**פרויקט גמר - אלגוריתמים מתקדמים לתכנון ותזמון מערכות נבונות**

**מסמך מסכם - SRS**

**מגישים:**

במסמך המוגש

**קישורים:**  
[Youtube presentation](https://youtu.be/US__vTyvNSA)

[Youtube project demonstration](https://youtu.be/DQQgX3O9vAs)

[Github repository](https://github.com/ilan53/pygame_hide_seek/branches)

**סיכום**

הפרויקט הוא **משחק מחשב אינטראקטיבי**, אשר פותח בעזרת **שפת Python** ובשימוש בספריית **Pygame** ליצירת ממשק גרפי. המשחק מיועד ל-2 שחקנים – **שחקן נגד שחקן אחר** או **שחקן נגד מחשב**, כאשר לשחקן הממוחשב קיימות **שתי רמות קושי**: רגילה וקשה, בהתאם לאתגרים הלוגיים שהשחקן האנושי מחפש.

**תיאור הבעיה:**

המשחק פותח מתוך צורך ביצירת **חוויית משחק מאתגרת, תחרותית ואינטליגנטית**, שמשלבת יכולות חיפוש, תכנון מהלכים והסקת מסקנות. המשחק מעודד **חשיבה טקטית**, ומדמה סיטואציה של חיפוש אחרי יעד מסתתר בלוח, תוך כדי שימוש באסטרטגיה, ניהול משאבים, ותגובה לאירועים משתנים. זהו פתרון שמטרתו לשלב כיף עם אתגר אינטלקטואלי, במיוחד עבור שחקנים שאוהבים משחקי לוח ואסטרטגיה ממוחשבים.

**תיאור המשחק:**

המשחק מתרחש על **לוח בגודל 10×10**, שבו כל שחקן נע בתורו על גבי המשבצות. אחת הדמויות במשחק היא **ג'רי העכבר**, שמסתתר באחד מהמיקומים בלוח. שחקנים אחרים – כגון **תום החתול** ו**ספייק הכלב** – צריכים לנסות למצוא אותו.

לכל שחקן יכולת לבחור דמות עם **יכולת ייחודית** שמשפיעה על מהלך המשחק:

* **תום** ו**ספייק** יכולים לבחור **להתריע בפני ג'רי** (מה שיגרום לו לברוח ולהסתתר מחדש), או **למקם מחסום בגודל 2×1** בלוח – אופקית או אנכית – שמקשה על היריב לנוע בחופשיות.
* ג'רי לעומת זאת מנסה להישאר מוסתר לאורך שלבים שונים, ועשוי להחליף מקום אחת לשלב.

בנוסף, במהלך כל שלב מוצגת **קופסת מתנה** שמופיעה במקום אקראי בלוח. שחקן שמגיע אליה זוכה ביתרון – **הקפאת היריב לשתי תורות**.

**אלמנטים טכנולוגיים ואינטליגנטיים:**

בכדי לשפר את חוויית המשחק ולספק רמזים לשחקנים, כל שחקן מקבל מידע על **המרחק שלו מג'רי**, באמצעות שילוב של:

* **אלגוריתם A\*** – לחישוב מסלול אופטימלי לשם איתור ג'רי תוך התחשבות במכשולים.
* **מרחק מנהטן** – לחישוב המרחק בין מיקומו הנוכחי של השחקן לבין מיקומו של ג'רי, מבלי לחשוף את המיקום המדויק.

שילוב האלגוריתמים מוסיף רובד של תחכום למשחק ומעודד את השחקנים לחשוב אסטרטגית על כל צעד.

**מטרת המשחק:**

המטרה של כל שחקן היא להיות הראשון שמאתר את מיקומו של ג'רי בלוח. המשחק כולל מספר שלבים, כאשר בכל שלב משתנים המיקומים והמצבים בלוח. השחקן שמאתר את ג'רי ראשון מנצח את השלב.

**Code:**

import pygame

import sys

import random

import heapq

import math

from enum import Enum

pygame.init()

pygame.mixer.init()

pygame.mixer.music.set\_volume(0.5)

GRID\_SIZE = 10

CELL\_SIZE = 60

WIDTH = GRID\_SIZE \* CELL\_SIZE

HEIGHT = GRID\_SIZE \* CELL\_SIZE

WINDOW\_WIDTH = WIDTH + 400

WINDOW\_HEIGHT = HEIGHT + 200

FPS = 60

WHITE = (255, 255, 255)

BLACK = (0, 0, 0)

GREEN = (0, 255, 0)

LIGHT\_GREEN = (180, 255, 230)

RED = (255, 0, 0)

ORANGE = (255, 165, 0)

# --- Center the grid in the window ---

GRID\_OFFSET\_X = (WINDOW\_WIDTH - WIDTH) // 3

GRID\_OFFSET\_Y = 120 # Slightly more space for buttons above

screen = pygame.display.set\_mode((WINDOW\_WIDTH, WINDOW\_HEIGHT))

FONT = pygame.font.SysFont("Segoe UI Emoji", 28)

class GameState(Enum):

MENU = 1

PLAYER1\_TURN = 2

PLAYER2\_TURN = 3

GAME\_OVER = 4

class HideSeekGame:

def \_\_init\_\_(self):

self.stars = [

{

"x": random.randint(0, WINDOW\_WIDTH),

"y": random.randint(0, WINDOW\_HEIGHT),

"size": random.randint(2, 5),

"speed": random.uniform(0.3, 0.8),

"color": random.choice([(255,255,255), (255,230,200), (200,220,255), (255,255,180)]),

"alpha": random.randint(100, 255),

"alpha\_direction": random.choice([-1, 1])

}

for \_ in range(60)

]

self.main\_menu\_button = None

self.clock = pygame.time.Clock()

self.font = pygame.font.Font(None, 24)

self.big\_font = pygame.font.Font(None, 36)

self.game\_mode = None # 'pvc' or 'pvp'

self.computer\_difficulty = "normal" # 'normal' or 'hard'

self.next\_round\_button = None

self.move\_target\_button = None

self.computer\_thinking = False

self.computer\_think\_time = 0

self.player1\_moved\_target = False

self.player2\_moved\_target = False

# Block system variables

self.blocks = [] # List of block positions and orientations

self.player1\_blocks\_remaining = 1

self.player2\_blocks\_remaining = 1

self.place\_block\_button = None

self.block\_placement\_mode = False

self.block\_orientation = "horizontal" # "horizontal" or "vertical"

# Block preview variables

self.block\_preview\_pos = None # (x, y) position for preview

self.block\_preview\_valid = False # Whether the preview position is valid

self.tom\_images = {

"up": pygame.image.load("tom/tom\_walking\_up.png"),

"down": pygame.image.load("tom/tom\_walking\_down.png"),

"left": pygame.image.load("tom/tom\_walking\_left.png"),

"right": pygame.image.load("tom/tom\_walking\_right.png"),

"idle": pygame.image.load("tom/tom\_standing.png"),

}

for key in self.tom\_images:

self.tom\_images[key] = pygame.transform.scale(self.tom\_images[key], (CELL\_SIZE, CELL\_SIZE))

self.tom\_direction = "idle"

self.spike\_images = {

"up": pygame.image.load("spike/spike\_walking\_up.png"),

"down": pygame.image.load("spike/spike\_walking\_down.png"),

"left": pygame.image.load("spike/spike\_walking\_left.png"),

"right": pygame.image.load("spike/spike\_walking\_right.png"),

"idle": pygame.image.load("spike/spike\_standing.png"),

}

for key in self.spike\_images:

self.spike\_images[key] = pygame.transform.scale(self.spike\_images[key], (CELL\_SIZE, CELL\_SIZE))

self.spike\_direction = "idle"

# Load block images

self.block\_horizontal = pygame.transform.scale(pygame.image.load("assets/block\_horizontal.png"), (CELL\_SIZE \* 2, CELL\_SIZE))

self.block\_vertical = pygame.transform.scale(pygame.image.load("assets/block\_vertical.png"), (CELL\_SIZE, CELL\_SIZE \* 2))

# Load cheese wedge image for hiding spots

self.cheese\_image = pygame.transform.scale(pygame.image.load("assets/Cheese-wedge.png"), (CELL\_SIZE, CELL\_SIZE))

self.jerry\_image = pygame.transform.scale(pygame.image.load("jerry/jerry\_hiding2.png"), (CELL\_SIZE, CELL\_SIZE))

self.jerry\_running\_frames = [

pygame.transform.scale(

pygame.image.load(f"jerry/frame\_{i:02d}\_delay-0.08s.png"),

(CELL\_SIZE, CELL\_SIZE)

)

for i in range(14)

]

self.jerry\_running\_frame\_index = 0

self.jerry\_running\_frame\_timer = 0

self.jerry\_running\_frame\_duration = 50 # milliseconds = 0.05s

self.show\_jerry\_running = False

self.jerry\_running\_pos = None

self.jerry\_running\_start\_time = 0

self.feedback\_images = {

"FOUND": pygame.image.load("feed\_back/found.png"),

"BURNING": pygame.image.load("feed\_back/burning\_hot.png"),

"HOT": pygame.image.load("feed\_back/hot.png"),

"WARM": pygame.image.load("feed\_back/warm.png"),

"COOL": pygame.image.load("feed\_back/cool.png"),

"COLD": pygame.image.load("feed\_back/cold.png")

