```
def draw_dartboard(ax, center, radius):
    ax.set_aspect('equal')
    ax.set_xlim([-radius, radius])
    ax.set_ylim([-radius, radius])

    circle = plt.Circle(center, radius, fill=False, color='black')
    ax.add_artist(circle)

    circle = plt.Circle(center, radius * 0.1, fill=False, color='black')
    ax.add_artist(circle)

    n = 20
    for i in range(n):
        angle = i * 2 * np.pi / n
        x = center[0] + radius * np.cos(angle)
        y = center[1] + radius * np.sin(angle)
        if radius * 0.15 <= np.sqrt(x ** 2 + y ** 2) <= radius * 0.5:
            continue
        line = plt.Line2D([center[0], x], [center[1], y], color='black', linestyle='--')
        ax.add_artist(line)

    circle = plt.Circle(center, radius * 0.9, fill=False, color='black')
    ax.add_artist(circle)

    circle = plt.Circle(center, radius * 0.6, fill=False, color='black')
    ax.add_artist(circle)

    circle = plt.Circle(center, radius * 0.5, fill=False, color='black')
    ax.add_artist(circle)

    circle = plt.Circle(center, radius * 0.15, fill=False, color='black')
    ax.add_artist(circle)

    circle = plt.Circle(center, radius * 0.1, fill=False, color='black')
    ax.add_artist(circle)
    ax.set_ylim(1.73-0.2255*1.5, 1.73+0.2255*1.5)
    ax.set_xlim(-0.2255*1.5, 0.2255*1.5)

import numpy as np
import matplotlib.pyplot as plt


def calculate_trajectory(alpha, beta, V0, difficulty, x_target=2.37, m=0.025, k=0.05):
    alpha = np.radians(alpha)
    beta = np.radians(beta)
    ramda = np.arccos(np.sqrt(1 - np.cos(alpha)**2 - np.cos(beta)**2))

    if difficulty == "beginner":
```

```
        V_prime = 5
    elif difficulty == "intermediate":
        V_prime = 10
    elif difficulty == "advanced":
        V_prime = 15
    else:
        raise ValueError("Invalid difficulty level. Choose from 'beginner', 'intermediate', or
'advanced'.")

    # Calculate time of flight using t = R / V0cos(alpha)
    t_flight = x_target / (V0 * np.cos(alpha))

    # Calculate x, y, and z positions over time
    t = np.linspace(0, t_flight, num=1000)
    x_vals = V0 * np.cos(alpha) * t
    y_vals = V0 * np.sin(beta) * t - 0.5 * 9.81 * t**2
    z_vals = (V0 * np.cos(ramda) + V_prime) * m / k * (1 - np.exp(-k * t / m)) - V_prime * t

    return x_vals, y_vals, z_vals

def plot_all_graphs(x_vals, y_vals, z_vals):
    fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(30, 5))

    center = (0, 1.73)
    radius = 0.2255

    draw_dartboard(ax1, center, radius)
    ax1.scatter(z_vals[-1], y_vals[-1], s=50, c='r', marker='o', label='Dart arrival point')
    ax1.set_xlabel('z')
    ax1.set_ylabel('y')
    ax1.set_title('Dart Arrival Point on (y, z) plane')
    ax1.legend()
    ax1.grid()

    # Plot trajectory on (x, y) plane
    ax2.plot(x_vals, y_vals, label='Trajectory')
    ax2.scatter(x_vals[-1], y_vals[-1], label='Dart arrival point', color='red')
    ax2.set_xlabel('x')
    ax3.set_xlim(0, 3)
    ax2.set_ylabel('y')
    ax2.set_title('Dart Trajectory on (x, y) plane')
    ax2.axvline(2.37, color='black', linestyle='--', label='Location of the Dart Board')
    ax2.legend()
    ax2.grid()

    # Plot trajectory on (x, z) plane
    ax3.plot(x_vals, z_vals, label='Trajectory')
    ax3.scatter(x_vals[-1], z_vals[-1], label='Dart arrival point', color='red')
```

```
    ax3.set_xlabel('x')
    ax3.set_xlim(0, 3)
    ax3.set_ylabel('z')
    ax3.set_title('Dart Trajectory on (x, z) plane')
    ax3.axvline(2.37, color='black', linestyle='--', label='Location of the Dart Board')
    ax3.legend()
    ax3.grid()

    plt.show()

if __name__ == "__main__":
    try:
        alpha = float(input("Enter alpha angle (in degrees): "))
        beta = float(input("Enter beta angle (in degrees): "))
        V0 = float(input("Enter initial velocity (in m/s): "))
        difficulty = input("Enter difficulty level (beginner, intermediate, advanced): ").lower()
        x_vals, y_vals, z_vals = calculate_trajectory(alpha, beta, V0, difficulty)
        plot_all_graphs(x_vals, y_vals, z_vals)
    except ValueError as e:
        print(e)

print("The dart hits the target at (z, y) = ({:.2f}, {:.2f})".format(z_vals[-1], y_vals[-1]))
```

