**Escape The Grid With A\***

ליאב יהושע 314957499  
רוני שררה 207642786  
מוריאל גניש 206865123

לינקים:

מצגת:

<https://www.youtube.com/watch?v=SWJ-TBnytzQ&ab_channel=%D7%94%D7%A7%D7%9C%D7%98%D7%95%D7%AA%D7%9E%D7%93%D7%A2%D7%99%D7%94%D7%9E%D7%97%D7%A9%D7%91>

הצגת מהלך המשחק: <https://www.youtube.com/watch?v=fgX6HWL9p5s&ab_channel=%D7%94%D7%A7%D7%9C%D7%98%D7%95%D7%AA%D7%9E%D7%93%D7%A2%D7%99%D7%94%D7%9E%D7%97%D7%A9%D7%91>

Github:

<https://github.com/liav-yehoshua/Escape-The-Grid.git>

סיכום כללי:

**תיאור הבעיה- מטרת הפרויקט הייתה לפתח משחק מבוך דינמי לשני שחקנים, שבו כל שחקן מנסה להגיע אל נקודת היעד האישית שלו. במהלך המשחק, השחקנים יכולים להניח מחסומים (בולי עץ) כדי להקשות על התקדמות היריב. המשחק משלב אלגוריתם חיפוש** (A\*) **יחד עם מודל למידת מכונה** (MLPRegressor) **שמטרתם לסייע בקבלת החלטות אסטרטגיות במהלך המשחק.**

מהלך משחק: **המשחק מחולק לשני שלבים חוזרים-  
1. שלב המחסומים – כל שחקן מניח 5 בולי עץ במטרה לחסום את היריב.  
2. שלב התזוזה – כל שחקן נע 5 צעדים לעבר היעד שלו, בהתאם למסלול המחושב.**

**דרך הפתרון-**

* **תכנון תנועה:  
  כל שחקן משתמש באלגוריתם A\* כדי לחשב את המסלול הקצר ביותר לנקודת היעד, תוך התחשבות בבולי העץ שכבר נמצאים על הלוח.**
* **קבלת החלטות אסטרטגיות:  
  כל שחקן מפעיל מודל למידת מכונה מסוג MLPRegressor, אשר לומד מתוך ניסיון העבר מתי נכון להניח בולי עץ (כדי להפריע ליריב) ומתי עדיף פשוט להתקדם לעבר היעד.**

**אלגוריתמים -**

אלגוריתם A\*-

* **משמש למציאת הדרך היעילה ביותר מנקודת המוצא של השחקן אל היעד שלו.**
* **האלגוריתם מסתמך על פונקציית הערכה שמשקללת את המרחק שעבר עד כה עם המרחק שנותר.**

למידת מכונה- MLPRegressor-

* **כל שחקן מפעיל מודל מסוג** Multi-Layer Perceptron **כדי לקבוע את הפעולה האופטימלית – האם לנוע או להציב מחסום.**
* **המודל מתאמן על בסיס נתונים ממשחקים קודמים ומשפר את הביצועים לאורך זמן.**
* **במהלך כל משחק, נאספים נתונים חדשים והמודל מעודכן מחדש בהתאם.**

הקוד המלא:

*import pygame*

*import sys*

*import json*

*import random*

*import numpy as np*

*from queue import PriorityQueue*

*from sklearn.neural\_network import MLPRegressor*

*import os*

*BASE\_DIR = os.path.dirname(os.path.abspath(\_\_file\_\_))*

*# Initialize Pygame*

*pygame.init()*

*# ─── Modern UI Fonts ──────────────────────────────────────────────────────────*

*FONT\_REGULAR = pygame.font.SysFont('Segoe UI', 20)*

*FONT\_TITLE   = pygame.font.SysFont('Segoe UI', 48)*

*FONT\_BUTTON  = pygame.font.SysFont('Segoe UI', 36)*

*FONT\_ARROW   = pygame.font.SysFont('Segoe UI', 48)*

*# ─── Modern Color Palette ─────────────────────────────────────────────────────*

*COLOR\_BG           = (34, 34, 34)*

*COLOR\_GRID\_LIGHT   = (60, 60, 60)*

*COLOR\_GRID\_DARK    = (50, 50, 50)*

*COLOR\_ACCENT       = (0, 204, 153)*

*COLOR\_BUTTON       = (45, 45, 45)*

*COLOR\_BUTTON\_HOVER = (0, 204, 153)*

*COLOR\_TEXT         = (230, 230, 230)*

*COLOR\_LOG          = (200, 155, 100)*

*# ─── Screen setup ─────────────────────────────────────────────────────────────*