}

for key in self.feedback\_images:

self.feedback\_images[key] = pygame.transform.scale(self.feedback\_images[key], (180, 180))

self.state = GameState.MENU

self.seeker1\_pos = (0, 0)

self.seeker2\_pos = (GRID\_SIZE - 1, GRID\_SIZE - 1)

self.hidden\_pos = None

self.feedback\_text = ""

self.winner = None

self.hiding\_spots = []

self.generate\_hiding\_spots()

self.tutorial\_image = pygame.image.load("assets/tutorial.png")

self.tutorial\_image = pygame.transform.scale(self.tutorial\_image, (887, 426))

self.player1\_keys\_image = pygame.image.load("assets/player1\_keys.jfif")

self.player1\_keys\_image = pygame.transform.scale(self.player1\_keys\_image, (160, 100))

self.player2\_keys\_image = pygame.image.load("assets/player2\_keys.jpg")

self.player2\_keys\_image = pygame.transform.scale(self.player2\_keys\_image, (160, 100))

self.FIND\_JERRY\_FIRST = pygame.image.load("assets/FIND\_JERRY\_FIRST.png")

self.FIND\_JERRY\_FIRST = pygame.transform.scale(self.FIND\_JERRY\_FIRST, (160, 160))

self.surprise\_image = pygame.image.load("assets/surprise-gift.png")

self.surprise\_image = pygame.transform.scale(self.surprise\_image, (160, 160))

# --- Frozen images ---

self.tom\_frozen\_image = pygame.transform.scale(pygame.image.load("tom/tom\_frozen.png"), (CELL\_SIZE, CELL\_SIZE))

self.spike\_frozen\_image = pygame.transform.scale(pygame.image.load("spike/spike\_frozen.png"), (CELL\_SIZE, CELL\_SIZE))

# If you have a tom frozen image, use: self.tom\_frozen\_image = pygame.transform.scale(pygame.image.load("tom/tom\_frozen.png"), (CELL\_SIZE, CELL\_SIZE))

# Freeze state

self.player1\_frozen\_turns = 0

self.player2\_frozen\_turns = 0

# Unfreeze animation state

self.player1\_unfreezing = False

self.player2\_unfreezing = False

self.player1\_unfreeze\_timer = 0

self.player2\_unfreeze\_timer = 0

self.unfreeze\_anim\_duration = 500 # ms

self.unfreeze\_anim\_jitter = 6 # px

# --- Gift Box Animation ---

self.gift\_box\_frames = [

pygame.transform.scale(

pygame.image.load(f"suprise\_box/on\_board/frame\_{i:03d}\_delay-0.03s.gif"),

(CELL\_SIZE, CELL\_SIZE)

) for i in range(45)

]

self.gift\_box\_frame\_index = 0

self.gift\_box\_frame\_timer = 0

self.gift\_box\_frame\_duration = 30 # ms per frame

self.gift\_box\_location = None

self.place\_gift\_box()

# --- Gift Box Pop Animation ---

self.gift\_box\_pop\_frames = [

pygame.transform.scale(

pygame.image.load(f"suprise\_box/take/frame\_{i:03d}\_delay-0.03s.gif"),

(CELL\_SIZE, CELL\_SIZE)

) for i in range(114, 150)

]

self.gift\_box\_popping = False

self.gift\_box\_pop\_frame\_index = 0

self.gift\_box\_pop\_frame\_timer = 0

self.gift\_box\_pop\_frame\_duration = 25 # ms per frame (faster)

self.gift\_box\_pop\_position = None

# --- Score tracking ---

self.scores = {"Tom": 0, "Spike": 0, "Computer": 0}

self.last\_game\_mode = None

self.debug\_message = None

def show\_title\_screen(self):

# Always reset scores and last\_game\_mode when entering main menu

self.scores = {"Tom": 0, "Spike": 0, "Computer": 0}

self.last\_game\_mode = None

background = pygame.image.load("assets/title\_screen.png")

background = pygame.transform.scale(background, (WINDOW\_WIDTH, WINDOW\_HEIGHT))

button\_color = (255, 200, 0)

button\_rect\_pvc = pygame.Rect(WINDOW\_WIDTH // 2 - 120, WINDOW\_HEIGHT - 220, 240, 50)

button\_rect\_pvp = pygame.Rect(WINDOW\_WIDTH // 2 - 120, WINDOW\_HEIGHT - 150, 240, 50)

button\_rect\_tutorial = pygame.Rect(WINDOW\_WIDTH // 2 - 120, WINDOW\_HEIGHT - 80, 240, 40)

# Difficulty selection UI

radio\_normal = pygame.Rect(WINDOW\_WIDTH // 2 - 100, WINDOW\_HEIGHT - 300, 30, 30)

radio\_hard = pygame.Rect(WINDOW\_WIDTH // 2 + 40, WINDOW\_HEIGHT - 300, 30, 30)

start\_button\_rect = pygame.Rect(WINDOW\_WIDTH // 2 - 80, WINDOW\_HEIGHT - 220, 160, 50)

show\_difficulty = False

selected\_difficulty = "normal"

running\_title = True

while running\_title:

screen.blit(background, (0, 0))

# Draw buttons

if not show\_difficulty:

pygame.draw.rect(screen, button\_color, button\_rect\_pvc)

pygame.draw.rect(screen, button\_color, button\_rect\_pvp)

pygame.draw.rect(screen, button\_color, button\_rect\_tutorial)

start\_text\_pvc = FONT.render("Player vs Computer", True, (0, 0, 0))

start\_text\_pvp = FONT.render("Player vs Player", True, (0, 0, 0))

tutorial\_text = FONT.render("Tutorial", True, (0, 0, 0))

screen.blit(start\_text\_pvc, start\_text\_pvc.get\_rect(center=button\_rect\_pvc.center))

screen.blit(start\_text\_pvp, start\_text\_pvp.get\_rect(center=button\_rect\_pvp.center))

screen.blit(tutorial\_text, tutorial\_text.get\_rect(center=button\_rect\_tutorial.center))

else:

# Draw difficulty selection background

bg\_rect = pygame.Rect(WINDOW\_WIDTH // 2 - 170, WINDOW\_HEIGHT - 360, 360, 200)

pygame.draw.rect(screen, (245, 240, 255), bg\_rect, border\_radius=18)

pygame.draw.rect(screen, (120, 120, 180), bg\_rect, 4, border\_radius=18)

# Draw difficulty selection

diff\_label = FONT.render("Computer Difficulty:", True, (0, 0, 0))

screen.blit(diff\_label, (WINDOW\_WIDTH // 2 - 140, WINDOW\_HEIGHT - 340))

# Radio buttons

pygame.draw.circle(screen, BLACK, radio\_normal.center, 15, 2)

pygame.draw.circle(screen, BLACK, radio\_hard.center, 15, 2)

if selected\_difficulty == "normal":

pygame.draw.circle(screen, (0, 200, 0), radio\_normal.center, 9)

else:

pygame.draw.circle(screen, (0, 200, 0), radio\_hard.center, 9)

normal\_text = FONT.render("Normal", True, (0, 0, 0))

hard\_text = FONT.render("Hard", True, (0, 0, 0))

screen.blit(normal\_text, (radio\_normal.right + 10, radio\_normal.y - 2))

screen.blit(hard\_text, (radio\_hard.right + 10, radio\_hard.y - 2))

# Start button

pygame.draw.rect(screen, button\_color, start\_button\_rect)

start\_text = FONT.render("Start", True, (0, 0, 0))

screen.blit(start\_text, start\_text.get\_rect(center=start\_button\_rect.center))

pygame.display.flip()

for event in pygame.event.get():

if event.type == pygame.QUIT:

pygame.quit()

sys.exit()

elif event.type == pygame.MOUSEBUTTONDOWN:

if not show\_difficulty:

if button\_rect\_pvc.collidepoint(event.pos):

show\_difficulty = True

elif button\_rect\_pvp.collidepoint(event.pos):

self.game\_mode = 'pvp'

running\_title = False

self.start\_game()

elif button\_rect\_tutorial.collidepoint(event.pos):

self.show\_tutorial\_screen()

else:

if radio\_normal.collidepoint(event.pos):

selected\_difficulty = "normal"

elif radio\_hard.collidepoint(event.pos):

selected\_difficulty = "hard"

elif start\_button\_rect.collidepoint(event.pos):

self.game\_mode = 'pvc'

self.computer\_difficulty = selected\_difficulty

running\_title = False

self.start\_game()

def show\_tutorial\_screen(self):

running\_tutorial = True

button\_width = 160

button\_height = 38

button\_x = 24

button\_y = 24

self.main\_menu\_button = pygame.Rect(button\_x, button\_y, button\_width, button\_height)

while running\_tutorial:

screen.fill((240, 240, 255))

# כותרת

title = self.big\_font.render("Tutorial", True, BLACK)

screen.blit(title, (WINDOW\_WIDTH // 2 - title.get\_width() // 2, 30))

# 🧠 הצגת תמונה ראשית במרכז

tutorial\_rect = self.tutorial\_image.get\_rect(center=(WINDOW\_WIDTH // 2, 150 + self.tutorial\_image.get\_height() // 3.2))

screen.blit(self.tutorial\_image, tutorial\_rect.topleft)

# 🕹️ Player 1 keys – ימין למטה

p1\_x = WINDOW\_WIDTH - 200

p1\_y = WINDOW\_HEIGHT - 180

screen.blit(self.player1\_keys\_image, (p1\_x, p1\_y))

p1\_text = self.font.render("Player 1", True, BLACK)

screen.blit(p1\_text, (p1\_x + 40, p1\_y + 110))

# 🧀 Find Jerry First – באמצע בין השניים

fjf\_rect = self.FIND\_JERRY\_FIRST.get\_rect(center=(WINDOW\_WIDTH // 2, p1\_y + 50))

screen.blit(self.FIND\_JERRY\_FIRST, fjf\_rect.topleft)

# 📦 Surprise gift explanation image – מתחת לתמונה הראשית

surprise\_rect = self.surprise\_image.get\_rect(center=(105, tutorial\_rect.bottom + -290))

screen.blit(self.surprise\_image, surprise\_rect.topleft)

# 🕹️ Player 2 keys – שמאל למטה

p2\_x = 60

p2\_y = WINDOW\_HEIGHT - 180

screen.blit(self.player2\_keys\_image, (p2\_x, p2\_y))

p2\_text = self.font.render("Player 2", True, BLACK)

screen.blit(p2\_text, (p2\_x + 40, p2\_y + 110))

# 🔙 Main Menu button

self.main\_menu\_button = pygame.Rect(button\_x, button\_y, button\_width, button\_height)

pygame.draw.rect(screen, (200, 200, 255), self.main\_menu\_button, border\_radius=8)

pygame.draw.rect(screen, BLACK, self.main\_menu\_button, 3, border\_radius=8)

menu\_text = self.font.render("Main Menu", True, (0, 0, 0))

screen.blit(menu\_text, menu\_text.get\_rect(center=self.main\_menu\_button.center))

pygame.display.flip()

for event in pygame.event.get():

if event.type == pygame.QUIT:

pygame.quit()

sys.exit()

elif event.type == pygame.MOUSEBUTTONDOWN:

if self.main\_menu\_button.collidepoint(event.pos):

running\_tutorial = False

def place\_gift\_box(self):

# Place the gift box at a random location not occupied by players or hiding spots

excluded = [(0, 0), (GRID\_SIZE-1, GRID\_SIZE-1)] + self.hiding\_spots

while True:

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

if pos not in excluded:

self.gift\_box\_location = pos

break

def generate\_hiding\_spots(self):

self.hiding\_spots = []

# Define starting positions that should be excluded

excluded\_positions = [(0, 0), (GRID\_SIZE-1, GRID\_SIZE-1)] # Tom's and Spike's starting positions

for \_ in range(random.randint(8, 12)):

while True:

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

if pos not in self.hiding\_spots and pos not in excluded\_positions:

self.hiding\_spots.append(pos)

break

# After hiding spots are generated, place the gift box

self.place\_gift\_box()

def is\_position\_blocked(self, pos):

"""Check if a position is blocked by any block"""

x, y = pos

for block in self.blocks:

block\_x, block\_y, orientation = block

if orientation == "horizontal":

if x == block\_x and y == block\_y or x == block\_x and y == block\_y + 1:

return True

else: # vertical

if x == block\_x and y == block\_y or x == block\_x + 1 and y == block\_y:

return True

return False

def can\_place\_block(self, x, y, orientation):

"""Check if a block can be placed at the given position and orientation"""

if orientation == "horizontal":

# Check if both cells are within bounds

if y + 1 >= GRID\_SIZE:

return False

# Check if either cell is blocked

if self.is\_position\_blocked((x, y)) or self.is\_position\_blocked((x, y + 1)):

return False

# Check if either cell is occupied by a player

if (x, y) == self.seeker1\_pos or (x, y) == self.seeker2\_pos:

return False

if (x, y + 1) == self.seeker1\_pos or (x, y + 1) == self.seeker2\_pos:

return False

# Check if either cell is a hiding spot

if (x, y) in self.hiding\_spots or (x, y + 1) in self.hiding\_spots:

return False

# Prevent blocking a gift box

if self.gift\_box\_location is not None:

if (x, y) == self.gift\_box\_location or (x, y + 1) == self.gift\_box\_location:

return False

else: # vertical

# Check if both cells are within bounds

if x + 1 >= GRID\_SIZE:

return False

# Check if either cell is blocked

if self.is\_position\_blocked((x, y)) or self.is\_position\_blocked((x + 1, y)):

return False

# Check if either cell is occupied by a player

if (x, y) == self.seeker1\_pos or (x, y) == self.seeker2\_pos:

return False

if (x + 1, y) == self.seeker1\_pos or (x + 1, y) == self.seeker2\_pos:

return False

# Check if either cell is a hiding spot

if (x, y) in self.hiding\_spots or (x + 1, y) in self.hiding\_spots:

return False

# Prevent blocking a gift box

if self.gift\_box\_location is not None:

if (x, y) == self.gift\_box\_location or (x + 1, y) == self.gift\_box\_location:

return False

# Prevent trapping a hiding spot in a corner with two blocks

# For each corner, if a hiding spot is there, check if this block would trap it

corners = [(0,0), (0,GRID\_SIZE-1), (GRID\_SIZE-1,0), (GRID\_SIZE-1,GRID\_SIZE-1)]

for hx, hy in self.hiding\_spots:

if (hx, hy) in corners:

# For each corner, check the two adjacent cells

if (hx, hy) == (0,0):

adj1 = (0,1)

adj2 = (1,0)

elif (hx, hy) == (0,GRID\_SIZE-1):

adj1 = (0,GRID\_SIZE-2)

adj2 = (1,GRID\_SIZE-1)

elif (hx, hy) == (GRID\_SIZE-1,0):

adj1 = (GRID\_SIZE-2,0)

adj2 = (GRID\_SIZE-1,1)

elif (hx, hy) == (GRID\_SIZE-1,GRID\_SIZE-1):

adj1 = (GRID\_SIZE-2,GRID\_SIZE-1)

adj2 = (GRID\_SIZE-1,GRID\_SIZE-2)

# If this block would block either adjacent cell, and the other is already blocked, disallow

blocks\_adj1 = self.is\_position\_blocked(adj1) or \

((orientation == "horizontal" and (x, y) == adj1) or (orientation == "vertical" and (x, y) == adj1))

blocks\_adj2 = self.is\_position\_blocked(adj2) or \

((orientation == "horizontal" and (x, y) == adj2) or (orientation == "vertical" and (x, y) == adj2))

if blocks\_adj1 and blocks\_adj2:

return False

return True

def place\_block(self, x, y, orientation, player):

"""Place a block at the given position and orientation"""

if self.can\_place\_block(x, y, orientation):

self.blocks.append((x, y, orientation))

if player == 1:

self.player1\_blocks\_remaining -= 1

else:

self.player2\_blocks\_remaining -= 1

return True

return False

def a\_star\_distance(self, start, goal):

if start == goal:

return 0

open\_set = [(0, start)]

g\_score = {start: 0}

while open\_set:

\_, current = heapq.heappop(open\_set)

if current == goal:

return g\_score[current]

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

neighbor = (current[0]+dx, current[1]+dy)

if 0 <= neighbor[0] < GRID\_SIZE and 0 <= neighbor[1] < GRID\_SIZE:

# Check if the neighbor position is blocked

if self.is\_position\_blocked(neighbor):

continue

temp = g\_score[current] + 1

if neighbor not in g\_score or temp < g\_score[neighbor]:

g\_score[neighbor] = temp

f = temp + abs(neighbor[0]-goal[0]) + abs(neighbor[1]-goal[1])

heapq.heappush(open\_set, (f, neighbor))

return float('inf')

def get\_feedback(self, distance):

if distance == 0:

return "FOUND"

elif distance <= 2:

return "BURNING"

elif distance <= 4:

return "HOT"

elif distance <= 6:

return "WARM"

elif distance <= 10:

return "COOL"

return "COLD"

def computer\_move(self):

# Use difficulty to branch AI logic

if self.computer\_difficulty == "normal":

# --- Move Target logic (same as hard mode) ---

if not self.player2\_moved\_target:

computer\_dist = self.a\_star\_distance(self.seeker2\_pos, self.hidden\_pos)

player\_dist = self.a\_star\_distance(self.seeker1\_pos, self.hidden\_pos)

if computer\_dist > 6 and player\_dist <= 4:

self.move\_target\_to\_new\_location()

self.player2\_moved\_target = True

self.state = GameState.PLAYER1\_TURN

return

# --- Block placement logic (improved for normal mode) ---

player\_dist = self.a\_star\_distance(self.seeker1\_pos, self.hidden\_pos)

computer\_dist = self.a\_star\_distance(self.seeker2\_pos, self.hidden\_pos)

# --- Gift box logic: go for the gift if it helps ---

go\_for\_gift = False

if self.gift\_box\_location:

# Get path to gift and to Jerry

path\_to\_gift = self.a\_star\_path(self.seeker2\_pos, self.gift\_box\_location)

path\_to\_jerry = self.a\_star\_path(self.seeker2\_pos, self.hidden\_pos)

# If the gift is on the way to Jerry, or if freezing the player would let computer win

player\_path\_to\_jerry = self.a\_star\_path(self.seeker1\_pos, self.hidden\_pos)

if self.gift\_box\_location in path\_to\_jerry:

go\_for\_gift = True

else:

# If computer is behind, but freezing player would let it catch up or win

if player\_dist < computer\_dist and (computer\_dist - player\_dist) <= 2:

# Estimate: if player is frozen for 2 turns, computer can catch up

go\_for\_gift = True

if go\_for\_gift:

# Move toward the gift box

path = self.a\_star\_path(self.seeker2\_pos, self.gift\_box\_location)

if len(path) > 1:

next\_pos = path[1]

dx = next\_pos[0] - self.seeker2\_pos[0]

dy = next\_pos[1] - self.seeker2\_pos[1]

if dx == -1:

self.spike\_direction = "up"

elif dx == 1:

self.spike\_direction = "down"

elif dy == -1:

self.spike\_direction = "left"

elif dy == 1:

self.spike\_direction = "right"

else:

self.spike\_direction = "idle"

self.seeker2\_pos = next\_pos

# Trigger gift box animation and freeze opponent immediately

if self.seeker2\_pos == self.gift\_box\_location:

self.gift\_box\_popping = True

self.gift\_box\_pop\_position = self.gift\_box\_location

self.gift\_box\_pop\_frame\_index = 0

self.gift\_box\_pop\_frame\_timer = pygame.time.get\_ticks()

self.gift\_box\_location = None

self.freeze\_opponent(2)

self.state = GameState.PLAYER1\_TURN

return

# --- Block placement logic for normal mode ---

should\_place\_block = False

if player\_dist <= 5 and computer\_dist > player\_dist:

should\_place\_block = True

elif player\_dist <= 3:

should\_place\_block = True

elif player\_dist < computer\_dist:

should\_place\_block = True

if should\_place\_block and self.player2\_blocks\_remaining > 0 and random.random() < 0.5: # 50% chance for normal mode

if self.computer\_place\_block():

self.state = GameState.PLAYER1\_TURN

return

# --- Movement logic: feedback-based ---

x, y = self.seeker2\_pos

feedback\_distance = self.a\_star\_distance(self.seeker2\_pos, self.hidden\_pos)

prev\_pos = getattr(self, '\_prev\_seeker2\_pos', None)

best\_moves = []

min\_new\_distance = feedback\_distance

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:

if self.is\_position\_blocked((nx, ny)):

continue

new\_distance = self.a\_star\_distance((nx, ny), self.hidden\_pos)

if new\_distance < min\_new\_distance:

min\_new\_distance = new\_distance

best\_moves = [(nx, ny)]

elif new\_distance == min\_new\_distance:

best\_moves.append((nx, ny))

# Prefer moves that decrease the distance

if best\_moves and min\_new\_distance < feedback\_distance:

chosen\_move = random.choice(best\_moves)

else:

# If no move decreases the distance, allow moves that keep the same distance, but avoid going back to previous position

same\_dist\_moves = []

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:

if self.is\_position\_blocked((nx, ny)):

continue

if self.a\_star\_distance((nx, ny), self.hidden\_pos) == feedback\_distance:

if prev\_pos is None or (nx, ny) != prev\_pos:

same\_dist\_moves.append((nx, ny))

if same\_dist\_moves:

chosen\_move = random.choice(same\_dist\_moves)

else:

# If stuck, just pick any valid move

valid\_moves = []

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:

if not self.is\_position\_blocked((nx, ny)):

valid\_moves.append((nx, ny))

if valid\_moves:

chosen\_move = random.choice(valid\_moves)

else:

chosen\_move = (x, y)

dx = chosen\_move[0] - x

dy = chosen\_move[1] - y

if dx == -1:

self.spike\_direction = "up"

elif dx == 1:

self.spike\_direction = "down"

elif dy == -1:

self.spike\_direction = "left"

elif dy == 1:

self.spike\_direction = "right"

else:

self.spike\_direction = "idle"

self.\_prev\_seeker2\_pos = self.seeker2\_pos

self.seeker2\_pos = chosen\_move

if self.seeker2\_pos == self.hidden\_pos:

self.winner = "Computer"

# debug\_msg = f"DEBUG: last\_game\_mode={self.last\_game\_mode}, Computer score before={self.scores['Computer']}"

if self.last\_game\_mode == 'pvc':

self.scores["Computer"] += 1

# debug\_msg += f", after={self.scores['Computer']}"

# self.debug\_message = debug\_msg

pygame.mixer.music.stop()

pygame.mixer.music.load("sound\_track/lose.mp3")

pygame.mixer.music.play()

self.state = GameState.GAME\_OVER

else:

self.state = GameState.PLAYER1\_TURN

return

# Hard: original logic

# First, decide if computer should use Move Target button

if not self.player2\_moved\_target:

computer\_dist = self.a\_star\_distance(self.seeker2\_pos, self.hidden\_pos)

player\_dist = self.a\_star\_distance(self.seeker1\_pos, self.hidden\_pos)

# Use Move Target if computer is far and player is getting close

if computer\_dist > 6 and player\_dist <= 4:

self.move\_target\_to\_new\_location()

self.player2\_moved\_target = True

self.state = GameState.PLAYER1\_TURN

return

# Decide whether to place a block or move

player\_dist = self.a\_star\_distance(self.seeker1\_pos, self.hidden\_pos)

computer\_dist = self.a\_star\_distance(self.seeker2\_pos, self.hidden\_pos)

# --- Gift box logic: go for the gift if it helps (hard mode) ---

go\_for\_gift = False

if self.gift\_box\_location:

path\_to\_gift = self.a\_star\_path(self.seeker2\_pos, self.gift\_box\_location)

path\_to\_jerry = self.a\_star\_path(self.seeker2\_pos, self.hidden\_pos)

player\_path\_to\_jerry = self.a\_star\_path(self.seeker1\_pos, self.hidden\_pos)

if self.gift\_box\_location in path\_to\_jerry:

go\_for\_gift = True

else:

if player\_dist < computer\_dist and (computer\_dist - player\_dist) <= 2:

go\_for\_gift = True

if go\_for\_gift:

path = self.a\_star\_path(self.seeker2\_pos, self.gift\_box\_location)

if len(path) > 1:

next\_pos = path[1]

dx = next\_pos[0] - self.seeker2\_pos[0]

dy = next\_pos[1] - self.seeker2\_pos[1]

if dx == -1:

self.spike\_direction = "up"

elif dx == 1:

self.spike\_direction = "down"

elif dy == -1:

self.spike\_direction = "left"

elif dy == 1:

self.spike\_direction = "right"

else:

self.spike\_direction = "idle"

self.seeker2\_pos = next\_pos

# Trigger gift box animation and freeze opponent immediately

if self.seeker2\_pos == self.gift\_box\_location:

self.gift\_box\_popping = True

self.gift\_box\_pop\_position = self.gift\_box\_location

self.gift\_box\_pop\_frame\_index = 0

self.gift\_box\_pop\_frame\_timer = pygame.time.get\_ticks()

self.gift\_box\_location = None

self.freeze\_opponent(2)

self.state = GameState.PLAYER1\_TURN

return

# Enhanced block placement logic for hard mode

# More aggressive and strategic than normal mode

should\_place\_block = False

# Condition 1: Player is close to Jerry and computer is farther

if player\_dist <= 5 and computer\_dist > player\_dist:

should\_place\_block = True

# Condition 2: Player is getting very close (within 3 steps)

elif player\_dist <= 3:

should\_place\_block = True

# Condition 3: Player is closer than computer by any margin

elif player\_dist < computer\_dist:

should\_place\_block = True

# Condition 4: Strategic blocking - even if computer is closer, block to maintain advantage

elif computer\_dist <= 4 and player\_dist <= computer\_dist + 2:

should\_place\_block = True

# Higher probability and more aggressive for hard mode

if should\_place\_block and self.player2\_blocks\_remaining > 0 and random.random() < 0.9:

if self.computer\_place\_block():

self.state = GameState.PLAYER1\_TURN

return

# --- Block placement logic for normal mode ---

should\_place\_block = False

if player\_dist <= 5 and computer\_dist > player\_dist:

should\_place\_block = True

elif player\_dist <= 3:

should\_place\_block = True

elif player\_dist < computer\_dist:

should\_place\_block = True

if should\_place\_block and self.player2\_blocks\_remaining > 0 and random.random() < 0.5: # 50% chance for normal mode

if self.computer\_place\_block():

self.state = GameState.PLAYER1\_TURN

return

# Regular movement logic

best\_move = None

best\_score = float('inf')

x, y = self.seeker2\_pos

# Consider both distance to Jerry and blocking player's path

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:

# Check if the move is blocked

if self.is\_position\_blocked((nx, ny)):

continue

d = self.a\_star\_distance((nx, ny), self.hidden\_pos)

# Bonus score for moving towards Jerry

score = d

# Consider blocking player's path if player is close to Jerry

player\_dist = self.a\_star\_distance(self.seeker1\_pos, self.hidden\_pos)

if player\_dist <= 3:

# Try to get closer to Jerry to compete

score = d \* 0.8 # Prioritize getting to Jerry

if score < best\_score:

best\_score = score

best\_move = (nx, ny)

if best\_move:

dx = best\_move[0] - self.seeker2\_pos[0]

dy = best\_move[1] - self.seeker2\_pos[1]

if dx == -1:

self.spike\_direction = "up"

elif dx == 1:

self.spike\_direction = "down"

elif dy == -1:

self.spike\_direction = "left"

elif dy == 1:

self.spike\_direction = "right"

else:

self.spike\_direction = "idle"

self.seeker2\_pos = best\_move

if self.seeker2\_pos == self.hidden\_pos:

self.winner = "Computer"

debug\_msg = f"DEBUG: last\_game\_mode={self.last\_game\_mode}, Computer score before={self.scores['Computer']}"

if self.last\_game\_mode == 'pvc':

self.scores["Computer"] += 1

debug\_msg += f", after={self.scores['Computer']}"

self.debug\_message = debug\_msg

pygame.mixer.music.stop()

pygame.mixer.music.load("sound\_track/lose.mp3")

pygame.mixer.music.play()

self.state = GameState.GAME\_OVER

else:

self.state = GameState.PLAYER1\_TURN

def draw\_grid(self):

for x in range(GRID\_SIZE):

for y in range(GRID\_SIZE):

rect = pygame.Rect(GRID\_OFFSET\_X + y \* CELL\_SIZE, GRID\_OFFSET\_Y + x \* CELL\_SIZE, CELL\_SIZE, CELL\_SIZE)

# pygame.draw.rect(screen, WHITE, rect) # Removed to make background transparent

pygame.draw.rect(screen, BLACK, rect, 2)

# Draw cheese wedges for hiding spots (but not where players are standing)

for x, y in self.hiding\_spots:

# Only draw cheese if no player is on this spot

if (x, y) != self.seeker1\_pos and (x, y) != self.seeker2\_pos:

rect = pygame.Rect(GRID\_OFFSET\_X + y \* CELL\_SIZE, GRID\_OFFSET\_Y + x \* CELL\_SIZE, CELL\_SIZE, CELL\_SIZE)

screen.blit(self.cheese\_image, rect.topleft)

# Draw animated gift box if present and not popping

if self.gift\_box\_location and not self.gift\_box\_popping:

gx, gy = self.gift\_box\_location

rect = pygame.Rect(GRID\_OFFSET\_X + gy \* CELL\_SIZE, GRID\_OFFSET\_Y + gx \* CELL\_SIZE, CELL\_SIZE, CELL\_SIZE)

now = pygame.time.get\_ticks()

if now - self.gift\_box\_frame\_timer > self.gift\_box\_frame\_duration:

self.gift\_box\_frame\_index = (self.gift\_box\_frame\_index + 1) % len(self.gift\_box\_frames)

self.gift\_box\_frame\_timer = now

current\_frame = self.gift\_box\_frames[self.gift\_box\_frame\_index]

screen.blit(current\_frame, rect.topleft)

# Draw player images (this will be on top of cheese if they're on a hiding spot)

for x in range(GRID\_SIZE):

for y in range(GRID\_SIZE):

rect = pygame.Rect(GRID\_OFFSET\_X + y \* CELL\_SIZE, GRID\_OFFSET\_Y + x \* CELL\_SIZE, CELL\_SIZE, CELL\_SIZE)

if (x, y) == self.seeker1\_pos:

if self.player1\_frozen\_turns > 0:

screen.blit(self.tom\_frozen\_image, rect.topleft)

elif self.player1\_unfreezing:

now = pygame.time.get\_ticks()

if now - self.player1\_unfreeze\_timer < self.unfreeze\_anim\_duration:

jitter = self.unfreeze\_anim\_jitter

offset\_x = random.randint(-jitter, jitter)

offset\_y = random.randint(-jitter, jitter)

screen.blit(self.tom\_frozen\_image, (rect.x + offset\_x, rect.y + offset\_y))

else:

self.player1\_unfreezing = False

self.tom\_direction = "idle"

screen.blit(self.tom\_images[self.tom\_direction], rect.topleft)

else:

screen.blit(self.tom\_images[self.tom\_direction], rect.topleft)

elif (x, y) == self.seeker2\_pos:

if self.player2\_frozen\_turns > 0:

screen.blit(self.spike\_frozen\_image, rect.topleft)

elif self.player2\_unfreezing:

now = pygame.time.get\_ticks()

if now - self.player2\_unfreeze\_timer < self.unfreeze\_anim\_duration:

jitter = self.unfreeze\_anim\_jitter

offset\_x = random.randint(-jitter, jitter)

offset\_y = random.randint(-jitter, jitter)

screen.blit(self.spike\_frozen\_image, (rect.x + offset\_x, rect.y + offset\_y))

else:

self.player2\_unfreezing = False

self.spike\_direction = "idle"

screen.blit(self.spike\_images[self.spike\_direction], rect.topleft)

else:

screen.blit(self.spike\_images[self.spike\_direction], rect.topleft)

if self.state == GameState.GAME\_OVER and (x, y) == self.hidden\_pos:

screen.blit(self.jerry\_image, rect.topleft)

# Draw popping animation if active (on top of player)

if self.gift\_box\_popping and self.gift\_box\_pop\_position:

gx, gy = self.gift\_box\_pop\_position

rect = pygame.Rect(GRID\_OFFSET\_X + gy \* CELL\_SIZE, GRID\_OFFSET\_Y + gx \* CELL\_SIZE, CELL\_SIZE, CELL\_SIZE)

now = pygame.time.get\_ticks()

if now - self.gift\_box\_pop\_frame\_timer > self.gift\_box\_pop\_frame\_duration:

self.gift\_box\_pop\_frame\_index += 1

self.gift\_box\_pop\_frame\_timer = now

if self.gift\_box\_pop\_frame\_index < len(self.gift\_box\_pop\_frames):

current\_frame = self.gift\_box\_pop\_frames[self.gift\_box\_pop\_frame\_index]

screen.blit(current\_frame, rect.topleft)

else:

# Animation finished, remove box

self.gift\_box\_popping = False

self.gift\_box\_pop\_position = None

# Draw blocks

for block in self.blocks:

x, y, orientation = block

if orientation == "horizontal":

block\_rect = pygame.Rect(GRID\_OFFSET\_X + y \* CELL\_SIZE, GRID\_OFFSET\_Y + x \* CELL\_SIZE, CELL\_SIZE \* 2, CELL\_SIZE)

screen.blit(self.block\_horizontal, block\_rect.topleft)

else: # vertical

block\_rect = pygame.Rect(GRID\_OFFSET\_X + y \* CELL\_SIZE, GRID\_OFFSET\_Y + x \* CELL\_SIZE, CELL\_SIZE, CELL\_SIZE \* 2)

screen.blit(self.block\_vertical, block\_rect.topleft)

# Draw block preview

if self.block\_placement\_mode and self.block\_preview\_pos is not None:

x, y = self.block\_preview\_pos

preview\_color = GREEN if self.block\_preview\_valid else RED

if self.block\_orientation == "horizontal":

# Draw preview rectangle for horizontal block

preview\_rect = pygame.Rect(GRID\_OFFSET\_X + y \* CELL\_SIZE, GRID\_OFFSET\_Y + x \* CELL\_SIZE, CELL\_SIZE \* 2, CELL\_SIZE)

pygame.draw.rect(screen, preview\_color, preview\_rect, 3)

# Draw semi-transparent overlay

preview\_surface = pygame.Surface((CELL\_SIZE \* 2, CELL\_SIZE))

preview\_surface.set\_alpha(100)

preview\_surface.fill(preview\_color)

screen.blit(preview\_surface, preview\_rect.topleft)

else: # vertical

# Draw preview rectangle for vertical block

preview\_rect = pygame.Rect(GRID\_OFFSET\_X + y \* CELL\_SIZE, GRID\_OFFSET\_Y + x \* CELL\_SIZE, CELL\_SIZE, CELL\_SIZE \* 2)

pygame.draw.rect(screen, preview\_color, preview\_rect, 3)

# Draw semi-transparent overlay

preview\_surface = pygame.Surface((CELL\_SIZE, CELL\_SIZE \* 2))

preview\_surface.set\_alpha(100)

preview\_surface.fill(preview\_color)

screen.blit(preview\_surface, preview\_rect.topleft)

# ✨ הצגת ג'רי רץ במיקום הישן

if self.show\_jerry\_running and self.jerry\_running\_pos:

if pygame.time.get\_ticks() - self.jerry\_running\_start\_time < 1100:

now = pygame.time.get\_ticks()

if now - self.jerry\_running\_frame\_timer > self.jerry\_running\_frame\_duration:

if self.jerry\_running\_frame\_index < len(self.jerry\_running\_frames) - 1:

self.jerry\_running\_frame\_index += 1

self.jerry\_running\_frame\_timer = now

x, y = self.jerry\_running\_pos

rect = pygame.Rect(

GRID\_OFFSET\_X + y \* CELL\_SIZE,

GRID\_OFFSET\_Y + x \* CELL\_SIZE,

CELL\_SIZE, CELL\_SIZE

)

current\_frame = self.jerry\_running\_frames[self.jerry\_running\_frame\_index]

screen.blit(current\_frame, rect.topleft)

else:

self.show\_jerry\_running = False

def draw\_animated\_background(self):

screen.fill((230, 230, 255))

for star in self.stars:

star["y"] -= star["speed"]

star["size"] += 0.015

star["alpha"] += star["alpha\_direction"] \* 2

if star["alpha"] > 255:

star["alpha"] = 255

star["alpha\_direction"] = -1

elif star["alpha"] < 80:

star["alpha"] = 80

star["alpha\_direction"] = 1

if star["y"] < 0:

star["x"] = random.randint(0, WINDOW\_WIDTH)

star["y"] = WINDOW\_HEIGHT + random.randint(0, 100)

star["size"] = random.randint(2, 4)

star["speed"] = random.uniform(0.3, 0.8)

star["color"] = random.choice([(255,255,255), (255,230,200), (200,220,255), (255,255,180)])

surface = pygame.Surface((int(star["size"]\*2), int(star["size"]\*2)), pygame.SRCALPHA)

pygame.draw.circle(surface, star["color"] + (int(star["alpha"]),), (int(star["size"]), int(star["size"])), int(star["size"]))

screen.blit(surface, (int(star["x"] - star["size"]), int(star["y"] - star["size"])))

def draw\_ui(self):

# Place Main Menu and Next Round buttons in the same row at the top, with smaller padding

button\_y = 24

button\_width = 160

button\_height = 38

button\_spacing = 12

# Main Menu button (leftmost)

self.main\_menu\_button = pygame.Rect(24, button\_y, button\_width, button\_height)

pygame.draw.rect(screen, (200, 200, 255), self.main\_menu\_button, border\_radius=8)

pygame.draw.rect(screen, BLACK, self.main\_menu\_button, 3, border\_radius=8)

menu\_text = self.font.render("Main Menu", True, (0, 0, 0))

menu\_rect = menu\_text.get\_rect(center=self.main\_menu\_button.center)

screen.blit(menu\_text, menu\_rect)

