**MINI PROJECT**

SRN - 202101425

Name: Shravan Sudhir Meshram

Roll no. 31

Problem statement: Implement any two Player game.

* **Intro of the game:**

Welcome to Air Hockey Battle!

Step onto the virtual air hockey arena and prepare for an exhilarating clash of skill, speed, and strategy. Whether you're a seasoned player seeking a formidable challenge or a newcomer eager to test your mettle, Air Hockey Battle offers an electrifying experience for players of all levels.

Choose your mode:

1 Player (vs. AI): Enter the arena and face off against an AI opponent programmed to push you to your limits. Adapt your tactics, anticipate your opponent's moves, and master the art of precision to emerge victorious.

2 Players (vs. Computer): Grab a friend and engage in a thrilling head-to-head battle for supremacy. With split-screen action and intuitive controls, challenge each other to intense matches where every shot counts.

Immerse yourself in the heart-pounding excitement of air hockey, where lightning-fast reflexes and razor-sharp focus are the keys to success. With vibrant visuals, dynamic gameplay, and responsive controls, Air Hockey Battle delivers an adrenaline-fueled experience that will keep you coming back for more.

Are you ready to take your place in the arena? The battle awaits!

* **What are the playing and non-playing characters:**

In Air Hockey Battle, the playing characters are the paddles controlled by the players. These paddles are used to hit the puck and defend the goal.

Player 1: The first player controls one of the paddles, aiming to strike the puck towards the opponent's goal while defending their own.

Player 2: In two-player mode, the second player controls the opposing paddle, striving to outmaneuver their opponent and score goals while preventing goals from being scored against them.

The non-playing characters in the game include:

AI Opponent: In single-player mode, the AI opponent serves as the adversary for the player. It controls its own paddle and employs strategies to compete against the player, providing a challenging and engaging gameplay experience.

Game Over Screen: This screen appears when one of the players reaches the score limit, signalling the end of the game. It displays the final scores and declares the winner, allowing players to celebrate their victory or contemplate their defeat before starting a new game.

These characters add depth and excitement to Air Hockey Battle, creating an immersive and competitive atmosphere for players to enjoy.

* **Rules of the game:**

The rules of Air Hockey Battle are simple yet engaging, providing a dynamic gameplay experience for players. Here are the rules:

* **Objective:** The primary objective of Air Hockey Battle is to score goals by hitting the puck into the opponent's goal while defending your own goal.
* **Setup:** The game is played on a smooth playing surface with two goals placed at opposite ends. Each player controls a paddle, which they use to strike the puck.
* **Scoring:** A player scores a point when the puck enters their opponent's goal. The game typically has a predetermined score limit, and the first player to reach or exceed this limit wins the game.
* **Paddle Control:** Players control their paddles using designated controls, such as arrow keys or mouse movements. They can move their paddles horizontally and vertically to intercept the puck and direct it towards the opponent's goal.
* **Puck Movement:** The puck moves across the playing surface in response to player actions and interactions with the paddles. It can bounce off the walls and paddles, changing direction and speed dynamically.
* **Gameplay:** Players take turns striking the puck, attempting to outmaneuver their opponent and score goals. They must also defend their own goal by blocking shots from their opponent.
* **Fouls:** Common fouls in Air Hockey Battle include touching the puck with any part of the body other than the paddle and blocking the opponent's view of the puck. Fouls may result in penalties or the opponent being awarded a point.
* **Game Over:** The game concludes when one of the players reaches the predetermined score limit. At this point, the game displays a "Game Over" screen, showing the final scores and declaring the winner.
* **AI algo used:**

The AI algorithm used in this Air Hockey game is a simple heuristic-based approach. Here's how it works:

* Heuristic Function: The heuristic function determines the movement of the AI paddle based on the current state of the game. It calculates the distance between the puck and the center of the AI paddle. If the puck is far from the paddle, the AI paddle moves towards the center. If the puck is close to the paddle, the AI paddle tries to intercept it by moving towards the puck's predicted position.
* Prediction: The AI doesn't predict the puck's exact trajectory but rather makes decisions based on the current state of the game. It reacts to the puck's position and adjusts its movement accordingly.
* Decision Making: The AI paddle decides whether to move up, down, or stay still based on the heuristic function's output. If the puck is above the paddle, it moves up; if below, it moves down. If the puck is at the same height as the paddle, it stops moving.
* Simplicity: This AI algorithm is relatively simple and doesn't involve complex calculations or predictive models. It's designed to provide a basic level of challenge for the player while being computationally efficient and easy to implement.
* **Why that algo was used:**

The heuristic-based AI algorithm was likely chosen for several reasons:

* Simplicity: Heuristic algorithms are often straightforward to implement and understand. They don't require complex mathematical calculations or extensive training data, making them suitable for simpler games like Air Hockey.
* Real-time Performance: Heuristic algorithms are computationally efficient and can make decisions in real-time without significant processing delays. In fast-paced games like Air Hockey, where quick reactions are essential, this real-time performance is crucial.
* Ease of Tuning: Heuristic algorithms allow developers to fine-tune the AI's behavior easily by adjusting parameters or heuristics. This flexibility makes it possible to balance the game's difficulty level and provide an appropriate challenge for players of varying skill levels.
* No Training Required: Unlike machine learning-based approaches, heuristic algorithms don't require training on large datasets. This eliminates the need for extensive data collection and training time, simplifying the development process.
* Suitability for Simple Games: Air Hockey is a relatively simple game with straightforward mechanics and limited state space. Heuristic algorithms can effectively handle the decision-making process in such games without the need for complex AI techniques.
* **Is the game balanced:**

The case of the Air Hockey game implemented with a heuristic-based AI algorithm, here are some considerations for assessing its balance:

* AI Difficulty: The AI opponent's difficulty level should provide an appropriate challenge for players while remaining beatable with practice. If the AI is too easy or too difficult to defeat consistently, players may become frustrated or disinterested.
* Player Versus AI Interaction: The interaction between the player and the AI should feel engaging and competitive. The AI's responses to the player's actions should be reasonable and coherent, enhancing the overall gaming experience.
* Game Progression: The game's progression, including score tracking and difficulty scaling, should feel balanced and rewarding. Players should feel a sense of accomplishment as they improve their skills and progress through the game.
* Fairness: The game mechanics and AI behavior should feel fair to the player. Unpredictable or unfair AI actions, such as unrealistic puck movements or impossible-to-block shots, can detract from the game's balance and enjoyment.
* Feedback Mechanisms: The game should provide clear feedback to the player, indicating their performance and the AI's actions. Visual and auditory cues can help players understand the game state and make informed decisions.
* Adjustable Parameters: Providing options to adjust game parameters, such as AI difficulty or puck speed, can enhance the game's balance by allowing players to tailor the experience to their preferences and skill level.
* **Is the game adaptive:**

In the case of the Air Hockey game implemented with a heuristic-based AI algorithm, its adaptiveness can be evaluated based on several factors:

* AI Difficulty Scaling: The game could potentially adjust the difficulty of the AI opponent based on the player's performance. For example, if the player consistently wins matches, the AI's skill level could increase incrementally to provide a greater challenge. Conversely, if the player struggles or loses matches frequently, the AI's difficulty could decrease to maintain an appropriate level of challenge without becoming frustrating.
* Puck Speed and Game Speed: The game could offer options to adjust the speed of the puck or the overall game speed. Allowing players to customize these parameters can accommodate different skill levels and play styles, making the game more adaptive to individual preferences.
* Feedback and Tutorial Systems: Adaptive feedback systems or tutorials can help players improve their skills by providing guidance or hints during gameplay. For example, if a player consistently fails to block shots from the AI, the game could offer tips or visual cues to help them improve their timing or positioning.
* Dynamic AI Behavior: The AI opponent could adapt its strategies based on the player's actions or patterns. For instance, if the player consistently favors a particular shot or movement, the AI could adjust its defensive or offensive tactics accordingly to provide a more challenging and varied experience.
* Personalized Challenges: Offering personalized challenges or achievements based on the player's performance can add an adaptive element to the game. By dynamically generating challenges tailored to the player's skill level or progress, the game can keep players engaged and motivated to continue playing.
* **Feedback to the player:**