*WIDTH, HEIGHT = pygame.display.Info().current\_w, pygame.display.Info().current\_h*

*GRID\_SIZE     = 10*

*CELL\_SIZE     = min(WIDTH // GRID\_SIZE, HEIGHT // GRID\_SIZE)*

*screen        = pygame.display.set\_mode((WIDTH, HEIGHT), pygame.FULLSCREEN)*

*pygame.display.set\_caption("Escape The Grid")*

*# ─── Load and scale images ─────────────────────────────────────────────────────*

*door\_image = pygame.transform.scale(*

*pygame.image.load(os.path.join(BASE\_DIR, 'door.png')), (CELL\_SIZE, CELL\_SIZE)*

*)*

*# Animal image filenames and names*

*ANIMAL\_IMAGES = [*

*("fox.png", "Fox"),*

*("lion.png", "Lion"),*

*("zebra.png", "Zebra"),*

*("penguin.png", "Penguin")*

*]*

*player1\_images = [*

*pygame.transform.scale(*

*pygame.image.load(os.path.join(BASE\_DIR, fname)), (CELL\_SIZE, CELL\_SIZE)*

*) for fname, \_ in ANIMAL\_IMAGES*

*]*

*player2\_images = list(player1\_images)*

*player1\_img, player2\_img = player1\_images[0], player2\_images[1]*

*# Store animal names for each player*

*player1\_animal = ANIMAL\_IMAGES[0][1]*

*player2\_animal = ANIMAL\_IMAGES[1][1]*

*log\_image\_raw = pygame.image.load(os.path.join(BASE\_DIR, 'log.png'))*

*# ─── A\* Pathfinding Functions ─────────────────────────────────────────────────*

*def* ***heuristic****(a, b):*

*return abs(b[0]-a[0]) + abs(b[1]-a[1])*

*def* ***get\_neighbors****(x, y, logs, can\_hop, has\_jumped):*

*neighbors = []*

*for dx, dy in [(0,1),(1,0),(0,-1),(-1,0)]:*

*nx, ny = x+dx, y+dy*

*if 0 <= nx < GRID\_SIZE and 0 <= ny < GRID\_SIZE:*

*blocked = any(*

*log.x<=nx<log.x+(2 if log.horizontal else 1) and*

*log.y<=ny<log.y+(2 if not log.horizontal else 1)*

*for log in logs*

*)*

*if not blocked:*

*neighbors.append((nx, ny))*

*elif can\_hop and not has\_jumped:*

*hx, hy = nx+dx, ny+dy*

*if 0<=hx<GRID\_SIZE and 0<=hy<GRID\_SIZE and not any(*

*log.x<=hx<log.x+(2 if log.horizontal else 1) and*

*log.y<=hy<log.y+(2 if not log.horizontal else 1)*

*for log in logs*

*):*

*neighbors.append((hx, hy))*

*return neighbors*

*def* ***a\_star****(start, goal, logs, can\_hop, has\_jumped, opponent\_pos=None):*

*frontier = PriorityQueue()*

*frontier.put((0, start))*

*came\_from = {start: None}*

*cost = {start: 0}*

*while not frontier.empty():*

*\_, current = frontier.get()*

*if current == goal:*

*path = []*

*while current:*

*path.append(current)*

*current = came\_from[current]*

*return list(reversed(path))*

*for nxt in get\_neighbors(current[0], current[1], logs, can\_hop, has\_jumped):*

*new\_cost = cost[current] + 1*

*# Penalty for being adjacent to opponent*

*if opponent\_pos:*

*if abs(nxt[0] - opponent\_pos[0]) <= 1 and abs(nxt[1] - opponent\_pos[1]) <= 1:*

