 
507         # See if the Pacman block has collided with anything.
508         blocks_hit_list = pygame.sprite.spritecollide(Pacman, block_list, True)
509 
510         # Check the list of collisions.
511         if len(blocks_hit_list) > 0:
512             score +=len(blocks_hit_list)
513 
514         # ALL GAME LOGIC SHOULD GO ABOVE THIS COMMENT
515 
```

```
        wall_list.draw(screen)
        gate.draw(screen)
        all_sprites_list.draw(screen)
        monsta_list.draw(screen)

        text=font.render("Score: "+str(score)+"/"+str(bll), True, red)
        screen.blit(text, [10, 10])

        if score == bll:
          doNext("Congratulations, you won!",145,all_sprites_list,block_list,mons

        monsta_hit_list = pygame.sprite.spritecollide(Pacman, monsta_list, False)

        if monsta_hit_list:
          doNext("Game Over",235,all_sprites_list,block_list,monsta_list,pacman_c

        # ALL CODE TO DRAW SHOULD GO ABOVE THIS COMMENT

        pygame.display.flip()

        clock.tick(10)

def doNext(message,left,all_sprites_list,block_list,monsta_list,pacman_collide,
  while True:
        # ALL EVENT PROCESSING SHOULD GO BELOW THIS COMMENT
        for event in pygame.event.get():
          if event.type == pygame.QUIT:
            pygame.quit()
          if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_ESCAPE:
              pygame.quit()
            if event.key == pygame.K_RETURN:
              del all_sprites_list
              del block_list
              del monsta_list
              del pacman_collide
              del wall_list
              del gate
              startGame()

        #Grey background
        w = pygame.Surface((400,200))  # the size of your rect
        w.set_alpha(10)                # alpha level
        w.fill((128,128,128))           # this fills the entire surface
        screen.blit(w, (100,200))    # (0,0) are the top-left coordinates

        #Won or lost
        text1=font.render(message, True, white)
        screen.blit(text1, [left, 233])

        text2=font.render("To play again, press ENTER.", True, white)
        screen.blit(text2, [135, 303])
        text3=font.render("To quit, press ESCAPE.", True, white)
        screen.blit(text3, [165, 333])

        pygame.display.flip()

        clock.tick(10)

startGame()

pygame.quit()
```

OUTPUT: