אולג אולוקוב 324320712
רוסלן וסילייב 319333001
פיליפ שלייפר 322357781

מסמך מסכם

לינק לסרטון על המצגת-
https://youtu.be/qP-GKfrg2LU

לינק לסרטון על המשחק-
https://youtu.be/gElVKFb2WnI

לינק לגיטהאב-
https://github.com/RoslanVasilew/AdvancedAlgorithmGame

בפרויקט זה, יצרנו משחק בשם "תופסת שחמט" באמצעות הספרייה Pygame.

המשחק משלב אלמנטים של לוח שחמט עם אלמנטים של התחמקות ומניפולציה על מכשולים בשילוב עם כוחות מיוחדים.

הבעיה:

הבעיה שהמשחק מציב בפני השחקן היא להצליח להוביל את הדמות שלו מפסגת לוח השחמט אל התחתית מבלי להיתפס על ידי חלקי השחמט המייצגים אויבים. השחקן צריך להתחמק מהאויבים, לעקוף מכשולים, ולנצל את היכולות המיוחדות שמוקצות לו.

פתרון:

- השתמשנו במספר אלגוריתמים ותהליכים כדי לפתח את המשחק:

1. **אלגוריתם חיפוש מינימקס עם גיזום אלפא-ביתא (Minimax with Alpha-Beta Pruning)**: כדי לחשב את מהלכי חלקי השחמט היריבים

ולמקסם את הנזק לשחקן. האלגוריתם מעריך מהלכים אפשריים לפי מרחק ממיקום השחקן ומכשולים בשטח.

2. **Heuristic Function**: פונקציה המשמשת להערכת מצבים לפי מיקום השחקן, מרחק ממכשולים, וקירבה לרכבת שיכולה לדחוף אותם מהמפה.
3. *A Element*: שילוב של חישוב (Manhattan Distance) כדי לחשב מרחקי מינימום בין אויבים לשחקן ולהעדיף מהלכים שמקרבים את האויבים לשחקן.

אלגוריתמים ותהליכים נוספים:

- **מניפולציה של מכשולים**: והשחקן יכול להניע מכשולים או להקפיא אוייבים, בהתאם ליכולות שקיבל.
- **חישוב מיקום מחדש של מכשולים**: יצרנו פונקציה שמוודאת שהמכשולים לא ייווצרו במיקום שייצור מצב בלתי אפשרי למשחק או חפיפות עם רכבת הנעה על המפה.