*new\_cost += 5  # Large penalty for being near opponent*

*# Penalty for being adjacent to a log*

*for log in logs:*

*log\_cells = []*

*if log.horizontal:*

*log\_cells = [(log.x, log.y), (log.x+1, log.y)]*

*else:*

*log\_cells = [(log.x, log.y), (log.x, log.y+1)]*

*for lx, ly in log\_cells:*

*if abs(nxt[0] - lx) <= 1 and abs(nxt[1] - ly) <= 1:*

*new\_cost += 2  # Penalty for being near a log*

*break*

*if nxt not in cost or new\_cost < cost[nxt]:*

*cost[nxt] = new\_cost*

*priority = new\_cost + heuristic(goal, nxt)*

*frontier.put((priority, nxt))*

*came\_from[nxt] = current*

*return None*

*# ─── Player & Log Classes ─────────────────────────────────────────────────────*

*class Player:*

*def* ***\_\_init\_\_****(self, x, y, image, goal):*

*self.x,self.y   = x,y*

*self.image      = image*

*self.goal       = goal*

*self.logs       = 5*

*self.path       = []*

*self.model      = MLPRegressor(hidden\_layer\_sizes=(50,50), max\_iter=1000, random\_state=42)*

*self.train\_data = []*

*self.train\_model()*

*self.logs\_placed = 0*

*self.moves\_left  = 5*

*self.has\_jumped  = False*

*def* ***draw****(self):*

*screen.blit(self.image, (GRID\_TOPLEFT\_X + self.x\*CELL\_SIZE, GRID\_TOPLEFT\_Y + self.y\*CELL\_SIZE))*

*def* ***move****(self, logs=None, opponent=None):*

*# Only follow the existing path, do NOT recalculate*

*if self.path and len(self.path) > 1 and self.moves\_left > 0:*

*nx,ny = self.path[1]*

*if abs(nx-self.x)+abs(ny-self.y)>1:*

*self.has\_jumped=True*

*self.x,self.y = nx,ny*

*self.path = self.path[1:]*

*self.moves\_left -= 1*

*def* ***train\_model****(self):*

*if len(self.train\_data)>100:*

*X,y = zip(\*self.train\_data)*

*self.model.fit(X,y)*

*else:*

*X=np.random.rand(1000,7)*

*y=np.random.rand(1000)*

*self.model.fit(X,y)*

*def* ***load\_training\_data****(self,data):*

*self.train\_data.extend(data)*

*self.train\_model()*

*def* ***decide\_action****(self, opp, logs):*

*md = abs(self.x-self.goal[0])+abs(self.y-self.goal[1])*

*od = abs(opp.x-opp.goal[0])+abs(opp.y-opp.goal[1])*

*X = np.array([[self.x,self.y,opp.x,opp.y,self.logs,md,od]])*

*p = self.model.predict(X)[0]*

*if p>0.7 and self.logs>0: action='place\_log'*

*elif p>0.3: action='hop'*

*else: action='move'*

*self.train\_data.append((X[0],1 if action=='place\_log' else (0.5 if action=='hop' else 0)))*

*return action*

*LOG\_BIG\_SIZE = (int(CELL\_SIZE \* 2.5), int(CELL\_SIZE \* 2.5))*

*log\_image\_big = pygame.transform.scale(log\_image\_raw, LOG\_BIG\_SIZE)*

*class Log:*

*def* ***\_\_init\_\_****(self,x,y,hor):*

*self.x,self.y,self.horizontal = x,y,hor*

*def* ***draw****(self):*

*offset\_x = GRID\_TOPLEFT\_X + self.x \* CELL\_SIZE + (CELL\_SIZE - LOG\_BIG\_SIZE[0]) // 2*

*offset\_y = GRID\_TOPLEFT\_Y + self.y \* CELL\_SIZE + (CELL\_SIZE - LOG\_BIG\_SIZE[1]) // 2*

*screen.blit(log\_image\_big, (offset\_x, offset\_y))*

*# ─── End Screen ────────────────────────────────────────────────────────────────*

*def* ***end\_screen****(winner):*

*screen.fill(COLOR\_BG)*

*if winner == 'Player 1':*

*text = f"The {player1\_animal} wins!"*

*elif winner == 'Player 2':*

*text = f"The {player2\_animal} wins!"*

*else:*

*text = "Draw!"*

*surf=FONT\_TITLE.render(text,True,COLOR\_ACCENT)*

*rect=surf.get\_rect(center=(WIDTH//2,HEIGHT//2-50))*

*screen.blit(surf,rect)*

*btn=pygame.Rect(WIDTH//2-100,HEIGHT//2+20,200,50)*

*while True:*

*mx,my=pygame.mouse.get\_pos()*

*hover=btn.collidepoint(mx,my)*

*draw\_button("Main Menu",btn,hover)*

*pygame.display.flip()*

*for e in pygame.event.get():*

*if e.type==pygame.QUIT: pygame.quit();sys.exit()*

*if e.type==pygame.MOUSEBUTTONDOWN and hover: return*

*# ─── Initialization ────────────────────────────────────────────────────────────*

*def* ***initialize\_players****():*

*global player1,player2,logs*

*player1=Player(0,0,player1\_img,(GRID\_SIZE-1,GRID\_SIZE-1))*

*player2=Player(GRID\_SIZE-1,GRID\_SIZE-1,player2\_img,(0,0))*

*logs=[]*

*player1.logs\_placed=player2.logs\_placed=0*

*player1.moves\_left=player2.moves\_left=5*

*player1.has\_jumped=player2.has\_jumped=False*

*# ─── Drawing Utilities ─────────────────────────────────────────────────────────*

*# Calculate grid topleft offset for centering*

*INFO\_PANEL\_WIDTH = 320*

*INFO\_PANEL\_HEIGHT = 200*

*INFO\_PANEL\_Y = (HEIGHT - INFO\_PANEL\_HEIGHT) // 2*

*INFO\_PANEL\_LEFT\_X = 60*

*# Calculate grid position first*

*GRID\_PIXEL\_SIZE = GRID\_SIZE \* CELL\_SIZE*

*GRID\_TOPLEFT\_X = INFO\_PANEL\_LEFT\_X + INFO\_PANEL\_WIDTH + 60*

*GRID\_TOPLEFT\_Y = (HEIGHT - GRID\_PIXEL\_SIZE) // 2*

*# Now you can safely use GRID\_TOPLEFT\_X*

*INFO\_PANEL\_RIGHT\_X = GRID\_TOPLEFT\_X + GRID\_PIXEL\_SIZE + 60*

*def* ***draw\_grid****():*

*shadow\_offset = 6  # pixels to offset the shadow*

*for x in range(GRID\_SIZE):*

*for y in range(GRID\_SIZE):*

*# Shadow rectangle (drawn first, slightly offset)*

*shadow\_rect = pygame.Rect(*

*GRID\_TOPLEFT\_X + x\*CELL\_SIZE + shadow\_offset,*

*GRID\_TOPLEFT\_Y + y\*CELL\_SIZE + shadow\_offset,*

*CELL\_SIZE,*

*CELL\_SIZE*

*)*

*pygame.draw.rect(screen, (20, 20, 20), shadow\_rect, border\_radius=12)*

*# Main cell rectangle (drawn on top)*

*cell\_rect = pygame.Rect(GRID\_TOPLEFT\_X + x\*CELL\_SIZE, GRID\_TOPLEFT\_Y + y\*CELL\_SIZE, CELL\_SIZE, CELL\_SIZE)*

*c = COLOR\_GRID\_LIGHT if (x+y)%2==0 else COLOR\_GRID\_DARK*

*pygame.draw.rect(screen, c, cell\_rect, border\_radius=12)*

*# Optional: Add a simple vertical gradient for extra depth*

*gradient\_surface = pygame.Surface((CELL\_SIZE, CELL\_SIZE), pygame.SRCALPHA)*

*for i in range(CELL\_SIZE):*

*alpha = int(40 \* (1 - i / CELL\_SIZE))  # fade out*

*pygame.draw.line(gradient\_surface, (255,255,255,alpha), (0,i), (CELL\_SIZE,i))*

*screen.blit(gradient\_surface, (GRID\_TOPLEFT\_X + x\*CELL\_SIZE, GRID\_TOPLEFT\_Y + y\*CELL\_SIZE))*

*# Accent border*

*pygame.draw.rect(screen, COLOR\_ACCENT, pygame.Rect(GRID\_TOPLEFT\_X, GRID\_TOPLEFT\_Y, GRID\_PIXEL\_SIZE, GRID\_PIXEL\_SIZE), width=4, border\_radius=16)*