# Next Round button (to the right of Main Menu), only show if game is over

if self.state == GameState.GAME\_OVER:

self.next\_round\_button = pygame.Rect(24 + button\_width + button\_spacing, button\_y, button\_width, button\_height)

pygame.draw.rect(screen, (200, 200, 255), self.next\_round\_button, border\_radius=8)

pygame.draw.rect(screen, BLACK, self.next\_round\_button, 3, border\_radius=8)

next\_text = self.font.render("Next Round", True, (0, 0, 0))

next\_rect = next\_text.get\_rect(center=self.next\_round\_button.center)

screen.blit(next\_text, next\_rect)

else:

self.next\_round\_button = None

# --- Display winner at the top center if game is over ---

if self.state == GameState.GAME\_OVER and self.winner:

win\_text = self.big\_font.render(f"{self.winner} Wins!", True, BLACK)

win\_rect = win\_text.get\_rect(center=(WINDOW\_WIDTH // 2, 40))

screen.blit(win\_text, win\_rect)

restart\_text = self.font.render("", True, BLACK)

restart\_rect = restart\_text.get\_rect(center=(WINDOW\_WIDTH // 2, 80))

screen.blit(restart\_text, restart\_rect)

# --- Move side text and action buttons to the right side ---

ui\_x = WINDOW\_WIDTH - 220

ui\_y = 80

line\_height = 32

if self.state == GameState.PLAYER1\_TURN or self.state == GameState.PLAYER2\_TURN:

# Show distance to Jerry for both players

if self.hidden\_pos is not None:

dist1 = self.a\_star\_distance(self.seeker1\_pos, self.hidden\_pos)

dist2 = self.a\_star\_distance(self.seeker2\_pos, self.hidden\_pos)

screen.blit(self.font.render(f"Tom -> Jerry: {dist1} steps", True, BLACK), (ui\_x, ui\_y))

screen.blit(self.font.render(f"Spike -> Jerry: {dist2} steps", True, BLACK), (ui\_x, ui\_y + line\_height))

screen.blit(self.font.render(f"Tom blocks: {self.player1\_blocks\_remaining}", True, BLACK), (ui\_x, ui\_y + 2 \* line\_height))

screen.blit(self.font.render(f"Spike blocks: {self.player2\_blocks\_remaining}", True, BLACK), (ui\_x, ui\_y + 3 \* line\_height))

# Show computer thinking indicator

if self.state == GameState.PLAYER2\_TURN and self.game\_mode != 'pvp':

screen.blit(self.font.render("Computer is thinking...", True, (255, 200, 0)), (ui\_x, ui\_y + 4 \* line\_height))

if not self.player2\_moved\_target:

screen.blit(self.font.render("(Can use Move Target)", True, (160, 32, 240)), (ui\_x, ui\_y + 5 \* line\_height))

# Draw Move Target button for current player if they haven't used it

action\_y = ui\_y + 6 \* line\_height

if self.state == GameState.PLAYER1\_TURN and not self.player1\_moved\_target:

self.move\_target\_button = pygame.Rect(ui\_x, action\_y, 180, 36)

pygame.draw.rect(screen, (255, 200, 0), self.move\_target\_button, border\_radius=8)

pygame.draw.rect(screen, BLACK, self.move\_target\_button, 3, border\_radius=8)

move\_text = self.font.render("Warn Jerry (Tom)", True, (0, 0, 0))

move\_rect = move\_text.get\_rect(center=self.move\_target\_button.center)

screen.blit(move\_text, move\_rect)

action\_y += 44

elif self.state == GameState.PLAYER2\_TURN and not self.player2\_moved\_target and self.game\_mode == 'pvp':

self.move\_target\_button = pygame.Rect(ui\_x, action\_y, 180, 36)

pygame.draw.rect(screen, (255, 200, 0), self.move\_target\_button, border\_radius=8)

pygame.draw.rect(screen, BLACK, self.move\_target\_button, 3, border\_radius=8)

move\_text = self.font.render("Warn Jerry (Spike)", True, (0, 0, 0))

move\_rect = move\_text.get\_rect(center=self.move\_target\_button.center)

screen.blit(move\_text, move\_rect)

action\_y += 44

else:

self.move\_target\_button = None

# Draw Place Block button for current player if they have blocks remaining

current\_player = 1 if self.state == GameState.PLAYER1\_TURN else 2

blocks\_remaining = self.player1\_blocks\_remaining if current\_player == 1 else self.player2\_blocks\_remaining

if blocks\_remaining > 0:

self.place\_block\_button = pygame.Rect(ui\_x, action\_y, 180, 36)

pygame.draw.rect(screen, (160, 32, 240), self.place\_block\_button, border\_radius=8)

pygame.draw.rect(screen, BLACK, self.place\_block\_button, 3, border\_radius=8)

player\_name = "Tom" if current\_player == 1 else "Spike"

block\_text = self.font.render(f"Place Block ({player\_name})", True, (0, 0, 0))

block\_rect = block\_text.get\_rect(center=self.place\_block\_button.center)

screen.blit(block\_text, block\_rect)

# Show current block orientation

orientation\_text = self.font.render(f"Orientation: {self.block\_orientation}", True, BLACK)

screen.blit(orientation\_text, (ui\_x, action\_y + 40))

screen.blit(self.font.render("Press 'R' to rotate", True, BLACK), (ui\_x, action\_y + 60))

# Show block placement instructions

if self.block\_placement\_mode:

screen.blit(self.font.render("Click on grid to place block", True, (160, 32, 240)), (ui\_x, action\_y + 80))

if self.block\_preview\_valid:

screen.blit(self.font.render("Green = Valid placement", True, GREEN), (ui\_x, action\_y + 100))

else:

screen.blit(self.font.render("Red = Invalid placement", True, RED), (ui\_x, action\_y + 100))

else:

self.place\_block\_button = None

# Feedback image and game over text remain on the right

if self.feedback\_text in self.feedback\_images:

screen.blit(self.feedback\_images[self.feedback\_text], (ui\_x, WINDOW\_HEIGHT - 240))

# Allow clicking main menu anytime

mouse\_pressed = pygame.mouse.get\_pressed()

if mouse\_pressed[0] and self.main\_menu\_button.collidepoint(pygame.mouse.get\_pos()):

pygame.mixer.music.stop()

self.state = GameState.MENU

self.show\_title\_screen()

# --- Display scores at the top right ---

score\_margin = 24

score\_text = None

if self.last\_game\_mode == 'pvp':

score\_text = self.font.render(f"Score: Tom {self.scores['Tom']} | Spike {self.scores['Spike']}", True, BLACK)

elif self.last\_game\_mode == 'pvc':

score\_text = self.font.render(f"Score: Tom {self.scores['Tom']} | Computer {self.scores['Computer']}", True, BLACK)

if score\_text:

score\_rect = score\_text.get\_rect(topright=(WINDOW\_WIDTH - score\_margin, score\_margin))

screen.blit(score\_text, score\_rect)

# --- Display debug message if present ---

if hasattr(self, 'debug\_message') and self.debug\_message:

debug\_font = pygame.font.SysFont("Consolas", 22)

debug\_text = debug\_font.render(self.debug\_message, True, (200, 0, 0))

screen.blit(debug\_text, (40, 80))

def move\_target\_to\_new\_location(self):

"""Move Jerry to a new random hiding location, but never to a position where a player is standing"""

if self.hiding\_spots:

self.jerry\_running\_pos = self.hidden\_pos

self.show\_jerry\_running = True

self.jerry\_running\_start\_time = pygame.time.get\_ticks()

self.jerry\_running\_frame\_index = 0

self.jerry\_running\_frame\_timer = pygame.time.get\_ticks()

# Choose a new location different from current and not occupied by a player

possible\_spots = [pos for pos in self.hiding\_spots if pos != self.hidden\_pos and pos != self.seeker1\_pos and pos != self.seeker2\_pos]

if not possible\_spots:

# If all other spots are occupied, fallback to any spot not occupied by a player

possible\_spots = [pos for pos in self.hiding\_spots if pos != self.seeker1\_pos and pos != self.seeker2\_pos]

if possible\_spots:

new\_pos = random.choice(possible\_spots)

self.hidden\_pos = new\_pos

# Update feedback for current player

if self.state == GameState.PLAYER1\_TURN:

dist = self.a\_star\_distance(self.seeker1\_pos, self.hidden\_pos)

self.feedback\_text = self.get\_feedback(dist)

elif self.state == GameState.PLAYER2\_TURN:

dist = self.a\_star\_distance(self.seeker2\_pos, self.hidden\_pos)

self.feedback\_text = self.get\_feedback(dist)

def start\_game(self):

pygame.mixer.music.load("sound\_track/backgroud\_music.mp3")

pygame.mixer.music.play(-1)

self.generate\_hiding\_spots()

self.hidden\_pos = random.choice(self.hiding\_spots)

self.seeker1\_pos = (0, 0)

self.seeker2\_pos = (GRID\_SIZE - 1, GRID\_SIZE - 1)

self.feedback\_text = ""

self.winner = None

self.tom\_direction = "idle"

self.spike\_direction = "idle"

self.state = GameState.PLAYER1\_TURN

self.player1\_moved\_target = False

self.player2\_moved\_target = False

# Reset block system

self.blocks = []

self.player1\_blocks\_remaining = 1

self.player2\_blocks\_remaining = 1

self.block\_placement\_mode = False

self.block\_orientation = "horizontal"

self.block\_preview\_pos = None

self.block\_preview\_valid = False

# Reset freeze state

self.player1\_frozen\_turns = 0

self.player2\_frozen\_turns = 0

self.player1\_unfreezing = False

self.player2\_unfreezing = False

self.player1\_unfreeze\_timer = 0

self.player2\_unfreeze\_timer = 0

# Reset gift box animation

self.gift\_box\_frame\_index = 0

self.gift\_box\_frame\_timer = pygame.time.get\_ticks()

# Track last game mode for score display (always set to current game mode)

if self.game\_mode:

self.last\_game\_mode = self.game\_mode

def computer\_place\_block(self):