הקוד:

```
import pygame
import sys
import random
import math
from collections import namedtuple

# Initialize Pygame
pygame.init()
size = width, height = 800, 800  # Size of the window
screen = pygame.display.set_mode(size)
pygame.display.set_caption('Chess Evasion Game')

# Define constants
BOARD_SIZE = 600  # Size of the chess board
BLOCK_SIZE = BOARD_SIZE // 8
BUTTON_WIDTH = 200
BUTTON_HEIGHT = 50
BUTTON_MARGIN = 20
ABILITY_TEXT_SIZE = 24
NUM_OBSTACLES = 5  # Number of obstacles
TRAIN_WIDTH = 2  # Train is 2 blocks wide
```

```
                                                    Load ability icons #
                    ('PUSH_ICON = pygame.image.load('Assets/mighty-force.png
                      ('FREEZE_ICON = pygame.image.load('Assets/ice-cube.png
            ('OBSTACLE_SHIFT_ICON = pygame.image.load('Assets/earth-spit.png
                 ('DOUBLE_MOVE_ICON = pygame.image.load('Assets/kangaroo.png

                                               Resize icons to fit the buttons #
ICON_SIZE = (BUTTON_HEIGHT - 10, BUTTON_HEIGHT - 10)  # Slightly smaller than
                                                                    the button
                    (PUSH_ICON = pygame.transform.scale(PUSH_ICON, ICON_SIZE
                (FREEZE_ICON = pygame.transform.scale(FREEZE_ICON, ICON_SIZE
 (OBSTACLE_SHIFT_ICON = pygame.transform.scale(OBSTACLE_SHIFT_ICON, ICON_SIZE
       (DOUBLE_MOVE_ICON = pygame.transform.scale(DOUBLE_MOVE_ICON, ICON_SIZE

                                                          Load icons #
                                ('RAM_ICON = pygame.image.load('Assets/ram.svg
        ('WARLORD_HELMET_ICON = pygame.image.load('Assets/warlord-helmet.svg
                            ('PLAYER_ICON = pygame.image.load('Assets/crown.png
                ('TRAIN_ICON = pygame.image.load('Assets/steam-locomotive.png

                                           Resize the icons to fit your pieces #
        ((RAM_ICON = pygame.transform.scale(RAM_ICON, (BLOCK_SIZE, BLOCK_SIZE
WARLORD_HELMET_ICON = pygame.transform.scale(WARLORD_HELMET_ICON, (BLOCK_SIZE,
                                                                  ((BLOCK_SIZE
   ((PLAYER_ICON = pygame.transform.scale(PLAYER_ICON, (BLOCK_SIZE, BLOCK_SIZE
 TRAIN_ICON = pygame.transform.scale(TRAIN_ICON, (BLOCK_SIZE, BLOCK_SIZE))  #
                                                        Resize to fit one block

                                                                  Colors #
                                              ("WHITE = pygame.Color("white
                                                ("GRAY = pygame.Color("gray
                                                ("BLUE = pygame.Color("blue
                                                  ("RED = pygame.Color("red
                                                ("GOLD = pygame.Color("gold
                                              ("BLACK = pygame.Color("black
                                              ("GREEN = pygame.Color("green
                                          ("DARK_GRAY = pygame.Color("darkgray

                                                Define an Ability namedtuple #
                    (['Ability = namedtuple('Ability', ['name', 'icon', 'key

                                                Define all available abilities #
                                                                  ] = ALL_ABILITIES
                                ,(Ability('Freeze', FREEZE_ICON, pygame.K_r
                                    ,(Ability('Push', PUSH_ICON, pygame.K_t
                    ,(Ability('Double Move', DOUBLE_MOVE_ICON, pygame.K_y
                (Ability('Obstacle Shift', OBSTACLE_SHIFT_ICON, pygame.K_u
                                                                        [
```

```
                                    Define the board drawing function #
                                                :(def draw_board(screen
                                                                    """
          .Draws the chess board on the screen with alternating colors
                                                                    """
                                                [colors = [WHITE, GRAY
                                                  :(for row in range(8
                                              :(for col in range(8
                                  [color = colors[(row + col) % 2
      pygame.draw.rect(screen, color, (col * BLOCK_SIZE, row *
                                        ((BLOCK_SIZE, BLOCK_SIZE, BLOCK_SIZE

                    :(def spawn_or_respawn_obstacles(player, pieces, train
                                                                    """
  Spawns or respawns obstacles on the board avoiding player, pieces, and
                                                        .train positions
                                                                    """
                                                      global obstacles
                                                    [] = new_obstacles
    [(available_positions = [(x, y) for x in range(8) for y in range(8

                                             Remove occupied positions #
  occupied_positions = [player.position] + [piece.position for piece in
                                                                  [pieces
                                                               :if train
 occupied_positions.extend([(train.position[0], train.position[1]),
                                  ([([(train.position[0] + 1, train.position[1

available_positions = [pos for pos in available_positions if pos not in
                                                    [occupied_positions

                                          Spawn NUM_OBSTACLES obstacles #
                                           :(for _ in range(NUM_OBSTACLES
                                              :if available_positions
                        (pos = random.choice(available_positions
                             ((new_obstacles.append(Obstacle(pos
                                 (available_positions.remove(pos

                                               obstacles = new_obstacles

                                        :(def draw_abilities_window(screen
                                                                    """
                     .Draws the abilities window with buttons and icons
                                                                    """
          ability_window_rect = pygame.Rect(0, height - BUTTON_HEIGHT -
                        (BUTTON_MARGIN, width, BUTTON_HEIGHT + BUTTON_MARGIN
                    (pygame.draw.rect(screen, BLACK, ability_window_rect

                         (font = pygame.font.Font(None, ABILITY_TEXT_SIZE
```

```
                            Helper function to draw a button with an icon #
                            :(def draw_button_with_icon(rect, ability, used
          (pygame.draw.rect(screen, DARK_GRAY if used else WHITE, rect
              ((screen.blit(ability.icon, (rect.left + 5, rect.top + 5
                 (text_surface = font.render(ability.name, True, BLACK
  text_rect = text_surface.get_rect(midleft=(rect.left + ICON_SIZE[0] +
                                                        ((10, rect.centery
                                  (screen.blit(text_surface, text_rect

                                                      [] = button_rects
                           :(for i, ability in enumerate(available_abilities
        button_rect = pygame.Rect(BUTTON_MARGIN + i * (BUTTON_WIDTH +
                                                             ,(BUTTON_MARGIN
    height - BUTTON_HEIGHT - BUTTON_MARGIN +
                                                              ,BUTTON_MARGIN
                (BUTTON_WIDTH, BUTTON_HEIGHT
                                      [used = ability_used[ability.name
                     (draw_button_with_icon(button_rect, ability, used
                                      (button_rects.append(button_rect

                                                  return button_rects

                        :(def display_message(screen, message, color, position
                                                                        """
                                        .Displays a message on the screen
                                                                        """
    screen.fill(BLACK)  # Fill screen with black before displaying message
                                      (font = pygame.font.Font(None, 74
                                   (text = font.render(message, True, color
                                      (rect = text.get_rect(center=position
                                                 (screen.blit(text, rect
                                                 ()pygame.display.update

                                                :(def reset_game(difficulty
                                                                        """
                            .Resets the game with the given difficulty level
                                                                        """
global player, pieces, player_turn, game_over, available_abilities, train,
                                                       obstacles, ability_used

player = Player((random.randint(0, 7), 0))  # Randomly spawn player on top
                                                                      edge


                                                    :"if difficulty == "easy
      [('pieces = [ChessPiece((random.randint(0, 7), 7), 'directional
                                                         train = None
                                                      [] = obstacles
                                                :"elif difficulty == "medium
```

```
                                                        ] = pieces
            ,('ChessPiece((random.randint(0, 7), 7), 'directional
                ('ChessPiece((random.randint(0, 7), 7), 'diagonal
                                                              [
                                                    ()train = Train
                                                     [] = obstacles
                                                          else:  # hard
                                                        ] = pieces
            ,('ChessPiece((random.randint(0, 7), 7), 'directional
               ,('ChessPiece((random.randint(0, 7), 7), 'diagonal
             ('ChessPiece((random.randint(0, 7), 7), 'directional
                                                              [
                                                    ()train = Train

              Make sure obstacles do not spawn on the train's path #
                    (spawn_or_respawn_obstacles(player, pieces, train

                                                   player_turn = True
                                                    game_over = False

                                           Randomly select 2 abilities #
                    (available_abilities = random.sample(ALL_ABILITIES, 2

                                         Initialize ability usage tracking #
    {ability_used = {ability.name: False for ability in available_abilities

                                                             Player class #
                                                              :class Player
                                                :(def __init__(self, position
                                                 self.position = position
                                                        self.color = BLUE
                                              self.double_move_active = False
                                                      self.icon = PLAYER_ICON

                                                          :(def draw(self, screen
                                                                        """
                                             .Draws the player on the screen
                                                                        """
                                      ,pygame.draw.rect(screen, self.color
    self.position[0] * BLOCK_SIZE, self.position[1] *)
                                                ((BLOCK_SIZE, BLOCK_SIZE, BLOCK_SIZE
screen.blit(self.icon, (self.position[0] * BLOCK_SIZE, self.position[1]
                                                                    ((* BLOCK_SIZE

        :(def move(self, direction, pieces, obstacles, train, board_size=8
                                                                        """
                                    .Moves the player in the given direction
                                                                        """
                                               :if self.double_move_active
```

```
      moved = self._move_once(direction, pieces, obstacles, train,
                                                                  (board_size
                                                          :if moved
          self._move_once(direction, pieces, obstacles, train,
                                                                  (board_size
                                      self.double_move_active = False
                                                                :else
      moved = self._move_once(direction, pieces, obstacles, train,
                                                                  (board_size
                                                           return moved

    :(def _move_once(self, direction, pieces, obstacles, train, board_size
                                                                    """
                        .Moves the player one step in the given direction
                                                                    """
                                                    x, y = self.position
                                                        } = move_offsets
  ,(UP': (0, -1), 'DOWN': (0, 1), 'LEFT': (-1, 0), 'RIGHT': (1, 0'
   UP_LEFT': (-1, -1), 'UP_RIGHT': (1, -1), 'DOWN_LEFT': (-1, 1),'
                                                   ('DOWN_RIGHT': (1, 1
                                                                   {
                            ((dx, dy = move_offsets.get(direction, (0, 0
                                          new_x, new_y = x + dx, y + dy

                :if 0 <= new_x < board_size and 0 <= new_y < board_size
  if not any(piece.position == (new_x, new_y) for piece in pieces)
                                                                    \ and
not any(obstacle.position == (new_x, new_y) for obstacle in
                                                        \ obstacles) and
  train is None or not train.is_occupied((new_x, new_y))))
                                                                    \ and
    train is None or not self.is_blocked_by_train((x, y),)
                                                  :(((new_x, new_y), train
                                (self.position = (new_x, new_y
                                                 return True
                                                        return False

                          :(def is_blocked_by_train(self, start, end, train
                                                                    """
                          .Checks if the player is blocked by the train
                                                                    """
                                                      x1, y1 = start
                                                        x2, y2 = end
                                                tx, ty = train.position

if y1 == y2 == ty:  # Moving horizontally on the same row as the train
                                                (min_x = min(x1, x2
                                                (max_x = max(x1, x2
           return min_x <= tx <= max_x or min_x <= tx + 1 <= max_x
```

```
                                        return False  # Not blocked

                                                    ChessPiece class #
                                                     :class ChessPiece
                                    :(def __init__(self, position, type
                                           self.position = position
                                                self.type = type
                            self.color = RED if type != 'flag' else GOLD
                                                 self.frozen = False
                                              self.frozen_turns = 0
    self.icon = RAM_ICON if type == 'diagonal' else WARLORD_HELMET_ICON

                                                 :(def draw(self, screen
                                                                """
                                        .Draws the chess piece on the screen
                                                                """
                                        ,pygame.draw.rect(screen, self.color
    self.position[0] * BLOCK_SIZE, self.position[1] *)
                                            ((BLOCK_SIZE, BLOCK_SIZE, BLOCK_SIZE
screen.blit(self.icon, (self.position[0] * BLOCK_SIZE, self.position[1]
                                                                ((* BLOCK_SIZE

                                        :(def possible_moves(self, board_size=8
                                                                """
                    .Returns a list of possible moves for the chess piece
                                                                """
                                                x, y = self.position
                                                              [] = moves
                                            :'if self.type == 'directional
                    [(move_offsets = [(0, 1), (0, -1), (1, 0), (-1, 0
                                             :'elif self.type == 'diagonal
                  [(move_offsets = [(1, 1), (1, -1), (-1, 1), (-1, -1
                                                                  :else
                                                 [] = move_offsets

                                              :for dx, dy in move_offsets