*def* ***draw\_button****(text,rect,hovered=False):*

*col = COLOR\_BUTTON\_HOVER if hovered else COLOR\_BUTTON*

*pygame.draw.rect(screen,col,rect,border\_radius=8)*

*txt=FONT\_BUTTON.render(text,True,COLOR\_TEXT)*

*screen.blit(txt,txt.get\_rect(center=rect.center))*

*def* ***draw\_arrows****(rect):*

*l=FONT\_ARROW.render("<",True,COLOR\_ACCENT)*

*r=FONT\_ARROW.render(">",True,COLOR\_ACCENT)*

*lr=l.get\_rect(midright=(rect.left-10,rect.centery))*

*rr=r.get\_rect(midleft=(rect.right+10,rect.centery))*

*screen.blit(l,lr);screen.blit(r,rr)*

*return lr,rr*

*def* ***draw\_text****(txt,pos,font):*

*surf=font.render(txt,True,COLOR\_TEXT)*

*screen.blit(surf,surf.get\_rect(topleft=pos))*

*# ─── Main Draw Function ─────────────────────────────────────────────────────────*

*def* ***draw****(mode):*

*screen.fill(COLOR\_BG)*

*draw\_grid()*

*for log in logs: log.draw()*

*screen.blit(door\_image,(GRID\_TOPLEFT\_X, GRID\_TOPLEFT\_Y))*

*screen.blit(door\_image,(GRID\_TOPLEFT\_X + (GRID\_SIZE-1)\*CELL\_SIZE, GRID\_TOPLEFT\_Y + (GRID\_SIZE-1)\*CELL\_SIZE))*

*player1.draw(); player2.draw()*

*# Stylish A\* path lines with distinct colors*

*for pl, color, glow in [*

*(player1, (0, 120, 255), (100, 180, 255, 90)),  # Blue for Player 1*

*(player2, (220, 40, 40), (255, 120, 120, 90))   # Red for Player 2*

*]:*

*if pl.path:*

*# Draw the next 5 steps of the path*

*for i in range(len(pl.path)-1):*

*sx,sy=pl.path[i]; ex,ey=pl.path[i+1]*

*sp=(GRID\_TOPLEFT\_X + sx\*CELL\_SIZE+CELL\_SIZE//2, GRID\_TOPLEFT\_Y + sy\*CELL\_SIZE+CELL\_SIZE//2)*

*ep=(GRID\_TOPLEFT\_X + ex\*CELL\_SIZE+CELL\_SIZE//2, GRID\_TOPLEFT\_Y + ey\*CELL\_SIZE+CELL\_SIZE//2)*

*# Glow effect (thicker, semi-transparent line)*

*glow\_surface = pygame.Surface((WIDTH, HEIGHT), pygame.SRCALPHA)*

*pygame.draw.line(glow\_surface, glow, sp, ep, 10)*

*screen.blit(glow\_surface, (0,0))*

*# Main colored line*

*pygame.draw.line(screen, color, sp, ep, 4)*

*# Info panels*

*if mode == 'place\_log':*

*# Log phase panel (left)*

*panel\_rect = pygame.Rect(INFO\_PANEL\_LEFT\_X, INFO\_PANEL\_Y, INFO\_PANEL\_WIDTH, INFO\_PANEL\_HEIGHT)*

*pygame.draw.rect(screen, COLOR\_BUTTON, panel\_rect, border\_radius=16)*

*y = INFO\_PANEL\_Y + 18*

*phase\_text = "Log Phase"*

*phase\_surf = FONT\_BUTTON.render(phase\_text, True, COLOR\_ACCENT)*

*phase\_rect = phase\_surf.get\_rect(midtop=(INFO\_PANEL\_LEFT\_X + INFO\_PANEL\_WIDTH//2, y))*

*screen.blit(phase\_surf, phase\_rect)*

*y += phase\_rect.height + 18*

*draw\_text(f"P1 Logs: {player1.logs}", (INFO\_PANEL\_LEFT\_X + 30, y), FONT\_REGULAR)*

*y += 40*

*draw\_text(f"P2 Logs: {player2.logs}", (INFO\_PANEL\_LEFT\_X + 30, y), FONT\_REGULAR)*

*elif mode == 'move':*

*# Move phase panel (right)*

*panel\_rect = pygame.Rect(INFO\_PANEL\_RIGHT\_X, INFO\_PANEL\_Y, INFO\_PANEL\_WIDTH, INFO\_PANEL\_HEIGHT)*