이 코드에 대한 4분짜리 발표를 할거야. 영어로 할거고. 영어 못하니까 쉬운 단어들로 구성해줘. 아래는 들어가야 할 내용.

- 다음으로, 다트 게임에 포물선 운동과 공기 저항 개념을 적용해보았다.
- 플레이어가 3개의 값을 조정하여, 3D 공간에서 다트 중심에 더 가까이 맞추는 게임을 만든 것이다.
- players can…
    - set the initial angle and velocity of the dart to control the Projectile Motion
    - set another initial angle of the dart to overcome air resistance (공기 저항이 파란색 화살표 방향으로 있으며, 주황색 각도를 조정해서 다트가 꽂히는 방향을 조정할 수 있는 것이다.)

In the next version of the dart game, players can control three values to get the dart as close to the center as possible in a 3D space. We can set the initial alpha angle and velocity of the dart to control the projectile motion, and set another initial beta angle to overcome air resistance. **The direction of air resistance is indicated by the blue arrow, and players can adjust the orange angle to control the direction in which the dart lands.**

- 우리 다트의 궤적은 그림과 같다. x, y, z축이 있고, 노란색 화살표가 다트의 진행방향이다.

The trajectory of our dart is shown in the picture, where we have x, y, and z axes, and the yellow arrow represents the direction of the dart's motion.

- force of air resistance를 구하는 방법은 속도를 한 번 곱하는 경우와, 두 번 곱하는 경우가 있다. 코드 설명하면서도 언급할거지만, constant인 k 값을 0.05로 설정했다.

To calculate the force of air resistance, we have two options - we can either multiply the velocity once or twice. We will compare the results from these two formulas. Also, we set the constant value 'k' to 0.05.

- 왼쪽은 draw_dartboard 함수의 정의이다. 앞에서 그린 다트는 cm단위로 보여줬는데, m단위로 보여주기 위해 새로운 draw_dartboard 함수를 정의하였다.
- 가장 큰 네모 박스 부분은 플레이어의 입력값 3개와, 플레이 레벨을 입력받는 곳이다.
- 그리고 우리의 코드는 최종적으로 다트 보드의 어떤 지점에 찍혔는지 좌표로 결과를 보여준다.

To display the darts in meters instead of centimeters, we defined a new draw_dartboard function. The largest rectangular box is where the player inputs three values and the play level. Finally, our code shows the results of where the dart hit on the dartboard in coordinates.

- 그 다음, calculate_trajectory라는 함수를 정의했다. force of air resistance에 따라 두 가지 버전이 있다. 지정된 두 개의 각도와 속도에 따라 다트의 어느 위치에 꽂히는지 return하는 함수이다.
- 핑크색 박스는 적절하지 않는 각도가 입력되었을 때, arccos값이 invalid하다는 결과를 출력하는 코드이다. 대략 30도 이상의 값을 입력해야 적절한 결과를 얻을 수 있다.
- 이 코드 중간에 beginner, intermediate, advanced의 경우가 분류되어 있는데, 우리 시뮬레이션은 이 레벨에 따라 바람의 저항을 더 세게 불도록 설정했다.
- 주황색 박스 안의 코드가 포물선 운동의 결과로 X, Y축의 좌표를 계산하는 코드이다.
- 빨간색과 파란색 박스 안의 코드는 공기 저항에 따른 Z축 방향으로의 위치를 계산하는 코드이다. 빨간색이 F=-kv일 경우, 파란색이 F=-kv^2이다.