Providing effective feedback to players is crucial for enhancing their experience, guiding their progress, and keeping them engaged. Here are some ways to provide feedback in the Air Hockey game:

* Visual Feedback: Use visual cues to indicate important events or actions during gameplay. For example, when the puck collides with a paddle, you can briefly highlight the impact point or display a visual effect to signify the collision. Additionally, displaying the trajectory of the puck after each collision can help players understand the outcome of their shots and improve their strategy.
* Auditory Feedback: Incorporate sound effects to complement visual feedback and reinforce player actions. For instance, you can use distinct sound effects for puck collisions, goals scored, or paddle movements. Adjusting the volume or pitch of these sounds based on the intensity of the action can enhance immersion and provide valuable feedback to players.
* Score Updates: Clearly display the current score on the screen to inform players about their progress and performance. Consider using dynamic animations or transitions to update the score visually whenever a goal is scored. Additionally, you can accompany score updates with brief messages or animations to celebrate successful shots or acknowledge milestones reached by the players.
* Error Messages: Provide clear and concise error messages to help players understand why certain actions failed or were invalid. For example, if a player attempts an illegal move or collides with the puck in an unintended way, display a message explaining the issue and suggesting possible solutions. Avoid generic error messages and strive to offer actionable feedback that assists players in improving their skills.
* Tutorial and Guidance: Integrate tutorial elements or guidance prompts to introduce new players to the game mechanics and controls gradually. Offer interactive tutorials or practice modes where players can learn basic and advanced techniques in a controlled environment. Provide feedback during tutorials to acknowledge correct actions and offer corrective feedback for mistakes.
* Performance Metrics: Track and display performance metrics such as accuracy, reaction time, and goal-to-goal duration to help players assess their skills and progress over time. Use visual representations such as graphs or charts to present these metrics in an easily understandable format. Offer suggestions or challenges based on performance metrics to encourage players to improve and strive for better outcomes.

Code:

**Opening Screen**

import pygame

import sys

# Initialize Pygame

pygame.init()

# Set up the window

WIDTH, HEIGHT = 800, 600

screen = pygame.display.set\_mode((WIDTH, HEIGHT))

pygame.display.set\_caption("Air Hockey")

# Define colors

WHITE = (255, 255, 255)

BLACK = (0, 0, 0)

# Define game modes

AI\_GAME = 1

COMPUTER\_GAME = 2

# Load background image

background\_image = pygame.image.load("1.jpg")  # Replace "background.jpg" with your image file

# Define functions for game modes

def launch\_ai\_game():

    import ai\_game

    ai\_game.main()

def launch\_computer\_game():

    import computer\_game

    computer\_game.main()

# Main opening screen loop

def opening\_screen():

    font = pygame.font.SysFont(None, 48)

    selection\_text = font.render("Select Game Mode:", True, WHITE)

    ai\_text = font.render("1 Player (AI)", True, WHITE)

    computer\_text = font.render("2 Players (Computer)", True, WHITE)

    selection\_text\_rect = selection\_text.get\_rect(center=(WIDTH // 2, HEIGHT // 2 - 50))

    ai\_text\_rect = ai\_text.get\_rect(center=(WIDTH // 2, HEIGHT // 2))

    computer\_text\_rect = computer\_text.get\_rect(center=(WIDTH // 2, HEIGHT // 2 + 50))

    while True:

        for event in pygame.event.get():

            if event.type == pygame.QUIT:

                pygame.quit()

                sys.exit()

            elif event.type == pygame.KEYDOWN:

                if event.key == pygame.K\_1:

                    launch\_ai\_game()

                elif event.key == pygame.K\_2:

                    launch\_computer\_game()

        screen.blit(background\_image, (0, 0))  # Draw background image

        screen.blit(selection\_text, selection\_text\_rect)

        screen.blit(ai\_text, ai\_text\_rect)

        screen.blit(computer\_text, computer\_text\_rect)

        pygame.display.flip()

if \_\_name\_\_ == "\_\_main\_\_":

    opening\_screen()

**AI GAME**

import pygame

import sys

import random

# Initialize Pygame

pygame.init()

# Set up the window

WIDTH, HEIGHT = 800, 600

screen = pygame.display.set\_mode((WIDTH, HEIGHT))

pygame.display.set\_caption("Air Hockey")

# Load background image

background\_image = pygame.image.load("2.jpg")  # Replace "2.jpg" with your image file

# Define colors

WHITE = (255, 255, 255)

RED = (255, 0, 0)

BLUE = (0, 0, 255)

GRAY = (200, 200, 200)  # Light gray color for scoreboard

BLACK = (0, 0, 0)  # Define the BLACK color constant

# Define game elements

class Paddle:

    def \_\_init\_\_(self, x, y, color):

        self.rect = pygame.Rect(x, y, 20, 100)

        self.color = color

        self.speed = 10  # Increased speed

        self.direction = 0  # 0 for not moving, -1 for up, 1 for down

    def update(self):

        self.rect.y += self.direction \* self.speed

        if self.rect.top < 0:

            self.rect.top = 0

        elif self.rect.bottom > HEIGHT:

            self.rect.bottom = HEIGHT

    def draw(self):

        pygame.draw.rect(screen, self.color, self.rect)

class Puck:

    def \_\_init\_\_(self, x, y):

        self.rect = pygame.Rect(x, y, 20, 20)

        self.color = WHITE  # Initial color of the puck

        self.speed\_x = 7

        self.speed\_y = 7

        self.glow\_rate = 5  # Rate of color transition

        self.glowing\_up = True  # Flag to indicate if glowing is increasing or decreasing

        self.collision\_point = None  # Point of collision with paddle

    def update\_color(self):

        if self.glowing\_up:

            self.color = (min(self.color[0] + self.glow\_rate, 255),

                          min(self.color[1] + self.glow\_rate, 255),

                          min(self.color[2] + self.glow\_rate, 255))

            if self.color == (255, 255, 255):  # If reached maximum brightness, start decreasing

                self.glowing\_up = False

        else:

            self.color = (max(self.color[0] - self.glow\_rate, 0),

                          max(self.color[1] - self.glow\_rate, 0),

                          max(self.color[2] - self.glow\_rate, 0))

            if self.color == (0, 0, 0):  # If reached minimum brightness, start increasing

                self.glowing\_up = True

    def update(self):

        self.rect.x += self.speed\_x

        self.rect.y += self.speed\_y

        # Check for collisions with top and bottom walls

        if self.rect.top <= 0 or self.rect.bottom >= HEIGHT:

            self.speed\_y = -self.speed\_y

    def draw(self):

        pygame.draw.ellipse(screen, self.color, self.rect)

        if self.collision\_point:

            pygame.draw.line(screen, WHITE, self.rect.center, self.collision\_point, 2)

def move\_ai\_paddle(ai\_paddle, puck):

    if puck.rect.centery < ai\_paddle.rect.centery:

        ai\_paddle.direction = -1

    elif puck.rect.centery > ai\_paddle.rect.centery:

        ai\_paddle.direction = 1

    else:

        ai\_paddle.direction = 0

def check\_paddle\_collision(paddle, puck):

    if paddle.rect.colliderect(puck.rect):