*pygame.draw.rect(screen, COLOR\_BUTTON, panel\_rect, border\_radius=16)*

*y = INFO\_PANEL\_Y + 18*

*phase\_text = "Move Phase"*

*phase\_surf = FONT\_BUTTON.render(phase\_text, True, COLOR\_ACCENT)*

*phase\_rect = phase\_surf.get\_rect(midtop=(INFO\_PANEL\_RIGHT\_X + INFO\_PANEL\_WIDTH//2, y))*

*screen.blit(phase\_surf, phase\_rect)*

*y += phase\_rect.height + 18*

*draw\_text(f"P1 Moves: {player1.moves\_left}", (INFO\_PANEL\_RIGHT\_X + 30, y), FONT\_REGULAR)*

*y += 40*

*draw\_text(f"P2 Moves: {player2.moves\_left}", (INFO\_PANEL\_RIGHT\_X + 30, y), FONT\_REGULAR)*

*# Back button in bottom left corner*

*back=pygame.Rect(30, HEIGHT-70, 140, 40)*

*draw\_button("Back", back, back.collidepoint(pygame.mouse.get\_pos()))*

*pygame.display.flip()*

*return back*

*# ─── Logs & Turns ───────────────────────────────────────────────────────────────*

*def* ***logs\_overlap****(nl,elogs):*

*for log in elogs:*

*if nl.horizontal==log.horizontal:*

*if nl.horizontal and nl.y==log.y and nl.x<log.x+2 and log.x<nl.x+2: return True*

*if not nl.horizontal and nl.x==log.x and nl.y<log.y+2 and log.y<nl.y+2: return True*

*else:*

*if nl.horizontal and ((log.x<=nl.x<log.x+1 or log.x<=nl.x+1<log.x+1) and (nl.y<=log.y<nl.y+1 or nl.y<=log.y+1<nl.y+1)): return True*

*if not nl.horizontal and ((nl.x<=log.x<nl.x+1 or nl.x<=log.x+1<nl.x+1) and (log.y<=nl.y<log.y+1 or log.y<=nl.y+1<log.y+1)): return True*

*return False*

*def* ***place\_log****(p,o):*

*if p.logs>0:*

*for \_ in range(50):*

*horiz=random.choice([True,False])*

*x=random.randint(0,GRID\_SIZE-2) if horiz else random.randint(0,GRID\_SIZE-1)*

*y=random.randint(0,GRID\_SIZE-1) if horiz else random.randint(0,GRID\_SIZE-2)*

*nl=Log(x,y,horiz)*

*if not logs\_overlap(nl,logs) and (x,y) not in [(p.x,p.y),(o.x,o.y),(0,0),(GRID\_SIZE-1,GRID\_SIZE-1)]:*

*tmp=logs+[nl]*

*if a\_star((p.x,p.y),p.goal,tmp,True,False) and a\_star((o.x,o.y),o.goal,tmp,True,False):*

*logs.append(nl); p.logs-=1; return True*

*return False*

*def* ***player\_turn****(p,o,mode):*

*if mode=='place\_log' and p.logs\_placed<5 and place\_log(p,o): p.logs\_placed+=1*

*elif mode=='move' and p.moves\_left>0:*

*p.move(logs=logs, opponent=o)*

*# If path is blocked, player does not move*

*def* ***check\_win****(p): return (p.x,p.y)==p.goal*

*def* ***clear\_logs****(): global logs; logs=[]*

*def* ***play\_game****():*

*turn,mode,winner=0,'place\_log',None*

*clock=pygame.time.Clock(); exit\_menu=False*

*while turn<1000 and not exit\_menu and winner is None:*

*back=draw(mode)*

*for e in pygame.event.get():*

*if e.type==pygame.QUIT: pygame.quit();sys.exit()*

*if e.type==pygame.MOUSEBUTTONDOWN and back.collidepoint(e.pos): exit\_menu=True*

*if exit\_menu: break*

*if mode=='place\_log' and player1.logs\_placed==5 and player2.logs\_placed==5:*

*mode='move'*

*player1.path = a\_star((player1.x, player1.y), player1.goal, logs, True, player1.has\_jumped, (player2.x, player2.y))*

*player2.path = a\_star((player2.x, player2.y), player2.goal, logs, True, player2.has\_jumped, (player1.x, player1.y))*

