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

exit = False

class Player:
    def __init__(self, hawk_percentage, dove_percentage, start_resources):
        self.hawk_percentage = hawk_percentage
        self.dove_percentage = dove_percentage
        self.resources = start_resources
        self.default_resources = start_resources
        self.wins = 0

    def play_strategy(self):
        r = random.random()
        if r < self.hawk_percentage:
            return "hawk"
        else:
            return "dove"

class Simulation:
    def __init__(self):
        self.root = tk.Tk()
        self.root.title("Hawk-Dove Simulation")
        self.menu_root = None

        self.player1_hawk_percentage = tk.StringVar()
        self.player1_dove_percentage = tk.StringVar()
        self.player2_hawk_percentage = tk.StringVar()
        self.player2_dove_percentage = tk.StringVar()
        self.start_resources = tk.StringVar()
        self.goal_resources = tk.StringVar()
        self.num_simulations = tk.StringVar()

        self.create_widgets()

    def create_widgets(self):
        custom_font = font.Font(size=int(1.5 *
font.nametofont("TkDefaultFont").actual()["size"]))
        tk.Label(self.root, text="Player 1 Hawk percentage ([0, 1]):", font =
custom_font).grid(row=0, column=0, padx=10, pady=5,)
        tk.Entry(self.root, textvariable=self.player1_hawk_percentage, font =
custom_font).grid(row=0, column=1)

        tk.Label(self.root, text="Player 1 Dove percentage ([0, 1]):", font =
custom_font).grid(row=1, column=0, padx=10, pady=5)
```

```
        tk.Entry(self.root, textvariable=self.player1_dove_percentage, font =
custom_font).grid(row=1, column=1)

        tk.Label(self.root, text="Player 2 Hawk percentage ([0, 1]):", font =
custom_font).grid(row=2, column=0, padx=10, pady=5)
        tk.Entry(self.root, textvariable=self.player2_hawk_percentage, font =
custom_font).grid(row=2, column=1)

        tk.Label(self.root, text="Player 2 Dove percentage ([0, 1]):", font =
custom_font).grid(row=3, column=0, padx=10, pady=5)
        tk.Entry(self.root, textvariable=self.player2_dove_percentage, font =
custom_font).grid(row=3, column=1)

        tk.Label(self.root, text="Amount of starting resources (>0):", font =
custom_font).grid(row=4, column=0, padx=10, pady=5)
        tk.Entry(self.root, textvariable=self.start_resources, font =
custom_font).grid(row=4, column=1)

        tk.Label(self.root, text="Win amount of resources (>start):", font =
custom_font).grid(row=5, column=0, padx=10, pady=5)
        tk.Entry(self.root, textvariable=self.goal_resources, font =
custom_font).grid(row=5, column=1)

        tk.Label(self.root, text="Number of simulations for testing (>0):",
font = custom_font).grid(row=6, column=0, padx=10, pady=5)
        tk.Entry(self.root, textvariable=self.num_simulations, font =
custom_font).grid(row=6, column=1)

        tk.Button(self.root, text="Start Simulation", font = custom_font,
command=self.validate_input).grid(row=7, column=0, columnspan=2, pady=10)

    def validate_input(self):
        player1_hawk = float(self.player1_hawk_percentage.get())
        player1_dove = float(self.player1_dove_percentage.get())
        player2_hawk = float(self.player2_hawk_percentage.get())
        player2_dove = float(self.player2_dove_percentage.get())
        start_resources = int(self.start_resources.get())
        goal_resources = int(self.goal_resources.get())
        num_simulations = int(self.num_simulations.get())

        if (
                0 <= player1_hawk <= 1
                and 0 <= player1_dove <= 1
                and 0 <= player2_hawk <= 1
                and 0 <= player2_dove <= 1
                and start_resources > 0
                and goal_resources > start_resources
                and num_simulations > 0
```

```
                and player1_hawk + player1_dove == 1
                and player2_hawk + player2_dove == 1
        ):
            self.show_main_menu()
        else:
            messagebox.showerror("Invalid Input", "Please enter valid input
values.")
            self.create_widgets()

    def show_main_menu(self):
        self.root.destroy()

        self.menu_root = tk.Tk()
        self.menu_root.title("Main Menu")
        custom_font = font.Font(size=3 *
font.nametofont("TkDefaultFont").actual()["size"])
        custom_font2 = font.Font(size=int(1.5 *
font.nametofont("TkDefaultFont").actual()["size"]))
        tk.Label(self.menu_root, text="Main Menu",
font=custom_font).pack(pady=10)
        tk.Button(self.menu_root, text="1. Play Hawk-Dove", font=custom_font2,
command=self.show_simulation_results).pack(pady=5)
        tk.Button(self.menu_root, text="2. Calculate Best
Strategies",font=custom_font2,
command=self.calculate_best_strategies).pack(pady=5)
        tk.Button(self.menu_root, text="3. Manually Set Strategies",
font=custom_font2, command=self.change_strategies).pack(pady=5)
        tk.Button(self.menu_root, text="4. Exit", font=custom_font2,
command=self.exit_simulation).pack(pady=5)

        self.menu_root.mainloop()

