**UCS2504- Foundations of Artificial Intelligence (TCP)**

**Project:**

**WHACK-A-MOLE**

KEERTHANA.G.S(3122225001059)

KEERTHANA.S(3122225001061)



**Department of Computer Science and Engineering**

**Sri Sivasubramaniya Nadar College of Engineering**

**Kalavakkam – 603110**

**Introduction**

The Whack-a-Mole game is a digital adaptation of the classic arcade game. Players compete with an AI to hit moles appearing on a grid. The player scores by clicking on moles that appear randomly. This game uses Python's Pygame library for graphics and interaction, providing a time-bound, score-based competition.

**1. Requirements**

* Python 3 and Pygame library installed on your system.
* Image and sound files:
  + Images for the moles (player, AI, golden mole) and hammer.
  + Sound file for the hit effect.

**2. Game Design**

The game consists of a 4x4 grid where moles appear at random intervals. The player has a limited time to score as many points as possible by hitting moles with a hammer cursor. Meanwhile, an AI opponent also competes by hitting moles on the screen.

* Grid Layout: The grid contains 16 cells, each capable of displaying a mole.
* Mole Types:
  + Player Mole
  + AI Mole
  + Golden Mole (special, rare appearance)

**3. Core Features**

* Player vs AI gameplay mechanics
* Score tracking system
* Time-limited matches (20 seconds)
* Special golden mole bonus feature
* Custom cursor with hammer graphic
* Sound effects for successful hits
* Visual feedback for hits

**4. Technology Stack**

* Primary Language: Python
* Game Framework: Pygame
* Additional Libraries:
  + random (for randomization)
  + time (for timing mechanisms)

**5. Mole Types**

1. Player Mole
   * Scoring: +1 point when hit by player
2. AI Mole
   * Scoring: -1 point when hit by player
3. Golden Mole
   * Scoring: +3 points

**6.Code:**

Game Initialization and Setup

6.1 Core Configurations

SCREEN\_WIDTH = 600

SCREEN\_HEIGHT = 700

GRID\_SIZE = 4

CELL\_SIZE = 150

MOLE\_SIZE = 100

GRID\_SPACING = 10

FPS = 30

MOLE\_TIME = 1

GAME\_TIME = 20

* Screen dimensions optimize gameplay area
* 4x4 grid provides balanced difficulty
* Cell and mole sizes ensure clear visibility
* 20-second game duration maintains engagement

6.2 Asset Management

PLAYER\_MOLE\_IMAGE\_PATH = 'mole1.png'

AI\_MOLE\_IMAGE\_PATH = 'mole2.png'

HAMMER\_IMAGE\_PATH = 'hammer.png'

GOLDEN\_MOLE\_IMAGE\_PATH = 'golden\_mole.png'

hit\_sound = pygame.mixer.Sound('sound-1-167181.mp3')

ai\_hit\_sound = pygame.mixer.Sound('sound-1-167181.mp3')

* Separate sprites for player and AI moles enhance visual distinction
* Custom hammer cursor improves user interaction
* Sound effects provide immediate feedback

6.3 Grid System Implementation

def draw\_grid():

for row in range(GRID\_SIZE):

for col in range(GRID\_SIZE):

x = col \* (CELL\_SIZE + GRID\_SPACING)

y = row \* (CELL\_SIZE + GRID\_SPACING)

pygame.draw.rect(screen, HOLE\_COLOR, (x, y, CELL\_SIZE, CELL\_SIZE))

* Iterative grid creation ensures consistent spacing
* Built-in spacing prevents visual crowding
* Brown hole color provides contrast with green background

6.4 Mole Drawing System

def draw\_mole(mole\_position, mole\_type):

row, col = mole\_position

x = col \* (CELL\_SIZE + GRID\_SPACING) + (CELL\_SIZE - MOLE\_SIZE) // 2

y = row \* (CELL\_SIZE + GRID\_SPACING) + (CELL\_SIZE - MOLE\_SIZE) // 2

if mole\_type == "player":

screen.blit(player\_mole\_image, (x, y))

elif mole\_type == "ai":

screen.blit(ai\_mole\_image, (x, y))

elif mole\_type == "golden":

screen.blit(golden\_mole\_image, (x, y))

* Centralized positioning within cells
* Type-based sprite selection
* Support for special golden mole

6.5 Input Processing

def get\_cell\_from\_mouse\_pos(pos):

x, y = pos

col = x // (CELL\_SIZE + GRID\_SPACING)

row = y // (CELL\_SIZE + GRID\_SPACING)

return row, col

* Efficient mouse position to grid conversion
* Accounts for grid spacing in calculations
* Returns normalized grid coordinates

6.6 Mole Spawn Logic

if current\_time - last\_mole\_time > MOLE\_TIME and player\_moles\_appeared + ai\_moles\_appeared < max\_moles:

mole\_position = (random.randint(0, GRID\_SIZE - 1), random.randint(0, GRID\_SIZE - 1))

if 9.5 <= elapsed\_time <= 10.5 and not golden\_mole\_appeared:

mole\_type = "golden"

*# Golden mole logic*

else:

if player\_moles\_appeared == ai\_moles\_appeared:

mole\_type = random.choice(["player", "ai"])

elif player\_moles\_appeared > ai\_moles\_appeared:

mole\_type = "ai"

else:

mole\_type = "player"

* Time-based spawn system
* Balanced distribution between player and AI moles
* Special golden mole timing implementation
* Maximum mole limit prevents endless gameplay

6.7 Scoring System

if clicked\_cell == mole\_position:

if mole\_type == "player":

player\_score += 1

elif mole\_type == "ai":

player\_score -= 1

elif mole\_type == "golden":

player\_score += 3

* Type-based point allocation
* Penalty system for incorrect hits
* Bonus points for golden mole

6.8 AI Implementation

def ai\_decide\_hit(mole\_position, ai\_last\_hit\_time, ai\_reaction\_time, mole\_visible):

current\_time = time.time()

if mole\_visible and (current\_time - ai\_last\_hit\_time >= ai\_reaction\_time):

return True

return False

* Time-based decision making
* Random reaction time (0.5-1.5 seconds)
* Visibility checking for valid hits

6.9 Visual Effects and Feedback

def draw\_hit\_effect(mole\_position):

row, col = mole\_position

x = col \* (CELL\_SIZE + GRID\_SPACING)

y = row \* (CELL\_SIZE + GRID\_SPACING)

pygame.draw.rect(screen, (255, 0, 0), (x, y, CELL\_SIZE, CELL\_SIZE))

* Visual feedback for successful hits
* Temporary red flash effect
* Grid-aligned positioning

6.10 Score and Timer display

score\_text = font.render(f"My Score: {player\_score}", True, (0, 0, 0))

ai\_score\_text = font.render(f"AI Score: {ai\_score}", True, (0, 0, 0))

time\_text = font.render(f"Time: {int(remaining\_time)}s", True, (0, 0, 0))

* Real-time score tracking
* Countdown timer display

**CONCLUSION:**

* The Whack-a-Mole game project demonstrates the effective use of basic AI-driven heuristics and interactive gameplay mechanics.
* By incorporating dynamic elements, such as randomized mole appearances, golden moles with bonus points, and AI reaction timing, the game provides an engaging experience that balances competition between the player and the AI.
* Additionally, visual effects, scoring feedback, and sound effects contribute to a well-rounded and immersive game experience.
* This project not only serves as a fun and interactive game but also showcases foundational principles of AI behavior, real-time interaction, and basic game development techniques in Python with Pygame.