*player1.moves\_left=player2.moves\_left=5*

*player1.has\_jumped=player2.has\_jumped=False*

*if turn%2==0: player\_turn(player1,player2,mode)*

*else:        player\_turn(player2,player1,mode)*

*if mode=='move':*

*if check\_win(player1): winner='Player 1'*

*if check\_win(player2): winner='Player 2'*

*if player1.moves\_left==0 and player2.moves\_left==0:*

*mode='place\_log'; clear\_logs()*

*player1.logs=player2.logs=5; player1.logs\_placed=player2.logs\_placed=0*

*turn+=1; clock.tick(2)*

*if exit\_menu: return 'menu'*

*end\_screen(winner)*

*return winner*

*def* ***save\_game\_data****(p1,p2,lg,win):*

*gd={'player1':{'x':p1.x,'y':p1.y,'logs':p1.logs},*

*'player2':{'x':p2.x,'y':p2.y,'logs':p2.logs},*

*'logs':[{'x':l.x,'y':l.y,'horizontal':l.horizontal} for l in lg],*

*'winner':win,'training\_data':{'player1':p1.train\_data,'player2':p2.train\_data}}*

*try: data=json.load(open('game\_data.json'))*

*except: data=[]*

*data.append(gd)*

*json.dump(data,open('game\_data.json','w'),indent=4)*

*def* ***load\_training\_data****():*

*try: data=json.load(open('game\_data.json'))*

*except: data=[]*

*p1=[]; p2=[]*

*for g in data: p1.extend(g['training\_data']['player1']); p2.extend(g['training\_data']['player2'])*

*return p1,p2*

*def* ***choose\_player\_images****():*

*global player1\_img, player2\_img, player1\_animal, player2\_animal*

*chosen1,chosen2=False,False*

*while not(chosen1 and chosen2):*

*screen.fill(COLOR\_BG)*

*for i,img in enumerate(player1\_images):*

*screen.blit(img,(WIDTH//4-CELL\_SIZE//2,HEIGHT//4+i\*(CELL\_SIZE+10)))*

*for i,img in enumerate(player2\_images):*

*screen.blit(img,(WIDTH\*3//4-CELL\_SIZE//2,HEIGHT//4+i\*(CELL\_SIZE+10)))*

*title1=FONT\_TITLE.render("Choose Player 1",True,COLOR\_TEXT)*

*screen.blit(title1,title1.get\_rect(center=(WIDTH//4,HEIGHT//4-40)))*

*title2=FONT\_TITLE.render("Choose Player 2",True,COLOR\_TEXT)*

*screen.blit(title2,title2.get\_rect(center=(WIDTH\*3//4,HEIGHT//4-40)))*

*pygame.display.flip()*

*for e in pygame.event.get():*

*if e.type==pygame.QUIT:pygame.quit();sys.exit()*

*if e.type==pygame.MOUSEBUTTONDOWN:*

*mx,my=e.pos*

*for i,img in enumerate(player1\_images):*

*r1=pygame.Rect(WIDTH//4-CELL\_SIZE//2,HEIGHT//4+i\*(CELL\_SIZE+10),CELL\_SIZE,CELL\_SIZE)*

*if r1.collidepoint(mx,my):*

*player1\_img,player1\_animal=img,ANIMAL\_IMAGES[i][1];chosen1=True*

*for i,img in enumerate(player2\_images):*

*r2=pygame.Rect(WIDTH\*3//4-CELL\_SIZE//2,HEIGHT//4+i\*(CELL\_SIZE+10),CELL\_SIZE,CELL\_SIZE)*

*if r2.collidepoint(mx,my):*

*player2\_img,player2\_animal=img,ANIMAL\_IMAGES[i][1];chosen2=True*

*def* ***main\_menu****():*

*sr=pygame.Rect(WIDTH//2-150,HEIGHT//2+60,300,50)*

*cr=pygame.Rect(WIDTH//2-150,HEIGHT//2+130,300,50)*

*er=pygame.Rect(WIDTH//2-150,HEIGHT//2+200,300,50)*

*ng=1*

*while True:*

*screen.fill(COLOR\_BG)*

*title\_surf=FONT\_TITLE.render("Escape The Grid",True,COLOR\_TEXT)*

*screen.blit(title\_surf,title\_surf.get\_rect(center=(WIDTH//2,HEIGHT//2-220)))*

