Here's the line-by-line explanation of the code:

---

Imports

import tkinter as tk:

Imports the tkinter library, used for creating GUI applications.
It is aliased as tk for easier reference.

from tkinter import messagebox:

Imports messagebox, a module in tkinter used to display pop-up messages to users.

import random:

Imports the random module, which generates random numbers and selections.

---

WumpusWorld Class

class WumpusWorld::

Defines the game world with pits, gold, and the Wumpus.

def __init__(self, size=4)::

Initializes a 4x4 grid (default size) and places items like gold, Wumpus, and pits.

self.size = size:

Stores the grid size.

self.grid = [['' for _ in range(size)] for _ in range(size)]:

Creates a 2D list (grid) filled with empty strings.

self.agent_position = (0, 0):

Places the agent at the top-left corner initially.

self.gold_position = self._place_item('G'):

Randomly places the gold ('G') in the grid.

self.wumpus_position = self._place_item('W'):

Randomly places the Wumpus ('W') in the grid.

self.pit_positions = self._place_pits():

Places pits ('P') randomly, ensuring the starting position is safe.

self.arrow = True:

Initializes a single arrow for the agent.

---

def _place_item(self, item)::

Helper function to place a single item (like gold or Wumpus) randomly.

while True::

Repeats until a valid location is found.

x, y = random.randint(0, self.size - 1), random.randint(0, self.size - 1):

Randomly selects a grid position.

if self.grid[x][y] == '' and (x, y) != (0, 0)::

Ensures the position is empty and not the agent’s starting point.

self.grid[x][y] = item:

Places the item in the grid.

return (x, y):

Returns the item's position.

---

def _place_pits(self)::

Helper function to place multiple pits.

pits = []:

Initializes an empty list to store pit positions.

for _ in range(self.size)::

Adds a number of pits equal to the grid size.

if self.grid[x][y] == '' and (x, y) != (0, 0)::

Ensures the pit doesn’t overwrite another item or occupy the starting position.

pits.append((x, y)):

Records the pit's position.

---

Agent Class

class Agent::

Represents the player's agent navigating the grid.

def __init__(self, world)::

Initializes the agent with a reference to the world.

self.world = world:

Links the agent to the game world.

self.position = world.agent_position:

Sets the initial position to (0, 0).

self.has_gold = False:

Indicates whether the agent has found the gold.

self.alive = True:

Keeps track of whether the agent is alive.

---

def perceive(self)::
Checks the agent’s current surroundings and returns a list of perceptions (e.g., 'breeze', 'stench').

---

Agent Actions

def move(self, direction)::

Moves the agent in a specified direction (up, down, left, right) and checks for encounters with hazards (pits or Wumpus).

def grab_gold(self)::

Picks up gold if the agent is on the same tile as the gold.

def shoot_arrow(self, direction)::

Attempts to shoot an arrow in the specified direction and checks whether it hits the Wumpus.

---

WumpusWorldGame Class

class WumpusWorldGame(tk.Tk)::

Manages the GUI for the game using tkinter.

self.create_widgets():

Sets up buttons for movement and actions.

self.update_grid():

Updates the visual representation of the grid based on the game state.

def move_agent(self, direction)::

Moves the agent, updates perceptions, and checks for victory/defeat.

if __name__ == "__main__"::