                                              nx, ny = x + dx, y + dy
                    :if 0 <= nx < board_size and 0 <= ny < board_size
                                          ((moves.append((nx, ny
                                                        return moves

        def calculate_move(self, player_position, pieces, obstacles, train,
                                                                  :(depth=5
                                                                """
    Calculates the best move for the chess piece using minimax algorithm
                                                    .with alpha-beta pruning
                                                                """
                                                   :if self.frozen
```

```
                                            return self.position

best_move = self.minimax_alpha_beta(depth, float('-inf'), float('inf'),
                                            ,True, player_position, pieces
                (obstacles, train

                        Ensure best_move is a valid position tuple #
            :if isinstance(best_move, tuple) and len(best_move) == 2
                                              return best_move
                                                          :else
If best_move is not a valid position, return the current position #
                                            return self.position

       def minimax_alpha_beta(self, depth, alpha, beta, maximizing_player,
                                  :(player_position, pieces, obstacles, train
                                                                      """
            .Implements the minimax algorithm with alpha-beta pruning
                                                                      """
            :(if depth == 0 or self.is_terminal_state(player_position
  (return self.evaluate_position(player_position, obstacles, train

                                          ()moves = self.possible_moves
    moves.sort(key=lambda move: self.heuristic(move, player_position,
                                (obstacles, train), reverse=maximizing_player

                                             :if maximizing_player
                                      ('max_eval = float('-inf
         best_move = self.position  # Default to current position
                                            :for move in moves
       :(if self.is_valid_move(move, pieces, obstacles, train
    eval = self.minimax_alpha_beta(depth - 1, alpha, beta,
                                   ,False, player_position, pieces, obstacles
                    (train
                                 :(if isinstance(eval, tuple
  (eval = self.evaluate_position(eval, obstacles, train
                                     :if eval > max_eval
                                    max_eval = eval
                                  best_move = move
                                (alpha = max(alpha, eval
                                       :if beta <= alpha
                                             break
                    return best_move if depth == 5 else max_eval
                                                          :else
                                      ('min_eval = float('inf
                                            :for move in moves
       :(if self.is_valid_move(move, pieces, obstacles, train
    eval = self.minimax_alpha_beta(depth - 1, alpha, beta,
                                   ,True, player_position, pieces, obstacles
                    (train
```

```
                                :(if isinstance(eval, tuple
 (eval = self.evaluate_position(eval, obstacles, train
                            (min_eval = min(min_eval, eval
                                    (beta = min(beta, eval
                                         :if beta <= alpha
                                                break
                                                  return min_eval

                  :(def is_valid_move(self, move, pieces, obstacles, train
                                                                    """
                                           .Checks if the move is valid
                                                                    """
 return not any(piece.position == move for piece in pieces if piece !=
                                                                 \ self) and
\ not any(obstacle.position == move for obstacle in obstacles) and
                          ((not train or not train.is_occupied(move)

                               :(def is_terminal_state(self, player_position
                                                                    """
                         .Checks if the current state is a terminal state
                                                                    """
                                  return self.position == player_position

           :(def evaluate_position(self, player_position, obstacles, train
                                                                    """
    Evaluates the position based on distance to player, obstacles, and
                                                                .train
                                                                    """
    (distance = self.manhattan_distance(self.position, player_position
                                               )obstacle_penalty = sum
                                        for obstacle in obstacles if 5
                (self.manhattan_distance(self.position, obstacle.position) < 2
train_penalty = 10 if train and self.manhattan_distance(self.position,
                                                    train.position) < 2 else 0

                A* element: Prefer moves that are closer to the player #
                                           a_star_score = -distance * 2

                  return a_star_score - obstacle_penalty - train_penalty

              :(def heuristic(self, move, player_position, obstacles, train
                                                                    """
                                .Calculates the heuristic value of a move