    def show_simulation_results(self):
        try:
            player1_hawk = float(self.player1_hawk_percentage.get())
            player1_dove = float(self.player1_dove_percentage.get())
            player2_hawk = float(self.player2_hawk_percentage.get())
            player2_dove = float(self.player2_dove_percentage.get())
            start_resources = int(self.start_resources.get())
            goal_resources = int(self.goal_resources.get())
            num_simulations = int(self.num_simulations.get())

            self.player1 = Player(player1_hawk, player1_dove, start_resources)
            self.player2 = Player(player2_hawk, player2_dove, start_resources)

            def clash(clash_player_1:Player, clash_player_2:Player):
                p1_strategy = clash_player_1.play_strategy()
                p2_strategy = clash_player_2.play_strategy()
```

```
                if p1_strategy == "hawk" and p2_strategy == "hawk":
                    # Both players pay tax, 50% chance of winning the new
resources
                    if random.random() < 0.5:
                        clash_player_1.resources += 1
                    else:
                        clash_player_2.resources += 1
                    clash_player_1.resources -= 2
                    clash_player_2.resources -= 2

                elif p1_strategy == "hawk" and p2_strategy == "dove":
                    # Player1's hawk wins and keeps 75% of resources
                    clash_player_1.resources += 3
                    clash_player_2.resources += 1

                elif p1_strategy == "dove" and p2_strategy == "hawk":
                    # Player2's hawk wins and keeps 75% of resources
                    clash_player_1.resources += 1
                    clash_player_2.resources += 3

                elif p1_strategy == "dove" and p2_strategy == "dove":
                    # Doves split the new resources equally
                    clash_player_1.resources += 1
                    clash_player_2.resources += 1

            def simulate_win_lose_game():
                player1_wins = 0
                player2_wins = 0

                for _ in range(num_simulations):
                    self.player1.resources = start_resources
                    self.player2.resources = start_resources
                    while (
                            self.player1.resources < goal_resources
                            and self.player2.resources < goal_resources
                            and self.player1.resources > 0
                            and self.player2.resources > 0
                    ):
                        clash(self.player1, self.player2)
                    if (
                            self.player1.resources >= goal_resources
                            or self.player2.resources <= 0
                    ):
                        # Player 1 wins
                        player1_wins += 1
                    elif (
                            self.player2.resources >= goal_resources
```

```
                            or self.player1.resources <= 0
                    ):
                        # Player 2 wins
                        player2_wins += 1

                player1_win_percentage = (player1_wins / num_simulations) *
100
                player2_win_percentage = (player2_wins / num_simulations) *
100

                result_root = tk.Tk()
                result_root.title("Simulation Results")

                custom_font = font.Font(size=int(1.5 *
font.nametofont("TkDefaultFont").actual()["size"]))

                label = tk.Label(result_root, text=f"Player 1
({self.player1.hawk_percentage}, {self.player1.dove_percentage})" +
                                 f", Player 2 ({self.player2.hawk_percentage},
{self.player2.dove_percentage})", font = custom_font)
                label.pack()

                tk.Label(result_root, text=f"Player 1 Win Percentage:
{player1_win_percentage:.2f}%", font = custom_font).pack(pady=5)
                tk.Label(result_root, text=f"Player 2 Win Percentage:
{player2_win_percentage:.2f}%", font = custom_font).pack(pady=5)

                tk.Button(result_root, text="Back to Main Menu", font =
custom_font, command=result_root.destroy).pack(pady=10)

                result_root.mainloop()

            simulate_win_lose_game()
        except ValueError:
            tk.messagebox.showerror("Error", "Invalid input. Please enter
valid numbers.")

    def calculate_best_strategies(self):


        def clash(clash_player_1:Player, clash_player_2:Player):
            p1_strategy = clash_player_1.play_strategy()
            p2_strategy = clash_player_2.play_strategy()

            if p1_strategy == "hawk" and p2_strategy == "hawk":
                # Both players pay tax, 50% chance of winning the new
resources
                if random.random() < 0.5:
                    clash_player_1.resources += 1
```

```
                else:
                    clash_player_2.resources += 1
                clash_player_1.resources -= 2
                clash_player_2.resources -= 2

            elif p1_strategy == "hawk" and p2_strategy == "dove":
                # Player1's hawk wins and keeps 75% of resources
                clash_player_1.resources += 3
                clash_player_2.resources += 1

            elif p1_strategy == "dove" and p2_strategy == "hawk":
                # Player2's hawk wins and keeps 75% of resources
                clash_player_1.resources += 1
                clash_player_2.resources += 3

            elif p1_strategy == "dove" and p2_strategy == "dove":
                # Doves split the new resources equally
                clash_player_1.resources += 1
                clash_player_2.resources += 1

        def simulate_resources_game(current_player:Player,
opponent_player:Player, num_simulations):
            for _ in range(num_simulations):
                current_player.resources = current_player.default_resources
                opponent_player.resources = opponent_player.default_resources
                for _ in range(250):
                    clash(current_player, opponent_player)
            return current_player.resources

        def best_strategy(current_player:Player, opponent_player:Player,
num_simulations, result_root):
            #     Analyzing current strategy against opponent
            current_average_resources = 0

            for _ in range(10):
                current_average_resources +=
simulate_resources_game(current_player, opponent_player, num_simulations)
            current_average_resources /= 10

            custom_font = font.Font(size=int(1.5 *
font.nametofont("TkDefaultFont").actual()["size"]))

            current_strategy_label = tk.Label(result_root, text="The average
resources using the current strategy is " + str(current_average_resources),
font = custom_font)
            current_strategy_label.pack(pady=5)