Moving on, we have defined a new function called 'calculate_trajectory'. This function has two versions depending on the force of air resistance. It takes in two speciied angles and velocity and returns the position at which the dart lands.

In the pink box, we have included code that displays an error message if an inappropriate angle is entered. This is because the arccos value becomes invalid if the angle is less than 30 degrees.

In the middle of the code, we have categorized the levels as beginner, intermediate, and advanced. Depending on the level, we have set the wind resistance to be stronger or weaker.

The code inside the orange box calculates the X and Y coordinates of the projectile motion.

Finally, the code inside the red and blue boxes calculates the Z coordinate of the projectile motion depending on the force of air resistance. The red box is for when the force is -kv and the blue box is for when the force is -kv^2.

- plot_all_graphs에서는 총 3개의 그래프를 그린다. 첫 번째는 다트 보드, 두 번째는 XY평면 상의 포물선 그래프, 세 번째는 XZ평면 상에서 공기저항에 의해 휘어지는 다트의 궤적을 그리는 코드이다.

In the function plot_all_graphs, three different graphs are generated. The first graph shows the dartboard, the second one shows the trajectory of the dart in the XY-plane, and the third one shows the trajectory of the dart in the XZ-plane.

- 최종적으로 이 코드를 실행하면 이와 같은 결과가 한 번에 나온다.
- 이건 제가 값을 선택해서 가장 중심과 가깝게 나온 결과입니다. 초기 속도를 6.5m/s로 설정하고, 포물선의 각도를 35도, 공기저항을 이겨내기 위한 각도를 70도라고 설정하면, 다트의 중앙에 거의 가까운, (0.13, 0.02)에 꽂히는 결과를 얻을 수 있었습니다.

When this code is executed, the result is displayed in a single output. To obtain a result that is close to the center of the dartboard, I selected an initial velocity of 6.5 m/s, an angle of 35 degrees for the trajectory, and an angle of 70 degrees to counteract the effect of air resistance. The dart hits the board very close to the center, at the coordinates (0.13, 0.02).

- 이건 같은 입력값을 넣었을 때 F=-kv^2인 경우의 결과입니다. 다트의 중심에서 너무 멀리 떨어져 있어 다트판에 표시되지 않는 상태입니다.

This is the result for the case where F=-kv^2 with the same input values. The dart is located too far from the center to be displayed on the dartboard.

- 이제 what did not worked에 대해 간단히 설명하겠습니다. 흔히 휴대폰에서 하는 게임과 같이, 마우스로 다트를 당겨서 초기값들을 설정하고 다트를 던지는 3D simulation을 구현하고 싶었습니다.
- html 혹은 pygame이라는 도구를 활용하면 구현할 수는 있다고 합니다. 하지만 저희가 익숙하지 않은 툴이라 구현에 어려움을 겪었고, 결국 2D 그래프로 결과를 따로 보여주는 데에서 그친 점이 아쉽습니다.
- air resistance의 정도를 advanced로 설정하면, 다트가 굉장히 많이 휘고 조절이 어려워서 중심을 맞추는 값을 찾기가 너무 어려웠습니다. 더 세밀하게 다트를 조절할 수 있는 코드로 보완하지 못한 점이 아쉽습니다.

Now, let me briefly explain what did not work. We wanted to implement a 3D simulation where we could drag the dart with the mouse to set the initial values and throw it, like the games we often play on our phones. We learned that it is possible to implement this / using tools such as html or pygame, but we struggled with implementation because we were not familiar with these tools, and ultimately, we were only able to show the results separately in a 2D graph.