        # Calculate collision point based on the side of the paddle

        if puck.speed\_x > 0:  # Puck moving to the right

            collision\_x = WIDTH - puck.rect.width // 2  # Collision point on the right wall

        else:  # Puck moving to the left

            collision\_x = puck.rect.width // 2  # Collision point on the left wall

        collision\_y = HEIGHT // 2  # Collision point at the center of the window

        puck.collision\_point = (collision\_x, collision\_y)

        puck.speed\_x = -puck.speed\_x

    else:

        puck.collision\_point = None

def reset\_game(player\_paddle, ai\_paddle, puck):

    puck.rect.center = (WIDTH // 2, HEIGHT // 2)

    player\_paddle.rect.center = (50, HEIGHT // 2)

    ai\_paddle.rect.center = (WIDTH - 70, HEIGHT // 2)

    puck.speed\_x = random.choice([-7, 7])

    puck.speed\_y = random.choice([-7, 7])

def check\_goal(player\_paddle, puck):

    if puck.rect.left <= 0:

        return True

    elif puck.rect.right >= WIDTH:

        return True

    return False

# Define the SCORE\_LIMIT constant

SCORE\_LIMIT = 2  # Adjust this value as needed

# Main game loop

def start\_screen():

    # Load the background image for the start screen

    start\_screen\_background = pygame.image.load("1.jpg")

    start\_screen\_background = pygame.transform.scale(start\_screen\_background, (WIDTH, HEIGHT))

    font = pygame.font.SysFont(None, 72)

    title\_text = font.render("Air Hockey", True, WHITE)

    start\_text = font.render("Press any key to start", True, WHITE)

    title\_text\_rect = title\_text.get\_rect(center=(WIDTH // 2, HEIGHT // 2 - 50))

    start\_text\_rect = start\_text.get\_rect(center=(WIDTH // 2, HEIGHT // 2 + 50))

    screen.blit(start\_screen\_background, (0, 0))  # Corrected line

    screen.blit(title\_text, title\_text\_rect)

    screen.blit(start\_text, start\_text\_rect)

    pygame.display.flip()

    waiting = True

    while waiting:

        for event in pygame.event.get():

            if event.type == pygame.QUIT:

                pygame.quit()

                sys.exit()

            elif event.type == pygame.KEYDOWN:

                waiting = False

# Heuristic function for AI paddle movement

def heuristic(player\_paddle, ai\_paddle, puck):

    # Calculate the distance between the puck and the center of the AI paddle

    distance = abs(puck.rect.centery - ai\_paddle.rect.centery)

    # If the puck is far from the AI paddle, move the paddle towards the center

    if distance > 20:

        if puck.rect.centery < ai\_paddle.rect.centery:

            ai\_paddle.direction = -1

        elif puck.rect.centery > ai\_paddle.rect.centery:

            ai\_paddle.direction = 1

    # If the puck is close to the AI paddle, try to intercept it

    else:

        if puck.rect.centery < ai\_paddle.rect.centery:

            ai\_paddle.direction = -1

        elif puck.rect.centery > ai\_paddle.rect.centery:

            ai\_paddle.direction = 1

        else:

            ai\_paddle.direction = 0

# Main game loop

def main():

    start\_screen()

    player\_paddle = Paddle(50, HEIGHT // 2 - 50, RED)

    ai\_paddle = Paddle(WIDTH - 70, HEIGHT // 2 - 50, BLUE)

    puck = Puck(WIDTH // 2 - 10, HEIGHT // 2 - 10)

    player\_score = 0

    ai\_score = 0

    font = pygame.font.SysFont(None, 36)

    while True:

        # Handle events

        for event in pygame.event.get():

            if event.type == pygame.QUIT:

                pygame.quit()

                sys.exit()

            elif event.type == pygame.KEYDOWN:

                if event.key == pygame.K\_ESCAPE:

                    pygame.quit()

                    sys.exit()

                elif event.key == pygame.K\_UP:

                    player\_paddle.direction = -1

                elif event.key == pygame.K\_DOWN:

                    player\_paddle.direction = 1

            elif event.type == pygame.KEYUP:

                if event.key == pygame.K\_UP or event.key == pygame.K\_DOWN:

                    player\_paddle.direction = 0

        # Update game state

        heuristic(player\_paddle, ai\_paddle, puck)  # Call the heuristic function

        player\_paddle.update()

        ai\_paddle.update()

        puck.update()

        check\_paddle\_collision(player\_paddle, puck)

        check\_paddle\_collision(ai\_paddle, puck)

        puck.update\_color()