"""Computer places a block strategically to interfere with player's path"""

if self.player2\_blocks\_remaining <= 0:

return False

player\_dist = self.a\_star\_distance(self.seeker1\_pos, self.hidden\_pos)

computer\_dist = self.a\_star\_distance(self.seeker2\_pos, self.hidden\_pos)

# Only place block if player is closer to Jerry than computer

if player\_dist >= computer\_dist:

return False

# For normal mode, be more aggressive with block placement

# Lower the threshold for "significant impact"

min\_impact = 1 if self.computer\_difficulty == "normal" else 1 # Hard mode can also place blocks with 1-step impact

# Try to block the player's path

best\_block = None

best\_impact = 0

best\_score = -1 # Higher score is better

for x in range(GRID\_SIZE):

for y in range(GRID\_SIZE):

# Try horizontal block

if self.can\_place\_block(x, y, "horizontal"):

# Check if this block would block the player's path

old\_player\_dist = self.a\_star\_distance(self.seeker1\_pos, self.hidden\_pos)

old\_computer\_dist = self.a\_star\_distance(self.seeker2\_pos, self.hidden\_pos)

# Temporarily place block

self.blocks.append((x, y, "horizontal"))

new\_player\_dist = self.a\_star\_distance(self.seeker1\_pos, self.hidden\_pos)

new\_computer\_dist = self.a\_star\_distance(self.seeker2\_pos, self.hidden\_pos)

self.blocks.pop() # Remove temporary block

# Calculate impact (how much it increases player's path)

player\_impact = new\_player\_dist - old\_player\_dist

computer\_impact = new\_computer\_dist - old\_computer\_dist

# Score this block placement

# Higher score if it blocks player more and doesn't block computer

score = player\_impact \* 2 - computer\_impact

# Hard mode gets bonus for strategic positioning

if self.computer\_difficulty == "hard" and self.hidden\_pos is not None:

# Bonus for significant impact

if player\_impact >= 3:

score += 2

# Bonus for blocking close to Jerry

target\_dist = abs(x - self.hidden\_pos[0]) + abs(y - self.hidden\_pos[1])

if target\_dist <= 2:

score += 1

if player\_impact > best\_impact and new\_player\_dist != float('inf') and score > best\_score:

best\_impact = player\_impact

best\_score = score

best\_block = (x, y, "horizontal")

# Try vertical block

if self.can\_place\_block(x, y, "vertical"):

# Check if this block would block the player's path

old\_player\_dist = self.a\_star\_distance(self.seeker1\_pos, self.hidden\_pos)

old\_computer\_dist = self.a\_star\_distance(self.seeker2\_pos, self.hidden\_pos)

# Temporarily place block

self.blocks.append((x, y, "vertical"))

new\_player\_dist = self.a\_star\_distance(self.seeker1\_pos, self.hidden\_pos)

new\_computer\_dist = self.a\_star\_distance(self.seeker2\_pos, self.hidden\_pos)

self.blocks.pop() # Remove temporary block

# Calculate impact (how much it increases player's path)

player\_impact = new\_player\_dist - old\_player\_dist

computer\_impact = new\_computer\_dist - old\_computer\_dist

# Score this block placement

# Higher score if it blocks player more and doesn't block computer

score = player\_impact \* 2 - computer\_impact

# Hard mode gets bonus for strategic positioning

if self.computer\_difficulty == "hard" and self.hidden\_pos is not None:

# Bonus for significant impact

if player\_impact >= 3:

score += 2

# Bonus for blocking close to Jerry

target\_dist = abs(x - self.hidden\_pos[0]) + abs(y - self.hidden\_pos[1])

if target\_dist <= 2:

score += 1

if player\_impact > best\_impact and new\_player\_dist != float('inf') and score > best\_score:

best\_impact = player\_impact

best\_score = score

best\_block = (x, y, "vertical")

# Place the best block if it has significant impact

if best\_block and best\_impact >= min\_impact:

x, y, orientation = best\_block

self.place\_block(x, y, orientation, 2)

return True

# If no good strategic block found, try to block on the player's likely path

player\_path = self.get\_player\_likely\_path()

if len(player\_path) > 1: # If we have a path to block

# Try to block on the next few steps of the player's path

for i in range(1, min(4, len(player\_path))): # Look at next 3 steps

path\_pos = player\_path[i]

# Try horizontal block on path

if self.can\_place\_block(path\_pos[0], path\_pos[1], "horizontal"):

# Check if this actually blocks the player's path

old\_player\_dist = self.a\_star\_distance(self.seeker1\_pos, self.hidden\_pos)

self.blocks.append((path\_pos[0], path\_pos[1], "horizontal"))

new\_player\_dist = self.a\_star\_distance(self.seeker1\_pos, self.hidden\_pos)

self.blocks.pop()

if new\_player\_dist > old\_player\_dist and new\_player\_dist != float('inf'):

self.place\_block(path\_pos[0], path\_pos[1], "horizontal", 2)

return True

# Try vertical block on path

if self.can\_place\_block(path\_pos[0], path\_pos[1], "vertical"):

# Check if this actually blocks the player's path

old\_player\_dist = self.a\_star\_distance(self.seeker1\_pos, self.hidden\_pos)

self.blocks.append((path\_pos[0], path\_pos[1], "vertical"))

new\_player\_dist = self.a\_star\_distance(self.seeker1\_pos, self.hidden\_pos)

self.blocks.pop()

if new\_player\_dist > old\_player\_dist and new\_player\_dist != float('inf'):

self.place\_block(path\_pos[0], path\_pos[1], "vertical", 2)

return True

# Fallback: try to block near the player's current position

if player\_dist <= 4: # Increased range for normal mode

# Try to block in the direction the player is likely to move

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

nx, ny = self.seeker1\_pos[0] + dx, self.seeker1\_pos[1] + dy

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

# Try horizontal block near player

if self.can\_place\_block(nx, ny, "horizontal"):

# Check if this actually blocks the player's path

old\_player\_dist = self.a\_star\_distance(self.seeker1\_pos, self.hidden\_pos)

self.blocks.append((nx, ny, "horizontal"))

new\_player\_dist = self.a\_star\_distance(self.seeker1\_pos, self.hidden\_pos)

self.blocks.pop()

if new\_player\_dist > old\_player\_dist and new\_player\_dist != float('inf'):

self.place\_block(nx, ny, "horizontal", 2)

return True

# Try vertical block near player

if self.can\_place\_block(nx, ny, "vertical"):

# Check if this actually blocks the player's path

old\_player\_dist = self.a\_star\_distance(self.seeker1\_pos, self.hidden\_pos)

self.blocks.append((nx, ny, "vertical"))

new\_player\_dist = self.a\_star\_distance(self.seeker1\_pos, self.hidden\_pos)

self.blocks.pop()

if new\_player\_dist > old\_player\_dist and new\_player\_dist != float('inf'):

self.place\_block(nx, ny, "vertical", 2)

return True

return False

def get\_player\_likely\_path(self):

"""Get the likely path the player will take to reach Jerry"""

if not hasattr(self, '\_player\_path\_cache') or self.\_player\_path\_cache[0] != (self.seeker1\_pos, self.hidden\_pos):

# Use A\* to find the shortest path from player to Jerry

path = self.a\_star\_path(self.seeker1\_pos, self.hidden\_pos)

self.\_player\_path\_cache = ((self.seeker1\_pos, self.hidden\_pos), path)

return self.\_player\_path\_cache[1]

def a\_star\_path(self, start, goal):

"""A\* algorithm that returns the actual path, not just distance"""

if start == goal:

return [start]

open\_set = [(0, start)]

came\_from = {}

g\_score = {start: 0}

f\_score = {start: abs(start[0]-goal[0]) + abs(start[1]-goal[1])}

while open\_set:

\_, current = heapq.heappop(open\_set)

if current == goal:

# Reconstruct path

path = []

while current in came\_from:

path.append(current)

current = came\_from[current]

path.append(start)

path.reverse()

return path

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

neighbor = (current[0]+dx, current[1]+dy)

if 0 <= neighbor[0] < GRID\_SIZE and 0 <= neighbor[1] < GRID\_SIZE:

if self.is\_position\_blocked(neighbor):

continue

temp\_g\_score = g\_score[current] + 1

if neighbor not in g\_score or temp\_g\_score < g\_score[neighbor]:

came\_from[neighbor] = current

g\_score[neighbor] = temp\_g\_score

f\_score[neighbor] = temp\_g\_score + abs(neighbor[0]-goal[0]) + abs(neighbor[1]-goal[1])

heapq.heappush(open\_set, (f\_score[neighbor], neighbor))

return [] # No path found

def run(self):

self.show\_title\_screen()

running = True

player\_turn = 1 # 1 for Tom, 2 for Spike (in PvP)

while running:

for event in pygame.event.get():

if event.type == pygame.QUIT:

running = False

elif event.type == pygame.KEYDOWN:

if self.state == GameState.MENU:

self.start\_game()

player\_turn = 1

elif self.state == GameState.GAME\_OVER:

# Do nothing on key press after game over; wait for Next Round button

pass

elif self.state == GameState.PLAYER1\_TURN:

# Block rotation

if event.key == pygame.K\_r:

self.block\_orientation = "vertical" if self.block\_orientation == "horizontal" else "horizontal"