Starts the game by creating and running the GUI application.

```
import tkinter as tk
from tkinter import messagebox
import random

class WumpusWorld:
    def __init__(self, size=4):
        self.size = size
        self.grid = [['' for _ in range(size)] for _ in range(size)]
        self.agent_position = (0, 0)
        self.gold_position = self._place_item('G')
        self.wumpus_position = self._place_item('W')
        self.pit_positions = self._place_pits()
        self.arrow = True
```

```
    def _place_item(self, item):
        while True:
            x, y = random.randint(0, self.size - 1), random.randint(0,
self.size - 1)
            if self.grid[x][y] == '' and (x, y) != (0, 0):
                self.grid[x][y] = item
                return (x, y)
```

```
    def _place_pits(self):
        pits = []
        for _ in range(self.size):
            x, y = random.randint(0, self.size - 1), random.randint(0,
self.size - 1)
            if self.grid[x][y] == '' and (x, y) != (0, 0):
                self.grid[x][y] = 'P'
                pits.append((x, y))
        return pits
```

```
class Agent:
    def __init__(self, world):
        self.world = world
        self.position = world.agent_position
        self.has_gold = False
        self.alive = True
```

```
    def perceive(self):
        x, y = self.position
        perceptions = []
        if (x, y) == self.world.gold_position:
            perceptions.append('glitter')
        if self._is_adjacent_to(x, y, self.world.wumpus_position):
            perceptions.append('stench')
        if any(self._is_adjacent_to(x, y, pit) for pit in
self.world.pit_positions):
            perceptions.append('breeze')
        return perceptions
```

```
    def _is_adjacent_to(self, x, y, pos):
        return abs(x - pos[0]) + abs(y - pos[1]) == 1
```

```
    def move(self, direction):
        x, y = self.position
        if direction == 'up' and x > 0:
            self.position = (x - 1, y)
        elif direction == 'down' and x < self.world.size - 1:
            self.position = (x + 1, y)
        elif direction == 'left' and y > 0:
            self.position = (x, y - 1)
        elif direction == 'right' and y < self.world.size - 1:
            self.position = (x, y + 1)
        else:
            print("Invalid move")
        print(f"Moved {direction} to {self.position}")
```

```
        if self.position == self.world.wumpus_position:
            self.alive = False
            print("You've been eaten by the Wumpus!")
```

```
        elif self.position in self.world.pit_positions:
            self.alive = False
            print("You fell into a pit!")
```

```
    def grab_gold(self):
        if self.position == self.world.gold_position:
            self.has_gold = True
            print("Gold acquired!")
        else:
            print("No gold here.")
```

```
    def shoot_arrow(self, direction):
        if not self.world.arrow:
            print("No arrows left.")
            messagebox.showinfo("Message", "No arrows left.")
            return
        self.world.arrow = False
        x, y = self.position
        if direction == 'up':
            target = (x - 1, y)
        elif direction == 'down':
            target = (x + 1, y)
        elif direction == 'left':
            target = (x, y - 1)
        elif direction == 'right':
            target = (x, y + 1)
        else:
            print("Invalid direction")
            return
```

```
        if target == self.world.wumpus_position:
            print("You shot the Wumpus!")
            messagebox.showinfo("Message", "You shot the Wumpus!")
            self.world.wumpus_position = (-1, -1)
        else:
            print("Missed!")
            messagebox.showinfo("Message", "Missed!")
```

```
class WumpusWorldGame(tk.Tk):
    def __init__(self):
        super().__init__()
        self.title("Wumpus World Game")
        self.configure(bg="lightgray")
        self.world = WumpusWorld()
        self.agent = Agent(self.world)
        self.create_widgets()
        self.update_grid()
```

```
    def create_widgets(self):
```

```
        self.grid_frame = tk.Frame(self, bg="lightgray")
        self.grid_frame.pack(padx=10, pady=10)
```

```
        self.cells = [[tk.Label(self.grid_frame, text='', width=8,
height=4, borderwidth=2, relief="solid", bg="white", font=("Arial", 12))
                       for _ in range(self.world.size)] for _ in
range(self.world.size)]
        for i in range(self.world.size):
            for j in range(self.world.size):
                self.cells[i][j].grid(row=i, column=j)
```

```
        self.control_frame = tk.Frame(self, bg="lightgray")
        self.control_frame.pack(padx=10, pady=10)
```

```
        self.move_up_button = tk.Button(self.control_frame, text="Move
Up", command=lambda: self.move_agent('up'), bg="lightblue")
        self.move_up_button.grid(row=0, column=1)
```

```
        self.move_left_button = tk.Button(self.control_frame,
text="Move Left", command=lambda: self.move_agent('left'),
bg="lightblue")
        self.move_left_button.grid(row=1, column=0)
```

```
        self.move_down_button = tk.Button(self.control_frame,
text="Move Down", command=lambda: self.move_agent('down'),
bg="lightblue")
        self.move_down_button.grid(row=1, column=1)
```

```
        self.move_right_button = tk.Button(self.control_frame,
text="Move Right", command=lambda: self.move_agent('right'),
bg="lightblue")
        self.move_right_button.grid(row=1, column=2)
```

```
        self.grab_button = tk.Button(self.control_frame, text="Grab
Gold", command=self.grab_gold, bg="yellow")
        self.grab_button.grid(row=2, column=0, columnspan=3, pady=5)
```

```
        self.shoot_frame = tk.Frame(self.control_frame, bg="lightgray")
        self.shoot_frame.grid(row=3, column=0, columnspan=3, pady=5)
```

```
        self.shoot_up_button = tk.Button(self.shoot_frame, text="Shoot
Up", command=lambda: self.shoot_arrow('up'), bg="lightcoral")
        self.shoot_up_button.grid(row=0, column=1)
```

```
        self.shoot_left_button = tk.Button(self.shoot_frame,
text="Shoot Left", command=lambda: self.shoot_arrow('left'),
bg="lightcoral")
        self.shoot_left_button.grid(row=1, column=0)
```

```
        self.shoot_down_button = tk.Button(self.shoot_frame,
text="Shoot Down", command=lambda: self.shoot_arrow('down'),
bg="lightcoral")
        self.shoot_down_button.grid(row=1, column=1)
```

```
        self.shoot_right_button = tk.Button(self.shoot_frame,
text="Shoot Right", command=lambda: self.shoot_arrow('right'),
bg="lightcoral")
        self.shoot_right_button.grid(row=1, column=2)
```

```
        self.perception_label = tk.Label(self, text="", bg="lightgray",
font=("Arial", 14))
        self.perception_label.pack(pady=5)
```

```
    def update_grid(self):
        for i in range(self.world.size):
            for j in range(self.world.size):
                self.cells[i][j].config(text='', bg='white')
```

```
        x, y = self.agent.position
        self.cells[x][y].config(text='A', bg='lightgreen')
```

```
        for (i, j) in self.world.pit_positions:
            self.cells[i][j].config(bg='gray')
```

```
        wumpus_x, wumpus_y = self.world.wumpus_position
        if wumpus_x >= 0 and wumpus_y >= 0:
            self.cells[wumpus_x][wumpus_y].config(bg='red')
```

```
        gold_x, gold_y = self.world.gold_position
        self.cells[gold_x][gold_y].config(bg='gold')
```

```
        perceptions = self.agent.perceive()
        self.perception_label.config(text=f"Perceptions: {',
'.join(perceptions)}")
```

```
    def move_agent(self, direction):
        self.agent.move(direction)
        self.check_status()
        self.update_grid()
```

```
    def grab_gold(self):
        self.agent.grab_gold()
        self.check_status()
```

```
    def shoot_arrow(self, direction):
        self.agent.shoot_arrow(direction)
        self.check_status()
```

```
    def check_status(self):
        if not self.agent.alive:
```

```
            messagebox.showinfo("Game Over", "You've been killed! Game
Over.")
            self.destroy()
        elif self.agent.has_gold:
            messagebox.showinfo("Congratulations", "You've found the
gold! You win!")
            self.destroy()
```

```
if __name__ == "__main__":
    game = WumpusWorldGame()
    game.mainloop()
```

