```python
import random

# Constants for the grid size
GRID_SIZE = 4

# Representations for different elements in the world
EMPTY = 0
PIT = 1
WUMPUS = 2
GOLD = 3
AGENT = 4

# Directions
UP, RIGHT, DOWN, LEFT = 0, 1, 2, 3

class WumpusWorld:
    def __init__(self):
        self.grid = [[EMPTY] * GRID_SIZE for _ in range(GRID_SIZE)]
        self.agent_position = [0, 0]
        self.agent_direction = RIGHT
        self.has_arrow = True
        self.has_gold = False

        # Place pits with a probability of 0.2
        for i in range(GRID_SIZE):
            for j in range(GRID_SIZE):
                if (i, j) != (0, 0) and random.random() < 0.2:
                    self.grid[i][j] = PIT

        # Place the Wumpus and gold
        self.grid[random.randint(1, GRID_SIZE-1)][random.randint(1, GRID_SIZE-
1)] = WUMPUS
        self.grid[random.randint(0, GRID_SIZE-1)][random.randint(0, GRID_SIZE-
1)] = GOLD

    def get_percepts(self):
        x, y = self.agent_position
        percepts = []

        # Check for stench (Wumpus nearby)
        if any(self.is_adjacent(x, y, WUMPUS)):
            percepts.append("Stench")

        # Check for breeze (Pit nearby)
        if any(self.is_adjacent(x, y, PIT)):
            percepts.append("Breeze")

        # Check for glitter (Gold in the same room)
        if self.grid[x][y] == GOLD:
            percepts.append("Glitter")

        return percepts

    def is_adjacent(self, x, y, element):
        adjacent = []
        if x > 0:
            adjacent.append(self.grid[x-1][y] == element)
        if x < GRID_SIZE-1:
            adjacent.append(self.grid[x+1][y] == element)
        if y > 0:
            adjacent.append(self.grid[x][y-1] == element)
        if y < GRID_SIZE-1:
            adjacent.append(self.grid[x][y+1] == element)
        return adjacent

    def move_forward(self):
        x, y = self.agent_position
        if self.agent_direction == UP and x > 0:
            self.agent_position[0] -= 1
        elif self.agent_direction == DOWN and x < GRID_SIZE-1:
            self.agent_position[0] += 1
        elif self.agent_direction == LEFT and y > 0:
            self.agent_position[1] -= 1
        elif self.agent_direction == RIGHT and y < GRID_SIZE-1:
            self.agent_position[1] += 1

    def turn_left(self):
        self.agent_direction = (self.agent_direction - 1) % 4

    def turn_right(self):
        self.agent_direction = (self.agent_direction + 1) % 4

    def grab_gold(self):
        x, y = self.agent_position
        if self.grid[x][y] == GOLD:
            self.has_gold = True
            self.grid[x][y] = EMPTY

    def shoot_arrow(self):
        if self.has_arrow:
            self.has_arrow = False
            # Simplified: Assume the Wumpus is in a direct line of sight and
is hit
            return "Scream"
        return None
```

```python
# Simulation
def simulate():
    world = WumpusWorld()
    steps = 0
    actions = ["Move Forward", "Turn Left", "Turn Right", "Grab Gold", "Shoot Arrow"]
    action_funcs = [world.move_forward, world.turn_left, world.turn_right, world.grab_gold, world.shoot_arrow]

    while True:
        percepts = world.get_percepts()
        print(f"Step {steps}: Agent at {world.agent_position}, Facing {world.agent_direction}")
        print("Percepts:", percepts)

        if "Glitter" in percepts:
            world.grab_gold()
            print("Action: Grab Gold")
            break

        # Simplified decision-making process
        if "Stench" in percepts and world.has_arrow:
            print("Action: Shoot Arrow")
            world.shoot_arrow()
        else:
            action = random.choice(action_funcs)
            action()
            print("Action:", actions[action_funcs.index(action)])

        steps += 1
        if steps > 100:  # Prevent infinite loop in case of unexpected issues
            break

simulate()
```

Observational Analysis

Constants and Representations

- **GRID_SIZE**: 4
- **Elements**: `EMPTY`, `PIT`, `WUMPUS`, `GOLD`, `AGENT`
- **Directions**: `UP`, `RIGHT`, `DOWN`, `LEFT`

Class: `WumpusWorld`

Initialization

- Creates a 4x4 grid.
- Agent starts at `(0, 0)` facing `RIGHT`.
- Randomly places pits with a 0.2 probability, and randomly places the Wumpus and gold.

Methods

- **`get_percepts`**: Returns percepts based on the agent's position ("Stench" for nearby Wumpus, "Breeze" for nearby pit, "Glitter" for gold in the same cell).
- **`move_forward`**: Moves the agent forward if within grid bounds.
- **`turn_left`** and **`turn_right`**: Rotate the agent.
- **`grab_gold`**: Picks up gold if in the current cell.
- **`shoot_arrow`**: Shoots an arrow if available.

Simulation (`simulate` function)

Initialization

- Creates a `WumpusWorld` instance.
- Defines actions and corresponding functions.

Main Loop

- Runs until the agent grabs the gold or exceeds 100 steps.
- Agent perceives surroundings, makes decisions, and acts:
    - Grabs gold if "Glitter" is perceived.
    - Shoots arrow if "Stench" is perceived and arrow is available.
    - Otherwise, performs a random action.

Methods and Data Structures

Methods

- Initialization, percept retrieval, movement, and actions.

Data Structures

- **Grid**: 4x4 2D list.
- **Agent State**: Position, direction, arrow, gold possession.
- **Percepts**: List of sensed environmental cues.

Step 0: Agent at [0, 0], Facing 1

Percepts: ['Breeze']

Action: Shoot Arrow

Step 1: Agent at [0, 0], Facing 1

Percepts: ['Breeze']

Action: Turn Right

Step 2: Agent at [0, 0], Facing 2

Percepts: ['Breeze']

Action: Move Forward

Step 3: Agent at [1, 0], Facing 2

Percepts: ['Stench', 'Breeze']

Action: Turn Left

Step 4: Agent at [1, 0], Facing 1

Percepts: ['Stench', 'Breeze']

Action: Grab Gold

Step 5: Agent at [1, 0], Facing 1

Percepts: ['Stench', 'Breeze']

Action: Move Forward

Step 6: Agent at [1, 1], Facing 1

Percepts: ['Breeze']

Action: Move Forward

Step 7: Agent at [1, 2], Facing 1

Percepts: ['Stench', 'Breeze']

Action: Move Forward

Step 8: Agent at [1, 3], Facing 1

Percepts: ['Glitter']

Action: Grab Gold