When we set the air resistance to 'advanced', the dart curved too much, making it difficult to control and find the values to hit the center. We wanted to improve the code to allow for more precise dart control.

- 가지고 있는 지식을 가지고 원하는 코드를 짜보는 것은 소중한 경험이었습니다.
- 코딩은 그 자체가 학문이 아니라, 새롭고 유의미한 결과를 내기 위한 도구라는 점도 깨닫게 되었습니다.
- 코딩이 필요한 다른 주제도 충분히 해낼 수 있다는 자신감이 생겼습니다.

Working on this code using the knowledge we have was a valuable experience.

We learned that coding is a tool that can be used to produce novel and meaningful results, rather than a field of study in its own right.

We also gained confidence that we can handle other topics that require coding.

[Final Script]

V **In the next version of the dart game, players can control three values to get the dart / as close to the center as possible / in a 3D space.** We can set the initial alpha angle and velocity of the dart to control the projectile motion, and set another initial beta angle to overcome air resistance. **The direction of air resistance is indicated by the blue arrow.**

V **The trajectory of our dart is shown in the picture, where we have x, y, and z axes, and the yellow arrow represents the direction of the dart's motion. The velocity along the xyz axes is obtained / by taking the dot product of the v vector / with the unit vectors along the xyz axes, which are given by cosine alpha, beta, and gamma.**

V **To calculate the force of air resistance, we have two options - we can either multiply the velocity once or twice. We will compare the results from these two formulas. Also, we set the constant value 'k' to 0.05.**

(1:10)

V **To display the darts in meters instead of centimeters, we defined a new / draw_dartboard function. The largest rectangular box is where / the player inputs three values and the play level. Finally, our code shows the results of where the dart hit on the dartboard in coordinates.**

V **Moving on, we have defined a new function called 'calculate_trajectory'. This function has two versions / depending on the force of air resistance. It takes in / two specified angles and velocity and returns the position at which the dart lands.**

In the pink box, we have included code that displays an error message if an inappropriate angle is entered. This is because, if the arccosine value goes beyond the range of -1 and 1, there is no valid angle that can satisfy the equation geometrically.

(In the middle of the code, we have categorized the levels as beginner, intermediate, and advanced. Depending on the level, we have set the air resistance to be stronger or weaker.)

The code inside the orange box calculates the X and Y coordinates of the projectile motion.

Finally, the code inside the red and blue boxes calculates the Z coordinate depending on the force of air resistance. The red box is for when the force is -kv and the blue box is for when the force is -kv^2.

(2:50)

V **In the function plot_all_graphs, three different graphs are generated. The first graph shows the dartboard, the second one shows the trajectory of the dart in the XY-plane, and the third one shows the XZ-plane.**

V **When this code is executed, the result is displayed in a single output. To obtain a result that is close to the center of the dartboard, I selected an initial velocity**

of 6.5 m/s, an angle of 35 degrees for the trajectory, and an angle of 70 degrees to counteract the effect of air resistance. The dart hits the board very close to the center, at the coordinates (0.13, 0.02).

V This is the result for the case where F=-kv^2 with the same input values. The dart is located too far from the center to be displayed on the dartboard.

(4:40)

V Now, let me briefly explain what did not work. We wanted to implement a 3D simulation where we could drag the dart with the mouse to set the initial values and throw it, like the games we often play on our phones. We learned that it is possible to implement this / using tools such as html or pygame, but we struggled with implementation because we were not familiar with these tools, and ultimately, we were only able to show the results separately in a 2D graph.

V When we set the air resistance to 'advanced', the dart curved too much, making it difficult to control and find the values to hit the center. We wanted to improve the code / to allow for / more precise dart control.

(5:10)

- **V Working on this code / using the knowledge we have / was a valuable experience.**
- **We learned that coding is a tool / that can be used to produce novel and meaningful results, rather than a field of study in its own right.**
- **We also gained confidence that we can handle other topics that require coding.**