1) Main Game

Actions			
Up: 0	Down: 1	Left: 2	Right: 3

Tile Type in Grid						
Queen: Q	Pig: P	Rock: R	Tile: T	Hen: H	Slingshot: S	Egg: E

Attriute	Information	Type
grid	The grid is a 2D list that represents the game environment. It's essentially a **10x10** grid where each cell can contain different entities or obstacles, such as a tile ('T'), hen ('H'), queen ('Q'), pig ('P'), egg ('E'), rock ('R'), or slingshot ('S'). `[['T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T'], ['T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T'], ['T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T'], ['T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T'], ['T', 'T', 'T', 'T', 'Q', 'T', 'T', 'T', 'T', 'T'], ['T', 'T', 'T', 'T', 'T', 'H', 'T', 'E', 'P', 'T'], ['T', 'T', 'R', 'T', 'T', 'T', 'T', 'T', 'P', 'T'], ['T', 'P', 'T', 'T', 'T', 'T', 'R', 'T', 'T', 'T'], ['T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'S'], ['T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T']]`	list
num_actions	This attribute keeps track of the number of actions the hen has performed in the game.	int

Function	Information	Return object
reset()	This function resets the environment, starting a new episode with the initial state of the environment.	-

hen_step(action)	This function defines how the hen moves in the grid environment based on the given action. It updates the grid and the hen's position accordingly if the movement is valid.	-
queen_step()	This function determines and executes the next movement for the queen within the grid. It updates the grid attribute and the queen's position accordingly if the movement is valid.	-
render(screen)	This function displays the game environment and can be called within the game loop.	-
generate_hen_successors(grid)	This function generates all possible valid successor states for the hen in the current grid environment. It evaluates potential moves based on the rules of the environment and returns a list of successor grids along with the actions required to reach them. The function returns: A **list of tuples**, where each tuple consists of: 1. **successor_grid**: A 2D list representing the new state of the environment after the hen moves. 2. **action**: An integer representing the direction of the new movement. **Output:** `[ (successor_grid_1, 0), (successor_grid_2, 1), (successor_grid_4, 3) ]`	list

generate_queen_successors(grid)	This generates all possible valid successor states for the queen's movement within the grid environment. It evaluates potential moves the queen can make, validates them according to the environment's rules, and returns a list of successor states paired with the corresponding actions. **Output:** `[ (successor_grid_1, 0), (successor_grid_2, 2), (successor_grid_4, 3) ]`	list
get_egg_coordinate(grid)	This function identifies all the positions (coordinates) of eggs ('E') on the grid. The function returns: • A **list of tuples**, where each tuple represents the row and column indices (r, c) of an egg's position on the grid. If no eggs are found, it returns an **empty list**. **Output:** `[(0, 1), (1, 0), (1, 2)]`	list
get_pig_coordinate(grid)	This method identifies the positions (coordinates) of all pigs ('P') in the grid environment. The function returns: • A **list of tuples**, where each tuple contains the row and column indices (r, c) of a pig's position on the grid. If no pigs are found, the function returns an **empty list**. **Output:** `[(0, 0), (7, 7), (8, 8)]`	list

get_hen_position(grid)	This method identifies the position (coordinates) of the hen ('H') in the grid environment. The function returns: • A **tuple (r, c)**, representing the row and column indices of the hen's position in the grid. If the hen ('H') is not found, the function implicitly returns None. **Output:** `(1, 1)`	tuple
get_queen_position(grid)	This method identifies the position (coordinates) of the queen ('Q') in the grid environment. The function returns: • A **tuple (r, c)**, representing the row and column indices of the queen's position in the grid. If the queen ('Q') is not found, the function implicitly returns None. **Output:** `(1, 1)`	tuple
get_slingshot_position(grid)	This method identifies the position (coordinates) of the slingshot ('S') in the grid environment. The function returns: • A **tuple (r, c)**, representing the row and column indices of the slingshot's position in the grid. If the slingshot ('S') is not found in the grid, the function implicitly returns None. **Output:** `(1, 1)`	tuple

is_win(grid)	This method determines if the Max player (or agent) has won the game or not. Output The function returns: • True if the player has won the game based on the specified win conditions. • False if the player has not yet won.	Bool
is_lose(grid, num_actions)	This method checks if the Max player has lost the game. It evaluates two conditions: whether the player has lost based on specific game rules or if the maximum number of allowed actions has been exceeded. Output The function returns: • True if the player has lost the game (either based on specific game conditions or by exceeding the maximum allowed actions). • False if the player has not lost.	Bool
calculate_score(grid, num_actions)	This method calculates the max player's score based on various factors, including the number of eggs collected, pigs defeated, the number of actions taken, and whether the player has won or lost the game. Output The function returns: • score: An integer representing the total score calculated based on the player's progress in the game.	int

is_queen_exists(grid), is_hen_exists(grid)	These two functions, **is_queen_exists** and **is_hen_exists**, are used to check the presence of the queen ('Q') and hen ('H') in the grid, respectively.	Bool
print_grid(grid)	This function generates a string representation of the game grid and returns it. Output: • **printed_grid**: A string that represents the grid, formatted with each row of the grid printed on a new line.	str

2) Optional Game

Actions			
Up: 0	Down: 1	Left: 2	Right: 3

Tile Type in Grid							
Queen: Q	Pig: P	Rock: R	Tile: T	Hen: H	Bird: B	Slingshot: S	Egg: E

Attriute	Information	Type
grid	The grid is a 2D list that represents the game environment. It's essentially a **10x10** grid where each cell can contain different entities or obstacles, such as a tile ('T'), hen ('H'), queen ('Q'), pig ('P'), egg ('E'), rock ('R'), bird ('B') or slingshot ('S'). `[['T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T'], ['T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T'], ['T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T'], ['T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T'], ['T', 'T', 'T', 'T', 'Q', 'T', 'T', 'T', 'T', 'T'], ['T', 'T', 'T', 'T', 'T', 'H', 'T', 'E', 'P', 'T'], ['T', 'T', 'R', 'B', 'T', 'T', 'T', 'T', 'P', 'T'], ['T', 'P', 'T', 'T', 'T', 'T', 'R', 'T', 'T', 'T'], ['T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'S'], ['T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T', 'T']]`	list
num_actions	This attribute keeps track of the number of actions the hen and bird have performed in the game.	int

Function	Information	Return object
reset()	This function resets the environment, starting a new episode with the initial state of the environment.	-
hen_step(action)	This function defines how the hen moves in the grid environment based on the given action. It updates the grid and the hen's position accordingly if the movement is valid.	-

queen_step()	This function determines and executes the next movement for the queen within the grid. It updates the grid and the queen's position accordingly if the movement is valid.	-
bird_step(action)	This function determines and executes the next movement for the bird within the grid. It updates the grid and the bird's position accordingly if the movement is valid	
render(screen)	This function displays the game environment and can be called within the game loop.	-
generate_hen_successors(grid)	This function generates all possible valid successor states for the hen in the current grid environment. It evaluates potential moves based on the rules of the environment and returns a list of successor grids along with the actions required to reach them. The function returns: A **list of tuples**, where each tuple consists of: 3. **successor_grid**: A 2D list representing the new state of the environment after the hen moves. 4. **action**: An integer representing the direction of the new movement. **Output:** `[ (successor_grid_1, 0), (successor_grid_2, 1), (successor_grid_4, 3) ]`	list

generate_queen_successors(grid)	This generates all possible valid successor states for the queen's movement within the grid environment. It evaluates potential moves the queen can make, validates them according to the environment's rules, and returns a list of successor states paired with the corresponding actions.
	
	Output:
	`[`
	`  (successor_grid_1, 0),`
	`  (successor_grid_2, 2),`
	`  (successor_grid_4, 3)`
	`]`
		list

generate_bird_successors(grid)	This generates all possible valid successor states for the bird's movement within the grid environment. It evaluates potential moves the bird can make, validates them according to the environment's rules, and returns a list of successor states paired with the corresponding actions.
	
	Output:
	`[`
	`  (successor_grid_1, 1),`
	`  (successor_grid_2, 2),`
	`  (successor_grid_4, 3)`
	`]`	

get_egg_coordinate(grid)	This function identifies all the positions (coordinates) of eggs ('E') on the grid. The function returns: • A **list of tuples**, where each tuple represents the row and column indices (r, c) of an egg's position on the grid. If no eggs are found, it returns an **empty list**. **Output:** `[(0, 1), (1, 0), (1, 2)]`	list

get_pig_coordinate(grid)	This method identifies the positions (coordinates) of all pigs ('P') in the grid environment. The function returns: • A **list of tuples**, where each tuple contains the row and column indices (r, c) of a pig's position on the grid. If no pigs are found, the function returns an **empty list**. **Output:** `[(0, 0), (7, 7), (8, 8)]`	list
get_hen_position(grid)	This method identifies the position (coordinates) of the hen ('H') in the grid environment. The function returns: • A **tuple (r, c)**, representing the row and column indices of the hen's position in the grid. If the hen ('H') is not found, the function implicitly returns None. **Output:** `(1, 1)`	tuple
get_queen_position(grid)	This method identifies the position (coordinates) of the queen ('Q') in the grid environment. The function returns: • A **tuple (r, c)**, representing the row and column indices of the queen's position in the grid. If the queen ('Q') is not found, the function implicitly returns None. **Output:** `(1, 1)`	tuple

get_slingshot_position(grid)	This method identifies the position (coordinates) of the slingshot ('S') in the grid environment. The function returns: • A **tuple (r, c)**, representing the row and column indices of the slingshot's position in the grid. If the slingshot ('S') is not found in the grid, the function implicitly returns None. **Output:** `(1, 1)`	tuple
get_bird_position(grid)	This method identifies the position (coordinates) of the Bird ('B') in the grid environment. The function returns: • A **tuple (r, c)**, representing the row and column indices of the bird's position in the grid. If the bird is not found in the grid, the function implicitly returns None. **Output:** `(1, 1)`	tuple
is_win(grid)	This method determines if the Max player (or agent) has won the game or not. **Output** The function returns: • **True** if the player has won the game based on the specified win conditions. • **False** if the player has not yet won.	Bool
is_lose(grid, num_actions)	This method checks if the Max player has lost the game. It evaluates two conditions: whether the player has lost based on specific game rules or if the	Bool

	maximum number of allowed actions has been exceeded. Output The function returns: • True if the player has lost the game (either based on specific game conditions or by exceeding the maximum allowed actions). • False if the player has not lost.	
calculate_score(grid, num_actions)	This method calculates the max player's score based on various factors, including the number of eggs collected, pigs defeated, the number of actions taken, and whether the player has won or lost the game. Output The function returns: • score: An integer representing the total score calculated based on the player's progress in the game.	int
is_queen_exists(grid), is_bird_exists(grid), is_hen_exists(grid)	These two functions, is_queen_exists, is_bird_exists, and is_hen_exists, are used to check the presence of the queen ('Q'), Bird ('B'), and hen ('H') in the grid, respectively.	Bool
print_grid(grid)	This function generates a string representation of the game grid and returns it. Output: • printed_grid: A string that represents the grid, formatted with each row of the grid printed on a new line.	str