```python
from utils import *
from constants import *
import numpy as np

class Agent:
    def __init__(
            self,
            x: float = None,
            y: float = None,
            vx: float = 0.0,
            vy: float = 0.0,
            abs_velocity: float = AGENT_ABSOLUTE_VELOCITY,
            comm_dist: float = 0.0
            ):
        x = np.random.random()*1000 if x is None else x
        y = np.random.random()*1000 if y is None else y
        self.pos = np.array([x, y])
        self.velocity = np.array([vx, vy])
        self.abs_velocity = abs_velocity
        self.comm_dist = comm_dist
        self.target_pos = None
        self.inside_task_radius = False

    def update_pos(self):
        """
        Updates the current position of the agent by adding the self.velocity vector to the
        self.pos vector. When updating positions, the function disallows the agent to go out
        of bounds of the square grid spanning from (0, 0) to (1000, 1000). This means if the
        agent would go out of bounds by following its trajectory at its current absolute
        velocity, it instead moves in the same direction but at a lower absolute velocity, so
        that it stops at the border of the grid.
        """

        self.pos = self.pos + self.velocity
        self.pos = np.minimum(self.pos, 1000)
        self.pos = np.maximum(self.pos, 0)

    def update_velocity(self):
        """
        Updates velocity of agent according to following conditions:

        If it has reached a task (is inside task radius), it stops moving.

        If target_pos is specified (not None) and the agent has not reached it, sets agent
velocity
        towards that position.

        Otherwise, makes the agent's movement random by changing the velocity in each direction
        to some random number. Components vx and vy are set so that the absolute velocity sums
up to
        abs_velocity.
        """

        # Removes target_pos (should it be set) if agent is inside any task radius
        if self.inside_task_radius:
            self.target_pos = None
            self.velocity = np.zeros((2))
            return

        # Goes towards target_pos if specified and is not (almost) equal to pos
        if self.target_pos is not None and not np.allclose(self.pos, self.target_pos):
            self.velocity = self.target_pos - self.pos
            norm = np.linalg.norm(self.velocity)
            self.velocity = (self.velocity / norm) * np.minimum(self.abs_velocity, norm)
            return

        # If not, removes target_pos and initializes random movement
        self.target_pos = None

        # Sets velocity in each direction to random number in interval [-1, 1]
        self.velocity = (1 - (-1))*np.random.random(self.velocity.shape) - 1

        # Normalizing vector, making the norm of self.velocity equal to 1
        norm = np.linalg.norm(self.velocity)
        # To solve the unlikely case that both directional velicities are sampled as 0, we
        # randomly set vx = 1 or vy = 1 if that happens
        if norm == 0:
            self.velocity = np.array([1, 0]) if np.random.random() > 0.5 else np.array([0,
1])
            norm = np.linalg.norm(self.velocity)
        self.velocity = self.velocity / norm

        # Multiplying self.velocity (now a unit vector) with abs_velocity to achieve desired
        # (or random) velocity
        self.velocity = self.velocity * self.abs_velocity

    def callout(self, agents: list):
        """
        When the agent is within the task radius of any task, it emits a signal to other
agents
        within comm_dist to make them go towards that location, by setting their target_pos to
the
        position of the agent emitting the signal.

        The called upon agents will then go towards the coordinate from which the signal was
emitted until:
        a) they reach the signal location.
        b) they find themselves within a task radius themselves.
        The above conditions are checked for each agent when their velocities are updated.
        """

        # Send a signal to any agent within comm_dist
        for agent in agents:
            # Skips itself
            if agent == self:
                continue

            # Checking if the agent is within the comm_dist or is already within a task radius
            # (an agent which as already detected a nearby task will itself send out a signal,
            # and presumably would then be more interested in its own discovered task than the
            # signal of another agent)
            if (not agent.inside_task_radius) and distance_euclid(self.pos, agent.pos) <
self.comm_dist:
                agent.target_pos = self.pos

    def calloff(self, agents: list):
        """
        Performs the "opposite" action of callout: instead of giving agents within comm_dist a
target_pos,
        this function removes their target_pos if their previous target_pos was this agent's
current pos.

        This method is called from the task when it checks whether it should be completed.
        """

        # Send a signal to any agent within comm_dist
        for agent in agents:
            # Skips itself
            if agent == self or agent.target_pos is None:
                continue

            if np.allclose(self.pos, agent.target_pos) and distance_euclid(self.pos,
agent.pos) < self.comm_dist:
                agent.target_pos = None
```