# Movement with block checking

x, y = self.seeker1\_pos

moved = False

if event.key == pygame.K\_UP and x > 0:

new\_pos = (x - 1, y)

if not self.is\_position\_blocked(new\_pos):

self.seeker1\_pos = new\_pos

self.tom\_direction = "up"

moved = True

elif event.key == pygame.K\_DOWN and x < GRID\_SIZE - 1:

new\_pos = (x + 1, y)

if not self.is\_position\_blocked(new\_pos):

self.seeker1\_pos = new\_pos

self.tom\_direction = "down"

moved = True

elif event.key == pygame.K\_LEFT and y > 0:

new\_pos = (x, y - 1)

if not self.is\_position\_blocked(new\_pos):

self.seeker1\_pos = new\_pos

self.tom\_direction = "left"

moved = True

elif event.key == pygame.K\_RIGHT and y < GRID\_SIZE - 1:

new\_pos = (x, y + 1)

if not self.is\_position\_blocked(new\_pos):

self.seeker1\_pos = new\_pos

self.tom\_direction = "right"

moved = True

if moved:

# Check for gift box collection

if self.gift\_box\_location and self.seeker1\_pos == self.gift\_box\_location:

self.gift\_box\_popping = True

self.gift\_box\_pop\_position = self.gift\_box\_location

self.gift\_box\_pop\_frame\_index = 0

self.gift\_box\_pop\_frame\_timer = pygame.time.get\_ticks()

self.gift\_box\_location = None

self.freeze\_opponent(1)

if self.seeker1\_pos == self.hidden\_pos:

self.winner = "Tom (Player 1)"

if self.last\_game\_mode == 'pvp':

self.scores["Tom"] += 1

else:

self.scores["Tom"] += 1

pygame.mixer.music.stop()

pygame.mixer.music.load("sound\_track/win.wav")

pygame.mixer.music.play()

self.state = GameState.GAME\_OVER

else:

dist = self.a\_star\_distance(self.seeker1\_pos, self.hidden\_pos)

self.feedback\_text = self.get\_feedback(dist)

if self.game\_mode == 'pvp':

self.state = GameState.PLAYER2\_TURN

else:

self.state = GameState.PLAYER2\_TURN

elif self.state == GameState.PLAYER2\_TURN:

if self.game\_mode == 'pvp':

# Block rotation

if event.key == pygame.K\_r:

self.block\_orientation = "vertical" if self.block\_orientation == "horizontal" else "horizontal"

# Movement with block checking

x, y = self.seeker2\_pos

moved = False

if event.key == pygame.K\_w and x > 0:

new\_pos = (x - 1, y)

if not self.is\_position\_blocked(new\_pos):

self.seeker2\_pos = new\_pos

self.spike\_direction = "up"

moved = True

elif event.key == pygame.K\_s and x < GRID\_SIZE - 1:

new\_pos = (x + 1, y)

if not self.is\_position\_blocked(new\_pos):

self.seeker2\_pos = new\_pos

self.spike\_direction = "down"

moved = True

elif event.key == pygame.K\_a and y > 0:

new\_pos = (x, y - 1)

if not self.is\_position\_blocked(new\_pos):

self.seeker2\_pos = new\_pos

self.spike\_direction = "left"

moved = True

elif event.key == pygame.K\_d and y < GRID\_SIZE - 1:

new\_pos = (x, y + 1)

if not self.is\_position\_blocked(new\_pos):

self.seeker2\_pos = new\_pos

self.spike\_direction = "right"

moved = True

if moved:

# Check for gift box collection

if self.gift\_box\_location and self.seeker2\_pos == self.gift\_box\_location:

self.gift\_box\_popping = True

self.gift\_box\_pop\_position = self.gift\_box\_location

self.gift\_box\_pop\_frame\_index = 0

self.gift\_box\_pop\_frame\_timer = pygame.time.get\_ticks()

self.gift\_box\_location = None

self.freeze\_opponent(2)

if self.seeker2\_pos == self.hidden\_pos:

self.winner = "Spike (Player 2)"

if self.last\_game\_mode == 'pvp':

self.scores["Spike"] += 1

elif self.last\_game\_mode == 'pvc':

self.scores["Computer"] += 1

pygame.mixer.music.stop()

if self.game\_mode == 'pvp':

pygame.mixer.music.load("sound\_track/spike\_win.wav")

else:

pygame.mixer.music.load("sound\_track/lose.mp3")

pygame.mixer.music.play()

self.state = GameState.GAME\_OVER

else:

dist = self.a\_star\_distance(self.seeker2\_pos, self.hidden\_pos)

self.feedback\_text = self.get\_feedback(dist)

self.state = GameState.PLAYER1\_TURN

elif event.type == pygame.MOUSEBUTTONDOWN:

if self.next\_round\_button and self.next\_round\_button.collidepoint(event.pos):

self.start\_game()

player\_turn = 1

elif self.state == GameState.GAME\_OVER and self.main\_menu\_button and self.main\_menu\_button.collidepoint(event.pos):

# No need to reset here anymore

self.state = GameState.MENU

self.show\_title\_screen()

elif self.move\_target\_button and self.move\_target\_button.collidepoint(event.pos):

if self.state == GameState.PLAYER1\_TURN and not self.player1\_moved\_target:

self.move\_target\_to\_new\_location()

self.player1\_moved\_target = True

# End turn and switch to player 2

if self.game\_mode == 'pvp':

self.state = GameState.PLAYER2\_TURN

else:

self.state = GameState.PLAYER2\_TURN

elif self.state == GameState.PLAYER2\_TURN and not self.player2\_moved\_target:

self.move\_target\_to\_new\_location()

self.player2\_moved\_target = True

# End turn and switch to player 1

self.state = GameState.PLAYER1\_TURN

elif self.place\_block\_button and self.place\_block\_button.collidepoint(event.pos):

# Enter block placement mode

self.block\_placement\_mode = True

elif self.block\_placement\_mode:

# Handle block placement

mouse\_x, mouse\_y = event.pos

if mouse\_x >= GRID\_OFFSET\_X and mouse\_y >= GRID\_OFFSET\_Y: # Click is on the grid

grid\_x = (mouse\_y - GRID\_OFFSET\_Y) // CELL\_SIZE

grid\_y = (mouse\_x - GRID\_OFFSET\_X) // CELL\_SIZE

if 0 <= grid\_x < GRID\_SIZE and 0 <= grid\_y < GRID\_SIZE:

current\_player = 1 if self.state == GameState.PLAYER1\_TURN else 2

if self.place\_block(grid\_x, grid\_y, self.block\_orientation, current\_player):

self.block\_placement\_mode = False

self.block\_preview\_pos = None

# End turn after placing block

if self.game\_mode == 'pvp':

if self.state == GameState.PLAYER1\_TURN:

self.state = GameState.PLAYER2\_TURN

else:

self.state = GameState.PLAYER1\_TURN

else:

if self.state == GameState.PLAYER1\_TURN:

self.state = GameState.PLAYER2\_TURN

elif event.type == pygame.MOUSEMOTION:

# Update block preview position

if self.block\_placement\_mode:

mouse\_x, mouse\_y = event.pos

if mouse\_x >= GRID\_OFFSET\_X and mouse\_y >= GRID\_OFFSET\_Y: # Mouse is on the grid

grid\_x = (mouse\_y - GRID\_OFFSET\_Y) // CELL\_SIZE

grid\_y = (mouse\_x - GRID\_OFFSET\_X) // CELL\_SIZE

if 0 <= grid\_x < GRID\_SIZE and 0 <= grid\_y < GRID\_SIZE:

self.block\_preview\_pos = (grid\_x, grid\_y)

self.block\_preview\_valid = self.can\_place\_block(grid\_x, grid\_y, self.block\_orientation)

else:

self.block\_preview\_pos = None

else:

self.block\_preview\_pos = None

# Handle freezing and skipping turns

# Player 1 frozen logic

if self.state == GameState.PLAYER1\_TURN and self.player1\_frozen\_turns > 0:

self.player1\_frozen\_turns -= 1

if self.player1\_frozen\_turns == 0:

self.player1\_unfreezing = True

self.player1\_unfreeze\_timer = pygame.time.get\_ticks()

self.state = GameState.PLAYER2\_TURN if self.game\_mode == 'pvp' else GameState.PLAYER2\_TURN

continue

# Player 2 frozen logic

if self.state == GameState.PLAYER2\_TURN and self.player2\_frozen\_turns > 0:

self.player2\_frozen\_turns -= 1

if self.player2\_frozen\_turns == 0:

self.player2\_unfreezing = True

self.player2\_unfreeze\_timer = pygame.time.get\_ticks()

self.state = GameState.PLAYER1\_TURN

continue

# --- Computer collects gift box logic ---

if self.state == GameState.PLAYER2\_TURN and self.game\_mode != 'pvp':

if self.gift\_box\_location and self.seeker2\_pos == self.gift\_box\_location:

self.gift\_box\_popping = True

self.gift\_box\_pop\_position = self.gift\_box\_location

self.gift\_box\_pop\_frame\_index = 0

self.gift\_box\_pop\_frame\_timer = pygame.time.get\_ticks()

self.gift\_box\_location = None

self.freeze\_opponent(2)

if self.state == GameState.PLAYER2\_TURN and self.game\_mode != 'pvp':

if not self.computer\_thinking:

self.computer\_thinking = True

self.computer\_think\_time = pygame.time.get\_ticks()

elif pygame.time.get\_ticks() - self.computer\_think\_time > 800: # 800ms delay

self.computer\_move()

self.computer\_thinking = False

screen.fill(LIGHT\_GREEN)

self.draw\_animated\_background()

if self.state != GameState.MENU:

self.draw\_grid()

self.draw\_ui()

pygame.display.flip()

self.clock.tick(FPS)

pygame.quit()

sys.exit()

def freeze\_opponent(self, player):

# player: 1 or 2 (the one who collected the gift)

if player == 1:

self.player2\_frozen\_turns = 2

else:

self.player1\_frozen\_turns = 2

if \_\_name\_\_ == "\_\_main\_\_":

HideSeekGame().run()