Overview of the Game: Wumpus World
The Wumpus World is a grid-based text-and-GUI-based adventure game where the player, acting as an agent, navigates a dungeon to retrieve gold while avoiding hazards (like pits and the Wumpus, a dangerous monster). It's a strategy game with randomized obstacles that demand careful decision-making.

---

Objective
Primary Goal: Find the gold in the dungeon and safely escape.
Secondary Goal: Survive hazards, including:
The Wumpus: A monster that eats you if you step into its tile.
Pits: Traps that instantly end the game if you fall into them.

---

Key Gameplay Elements
The Grid
Default size is 4x4.
Tiles contain various items:
Gold ('G'): The treasure the agent must grab.
Wumpus ('W'): A deadly hazard that kills the agent on contact.
Pits ('P'): Randomly placed tiles representing death traps.
The Agent
Starts at the top-left corner (0, 0).
Performs actions to navigate, perceive, grab the gold, or fight the Wumpus using an arrow.
The agent has one arrow to attempt to shoot the Wumpus.

---

Perceptions and Feedback
The agent perceives environmental clues about nearby hazards:
Glitter: Indicates the gold is on the current tile.
Stench: Warns of the Wumpus in an adjacent tile (not diagonal).
Breeze: Warns of pits in adjacent tiles.
Example:
If standing on tile (1,1):
If (1,2) has a pit: The agent perceives "breeze."
If (2,1) has a Wumpus: The agent perceives "stench."

---

Actions Available to the Agent

Move:

Navigate up, down, left, or right to explore the dungeon.
Invalid moves (e.g., going out of bounds) are ignored.
Each move may expose the agent to hazards.

Grab Gold:

If the agent is on the same tile as the gold, they can grab it.

Shoot Arrow:

Fires an arrow in a chosen direction (up, down, left, or right).
If the arrow hits the Wumpus, the monster is defeated.
If the arrow misses or has already been used, no further attacks are possible.

---

Victory and Defeat
Win:
The agent grabs the gold and exits the dungeon alive.
Lose:
The agent encounters the Wumpus (gets eaten).
The agent falls into a pit.
The player closes the game without grabbing the gold.

---

Game Design Features
Randomized Elements
The positions of the gold, Wumpus, and pits are randomized in each game.
Ensures unique gameplay for every session.
Visual Feedback with tkinter GUI
The GUI displays the grid and items:
Green Tile: Agent's position.
Gold Tile: Yellow tile.
Pit Tile: Gray tile.
Wumpus Tile: Red tile.
Perceptions are updated in the GUI in real time.

---

Flow of Gameplay

Game Initialization:

A 4x4 grid is created with random placement of hazards and the gold.
The agent starts at the top-left corner (0,0).

Exploration:

The player controls the agent to move around the grid using buttons.

Each move updates perceptions (e.g., breeze, stench, glitter).

Decision-Making:

Use feedback (perceptions) to:
Safely navigate the dungeon.
Avoid stepping into tiles containing hazards.
Plan when to use the arrow.

Outcome:

Either successfully find the gold and win the game or encounter a hazard and lose.

---

Strategy Tips
Use perceptions (breeze and stench) to avoid stepping into dangerous tiles.
Plan the use of the arrow wisely—you only get one shot!
Always remember the location of known hazards and the safe path you've explored.