                                                                    """
              (distance = self.manhattan_distance(move, player_position
                  obstacle_penalty = sum(5 for obstacle in obstacles if
                           (self.manhattan_distance(move, obstacle.position) < 2
         train_penalty = 10 if train and self.manhattan_distance(move,
                                                    train.position) < 2 else 0
```

```
            A* element: Prefer moves that are closer to the player #
                                    a_star_score = -distance * 2

              Prefer moves that are in the direction of the player #
        (direction_score = self.direction_score(move, player_position

           return a_star_score + direction_score - obstacle_penalty -
                                                                train_penalty

                                                          staticmethod@
                                      :(def manhattan_distance(pos1, pos2
                                                                """
             .Calculates the Manhattan distance between two positions
                                                                """
               ([return abs(pos1[0] - pos2[0]) + abs(pos1[1] - pos2[1

                          :(def direction_score(self, move, player_position
                                                                """
                             .Calculates the direction score for a move
                                                                """
            current_distance = self.manhattan_distance(self.position,
                                                             (player_position
        (new_distance = self.manhattan_distance(move, player_position
                  return 10 if new_distance < current_distance else -5

                                        :(def update_frozen_status(self
                                                                """
                        .Updates the frozen status of the chess piece
                                                                """
                                                  :if self.frozen
                                       self.frozen_turns += 1
                                    :if self.frozen_turns >= 2
                                       self.frozen = False
                                     self.frozen_turns = 0

  :(def push_away(self, player_position, pieces, obstacles, board_size=8
                                                                """
                          .Pushes the chess piece away from the player
                                                                """
                                                    ) = direction
                             ,[self.position[0] - player_position[0
                              [self.position[1] - player_position[1
                                                                  (
                ((([magnitude = max(abs(direction[0]), abs(direction[1
                                                  :if magnitude != 0
  (direction = (direction[0] / magnitude, direction[1] / magnitude

                      (new_x = self.position[0] + int(direction[0] * 2
```

```
                    (new_y = self.position[1] + int(direction[1] * 2

                :if 0 <= new_x < board_size and 0 <= new_y < board_size
 if not any(piece.position == (new_x, new_y) for piece in pieces if
                                                        \ piece != self) and
not any(obstacle.position == (new_x, new_y) for obstacle in
                                                                :(obstacles
                                    (self.position = (new_x, new_y

                                                            :class Obstacle
                                                :(def __init__(self, position
                                                 self.position = position
                                                       self.color = BLACK

                                                    :(def draw(self, screen
                                                                    """
                                            .Draws the obstacle on the screen
                                                                    """
                                        ,pygame.draw.rect(screen, self.color
    self.position[0] * BLOCK_SIZE, self.position[1] *)
                                                ((BLOCK_SIZE, BLOCK_SIZE, BLOCK_SIZE

                                                                :class Train
                                                        :(def __init__(self
                                    ((self.position = (7, random.randint(1, 6
                                                        self.direction = -1
                                            ("self.color = pygame.Color("brown
                                                        self.moving = False
                                                self.front_icon = TRAIN_ICON
self.back_icon = pygame.transform.rotate(TRAIN_ICON, 180)  # Rotate the
                                                    icon for the back of the train

                                                    :(def draw(self, screen
                                                                    """
                                                .Draws the train on the screen
                                                                    """
                                                    x, y = self.position
  pygame.draw.rect(screen, self.color, (x * BLOCK_SIZE, y * BLOCK_SIZE,
                                                    ((BLOCK_SIZE * 2, BLOCK_SIZE

                                    if self.direction == -1:  # Moving left
     ((screen.blit(self.front_icon, (x * BLOCK_SIZE, y * BLOCK_SIZE
((screen.blit(self.back_icon, ((x + 1) * BLOCK_SIZE, y * BLOCK_SIZE
                                                    else:  # Moving right