*# Modern Number of Games selector (higher up)*

*label\_surf = FONT\_BUTTON.render("Number of Games", True, COLOR\_ACCENT)*

*label\_rect = label\_surf.get\_rect(center=(WIDTH//2, HEIGHT//2-120))*

*screen.blit(label\_surf, label\_rect)*

*# Large number with glowing pill and shadow*

*num\_center = (WIDTH//2, HEIGHT//2-20)*

*pill\_width, pill\_height = 120, 90*

*pill\_rect = pygame.Rect(0,0,pill\_width,pill\_height)*

*pill\_rect.center = num\_center*

*# Glow effect*

*glow\_surf = pygame.Surface((pill\_width+24, pill\_height+24), pygame.SRCALPHA)*

*pygame.draw.ellipse(glow\_surf, (\*COLOR\_ACCENT, 60), glow\_surf.get\_rect())*

*screen.blit(glow\_surf, (pill\_rect.x-12, pill\_rect.y-12))*

*# Pill background*

*pygame.draw.ellipse(screen, COLOR\_BUTTON, pill\_rect)*

*pygame.draw.ellipse(screen, COLOR\_ACCENT, pill\_rect, 4)*

*# Number shadow*

*num\_surf\_shadow = FONT\_TITLE.render(f"{ng}", True, (0,0,0))*

*num\_rect = num\_surf\_shadow.get\_rect(center=(num\_center[0], num\_center[1]+4))*

*screen.blit(num\_surf\_shadow, num\_rect)*

*# Number*

*num\_surf = FONT\_TITLE.render(f"{ng}", True, COLOR\_TEXT)*

*num\_rect = num\_surf.get\_rect(center=num\_center)*

*screen.blit(num\_surf, num\_rect)*

*# Large arrow buttons*

*arrow\_size = 60*

*left\_arrow\_rect = pygame.Rect(num\_rect.left-arrow\_size-30, num\_rect.centery-arrow\_size//2, arrow\_size, arrow\_size)*

*right\_arrow\_rect = pygame.Rect(num\_rect.right+30, num\_rect.centery-arrow\_size//2, arrow\_size, arrow\_size)*

*# Draw arrows with hover effect*

*mouse\_pos = pygame.mouse.get\_pos()*

*for rect, symbol in [(left\_arrow\_rect, "<"), (right\_arrow\_rect, ">")]:*

*hovered = rect.collidepoint(mouse\_pos)*

*col = COLOR\_BUTTON\_HOVER if hovered else COLOR\_BUTTON*

*pygame.draw.rect(screen, col, rect, border\_radius=16)*

*arrow\_surf = FONT\_TITLE.render(symbol, True, COLOR\_ACCENT)*

*arrow\_rect = arrow\_surf.get\_rect(center=rect.center)*

*screen.blit(arrow\_surf, arrow\_rect)*

*# Main menu buttons (lower)*

*draw\_button("Start Game",sr,sr.collidepoint(pygame.mouse.get\_pos()))*

*draw\_button("Choose Players",cr,cr.collidepoint(pygame.mouse.get\_pos()))*

*draw\_button("Exit",er,er.collidepoint(pygame.mouse.get\_pos()))*

*pygame.display.flip()*

*for e in pygame.event.get():*

*if e.type==pygame.QUIT:return 'exit',0*

*if e.type==pygame.MOUSEBUTTONDOWN:*

*if sr.collidepoint(e.pos):return 'start',ng*

*if cr.collidepoint(e.pos):choose\_player\_images()*

*if er.collidepoint(e.pos):return 'exit',0*

*if left\_arrow\_rect.collidepoint(e.pos):ng=max(1,ng-1)*

*if right\_arrow\_rect.collidepoint(e.pos):ng+=1*

*if e.type==pygame.KEYDOWN:*

*if e.key==pygame.K\_UP:ng+=1*

*if e.key==pygame.K\_DOWN:ng=max(1,ng-1)*

*def* ***main****():*

*p1\_data,p2\_data=load\_training\_data()*

*while True:*

*action,ng=main\_menu()*

*if action=='exit':pygame.quit();sys.exit()*

*w1=w2=0*

*for \_ in range(ng):*

*initialize\_players()*

*player1.load\_training\_data(p1\_data)*

*player2.load\_training\_data(p2\_data)*

*win=play\_game()*

*if win=='Player 1':w1+=1*

*if win=='Player 2':w2+=1*

*player1.train\_model();player2.train\_model()*

*print(f"Final Score - P1: {w1}, P2: {w2}")*

*if \_\_name\_\_ == '\_\_main\_\_':*

*main()*