            #     Computing best strategy against opponent
```

```
            best_hawk = 1.0
            best_dove = 0.0
            best_average_resources = 0

            for i in range(1, 102):
                temp_hawk = abs((1 - i) / 100)
                temp_dove = abs(1 - temp_hawk)
                temp_player = Player(temp_hawk, temp_dove,
current_player.default_resources)
                temp_average_resources = 0

                for _ in range(10):
                    temp_average_resources +=
simulate_resources_game(temp_player, opponent_player, num_simulations)
                temp_average_resources /= 10

                if best_average_resources < temp_average_resources:
                    best_average_resources = temp_average_resources
                    best_hawk = round(temp_hawk, 2)
                    best_dove = round(temp_dove, 2)

            best_strategy_label = tk.Label(result_root, text="The best stable
mixed strategy against the opponent (" +
str(float(opponent_player.hawk_percentage)) + ", " +
str(float(opponent_player.dove_percentage)) + ") is ("
                                           + str(best_hawk) + ", " +
str(best_dove) + ")", font = custom_font)
            best_strategy_label.pack(pady=5)

            best_strategy_resources_label = tk.Label(result_root, text=("The
average resources using the best strategy is " + str(best_average_resources)),
font = custom_font)
            best_strategy_resources_label.pack(pady=5)

        try:
            result_root = tk.Tk()
            result_root.title("Strategy Results")

            player1_hawk = float(self.player1_hawk_percentage.get())
            player1_dove = float(self.player1_dove_percentage.get())
            player2_hawk = float(self.player2_hawk_percentage.get())
            player2_dove = float(self.player2_dove_percentage.get())
            start_resources = int(self.start_resources.get())
            num_simulations = int(self.num_simulations.get())

            self.player1 = Player(player1_hawk, player1_dove, start_resources)
            self.player2 = Player(player2_hawk, player2_dove, start_resources)
```

```
            custom_font = font.Font(size=int(2 *
font.nametofont("TkDefaultFont").actual()["size"]))
            custom_font2 = font.Font(size=int(1.5 *
font.nametofont("TkDefaultFont").actual()["size"]))

            label_player1 = tk.Label(result_root, text=f"Player 1
({player1_hawk}, {player1_dove})", font = custom_font2)
            label_player1.pack()
            best_strategy(self.player1, self.player2, num_simulations,
result_root)
            label_player2 = tk.Label(result_root, text=f"\nPlayer 2
({player2_hawk}, {player2_dove})", font = custom_font2)
            label_player2.pack()
            best_strategy(self.player2, self.player1, num_simulations,
result_root)

            tk.Button(result_root, text="Back to Main Menu", font =
custom_font, command=result_root.destroy).pack(pady=10)
            result_root.mainloop()

        except ValueError:
            tk.messagebox.showerror("Error", "Invalid input. Please enter
valid numbers.")

    def show_initial_menu(self):
        if (self.menu_root != None):
            self.menu_root.destroy()
            self.menu_root = None
        self.root = tk.Tk()
        self.root.title("Hawk-Dove Simulation")

        self.create_widgets()

    def change_strategies(self):
        self.menu_root.destroy()

    def exit_simulation(self):
        global exit
        exit = True
        self.menu_root.destroy()

    def run(self):
        self.root.mainloop()

while (exit == False):
    simulation = Simulation()
    simulation.run()
```

Усі модулі, яких потребує для роботи програма, автоматично йдуть разом з Python.

Через непослідовну поведінку модулю для графічного інтерфейсу tkinter програма знаходиться у безкінечному циклі, вийти з якого можна тільки кнопкою "Exit" у головному меню, а не хрестиком для закриття вікна.

Програма просить задати співвідношення голубів та яструбів гравців (числа у проміжку [0,1], які повинні сумуватись в одиницю).

Також програма просить стартову кількість ресурсів, кінцеву кількість, досягнувши якої гравець перемагає у грі, і кількість симуляцій гри, оскільки у залежності від динамічної стратегії результати декількох ігор можуть бути не репрезентабельні.

Перша кнопка підраховує скільки симуляцій гри переміг кожен з гравців.

Друга кнопка підраховує скільки в середньому отримують гравці ресурсів проти противника, і вираховує найкращу стратегію під стратегію противника.

Середня кількість ресурсів у другій кнопці може бути негативною, оскільки сама перемога чи поразка не є важливою стільки, скільки порівнювання якості стратегій.

Приклади вводу:

0.9 0.1 0.8 0.2 50 1000 200

0.5 0.5 0.5 0.5 100 750 300

0.7 0.3 0.4 0.6 200 500 400

1 0 0 1 300 2000 500