      ((screen.blit(self.back_icon, (x * BLOCK_SIZE, y * BLOCK_SIZE
            screen.blit(self.front_icon, ((x + 1) * BLOCK_SIZE, y *
                                                                ((BLOCK_SIZE

                                        :(def move(self, pieces, board_size=8
```

```
                                                                """
                    .Moves the train and pushes any pieces in its way
                                                                """
                                                    :if self.moving
                                                x, y = self.position
                                            new_x = x + self.direction

 if new_x < 0 or new_x >= board_size - 1:  # -1 because train is 2
                                                                    blocks wide
                     self.direction *= -1  # Reverse direction
  new_x = x + self.direction  # Recalculate new_x with updated
                                                                      direction

                                            Check for pieces to push #
                                                  :for piece in pieces
   if piece.position[1] == y and (new_x <= piece.position[0] <
                                                                :(new_x + 2
                                              Push the piece #
 push_x = new_x - 1 if self.direction == -1 else new_x + 2
                                :if 0 <= push_x < board_size
                           (piece.position = (push_x, y
                                                        :else
                 If can't push, don't move the train #
                                              return

                                            (self.position = (new_x, y
                  self.moving = False  # Train moves once per turn

                                        :(def is_occupied(self, position
                                                                """
               .Checks if the given position is occupied by the train
                                                                """
                                                    x, y = position
                                                tx, ty = self.position
                                    return tx <= x < tx + 2 and ty == y

                                                        :(def start_turn(self
                                                                """
                                .Allows the train to move on the next turn
                                                                """
                 self.moving = True  # Allow the train to move next turn

                                        :(def display_difficulty_menu(screen
                                                                      """
.Displays the difficulty selection menu and returns the selected difficulty
                                                                      """
                                                     (screen.fill(BLACK
                                            (font = pygame.font.Font(None, 74
```

```
                            (easy_text = font.render("Easy", True, GREEN
        (("medium_text = font.render("Medium", True, pygame.Color("yellow
                              (hard_text = font.render("Hard", True, RED

    ((easy_rect = easy_text.get_rect(center=(width // 2, height // 2 - 100
      ((medium_rect = medium_text.get_rect(center=(width // 2, height // 2
    ((hard_rect = hard_text.get_rect(center=(width // 2, height // 2 + 100

                                        (screen.blit(easy_text, easy_rect
                                    (screen.blit(medium_text, medium_rect
                                        (screen.blit(hard_text, hard_rect

                                                    ()pygame.display.flip

                                                                :while True
                                             :()for event in pygame.event.get
                                            :if event.type == pygame.QUIT
                                                        ()pygame.quit
                                                           ()sys.exit
                                  :if event.type == pygame.MOUSEBUTTONDOWN
                                    ()mouse_pos = pygame.mouse.get_pos
                                 :(if easy_rect.collidepoint(mouse_pos
                                                "return "easy
                               :(elif medium_rect.collidepoint(mouse_pos
                                             "return "medium
                                 :(elif hard_rect.collidepoint(mouse_pos
                                                "return "hard

                                                    Initialize game entities #
                                    (difficulty = display_difficulty_menu(screen
                                                    (reset_game(difficulty

                                                                Main game loop #
                                                                  running = True

                                                                  :while running
                                            screen.fill(BLACK)  # Clear screen
                                    (button_rects = draw_abilities_window(screen

                                             :()for event in pygame.event.get
                                            :if event.type == pygame.QUIT
                                                        ()pygame.quit
                                                           ()sys.exit
                                        :elif event.type == pygame.KEYDOWN
                                                        :if player_turn
                                                    } = key_mapping
pygame.K_w: 'UP', pygame.K_s: 'DOWN', pygame.K_a: 'LEFT',
                                                           ,'pygame.K_d: 'RIGHT
```

```
pygame.K_q: 'UP_LEFT', pygame.K_e: 'UP_RIGHT', pygame.K_z:
                                        ''DOWN_LEFT', pygame.K_c: 'DOWN_RIGHT
                                                                {
                        (direction = key_mapping.get(event.key
                                                    :if direction
  (moved = player.move(direction, pieces, obstacles, train
                                                    :if moved
                                     player_turn = False
                    :if event.key == pygame.K_r and not freeze_used
                           player_x, player_y = player.position
                                          :for piece in pieces
                                      :'if piece.type != 'flag
                        piece_x, piece_y = piece.position
    if abs(piece_x - player_x) <= 2 and abs(piece_y -
                                                            :player_y) <= 2
                                  piece.frozen = True
                                piece.frozen_turns = 0
                                             freeze_used = True
                     :if event.key == pygame.K_t and not push_used
                          player_x, player_y = player.position
                                          :for piece in pieces
                                      :'if piece.type != 'flag
        piece.push_away((player_x, player_y), pieces,
                                                                (obstacles
                                               push_used = True
              :if event.key == pygame.K_y and not double_move_used
                                 player.double_move_active = True
                                          double_move_used = True
                               :elif event.type == pygame.MOUSEBUTTONDOWN
                                        mouse_x, mouse_y = event.pos
                     :(for i, button_rect in enumerate(button_rects
                :(if button_rect.collidepoint(mouse_x, mouse_y
                                [ability = available_abilities[i
                           :[if not ability_used[ability.name
                            :'if ability.name == 'Freeze
              player_x, player_y = player.position
                                    :for piece in pieces
                         :'if piece.type != 'flag
        piece_x, piece_y = piece.position
      if abs(piece_x - player_x) <= 2 and
                                                :abs(piece_y - player_y) <= 2
                    piece.frozen = True
                piece.frozen_turns = 0
                               :'elif ability.name == 'Push
              player_x, player_y = player.position
                                    :for piece in pieces
                         :'if piece.type != 'flag
    piece.push_away((player_x, player_y),
                                                        (pieces, obstacles
```

```
                    :'elif ability.name == 'Double Move
                  player.double_move_active = True
                :'elif ability.name == 'Obstacle Shift
  (spawn_or_respawn_obstacles(player, pieces, train

                    ability_used[ability.name] = True

                                  :if not player_turn and not game_over
                                              :for piece in pieces
                                        :'if piece.type != 'flag
                                        :if not piece.frozen
      new_position = piece.calculate_move(player.position,
                                                  (pieces, obstacles, train
                                        :if new_position
                          piece.position = new_position
                                  ()piece.update_frozen_status
                                                  player_turn = True
                                                          :if train
                                             ()train.start_turn

                                            :if not game_over and train
                                                 (train.move(pieces

                                    Check if player reached the bottom row #
                                            :if player.position[1] == 7
    ((display_message(screen, "U WIN", GREEN, (width // 2, height // 2
                          pygame.time.delay(2000)  # Pause for 2 seconds
  difficulty = display_difficulty_menu(screen)  # Show difficulty menu
                                                                      again
          reset_game(difficulty)  # Reset the game with new difficulty

                                                    :for piece in pieces
        :if piece.type != 'flag' and piece.position == player.position
((display_message(screen, "YOU DIED", RED, (width // 2, height // 2
                     pygame.time.delay(2000)  # Pause for 2 seconds
   difficulty = display_difficulty_menu(screen)  # Show difficulty
                                                              menu again
      reset_game(difficulty)  # Reset the game with new difficulty
                                                           break

                                                Draw the chess board #
                                                    (draw_board(screen

                                                  Draw the obstacles #
                                             :for obstacle in obstacles
                                              (obstacle.draw(screen

                                        Draw the player, pieces, and train #
                                                    (player.draw(screen
```

```
                                                    :for piece in pieces
                                                  (piece.draw(screen
                                                              :if train
                                                  (train.draw(screen

                                                   ()pygame.display.flip
```