        # Check if the puck is out of bounds

        if puck.rect.left <= 0:

            ai\_score += 1

            reset\_game(player\_paddle, ai\_paddle, puck)

        elif puck.rect.right >= WIDTH:

            player\_score += 1

            reset\_game(player\_paddle, ai\_paddle, puck)

        # Check for game over condition

        if player\_score >= SCORE\_LIMIT or ai\_score >= SCORE\_LIMIT:

            game\_over(player\_score, ai\_score)

            player\_score = 0

            ai\_score = 0

            reset\_game(player\_paddle, ai\_paddle, puck)

        # Render

        screen.blit(background\_image, (0, 0))  # Draw background image

        # Draw game elements

        player\_paddle.draw()

        ai\_paddle.draw()

        puck.draw()

        # Draw scores with light gray color

        player\_score\_text = font.render("Player: " + str(player\_score), True, BLACK)

        ai\_score\_text = font.render("Computer: " + str(ai\_score), True, BLACK)

        screen.blit(player\_score\_text, (20, 20))

        screen.blit(ai\_score\_text, (WIDTH - 170, 20))

        # Draw boundary lines

        pygame.draw.line(screen, WHITE, (0, 0), (WIDTH, 0), 2)  # Top boundary line

        pygame.draw.line(screen, WHITE, (0, HEIGHT), (WIDTH, HEIGHT), 2)  # Bottom boundary line

        pygame.display.flip()  # Update the display

        # Cap the frame rate

        pygame.time.Clock().tick(60)

def game\_over(player\_score, ai\_score):

    # Load background image for game over screen

    game\_over\_background = pygame.image.load("3.jpg")  # Replace "3.jpg" with your image file

    game\_over\_background = pygame.transform.scale(game\_over\_background, (WIDTH, HEIGHT))  # Scale the image to fit the screen

    # Display game over screen with a background image

    screen.blit(game\_over\_background, (0, 0))  # Blit the background image onto the screen

    font = pygame.font.SysFont(None, 48)

    game\_over\_text = font.render("Game Over", True, WHITE)

    player\_score\_text = font.render("Player Score: " + str(player\_score), True, WHITE)

    ai\_score\_text = font.render("Computer Score: " + str(ai\_score), True, WHITE)

    # Calculate text positions

    game\_over\_text\_rect = game\_over\_text.get\_rect(center=(WIDTH // 2, HEIGHT // 2 - 50))

    player\_score\_text\_rect = player\_score\_text.get\_rect(center=(WIDTH // 2, HEIGHT // 2))

    ai\_score\_text\_rect = ai\_score\_text.get\_rect(center=(WIDTH // 2, HEIGHT // 2 + 50))

    # Blit text onto the screen

    screen.blit(game\_over\_text, game\_over\_text\_rect)

    screen.blit(player\_score\_text, player\_score\_text\_rect)

    screen.blit(ai\_score\_text, ai\_score\_text\_rect)

    pygame.display.flip()

    pygame.time.delay(3000)  # Delay for 3 seconds to allow players to see the final scores

if \_\_name\_\_ == "\_\_main\_\_":

    main()

**TWO-PLAYER GAME**

import pygame

import sys

import random

# Initialize Pygame

pygame.init()

# Set up the window

WIDTH, HEIGHT = 800, 600

screen = pygame.display.set\_mode((WIDTH, HEIGHT))

pygame.display.set\_caption("Air Hockey")

# Load background image

background\_image = pygame.image.load("2.jpg")  # Replace "2.jpg" with your image file

# Define colors

WHITE = (255, 255, 255)

RED = (255, 0, 0)

BLUE = (0, 0, 255)

GRAY = (200, 200, 200)  # Light gray color for scoreboard

BLACK = (0, 0, 0)  # Define the BLACK color constant

# Define game elements

class Paddle:

    def \_\_init\_\_(self, x, y, color):

        self.rect = pygame.Rect(x, y, 20, 100)

        self.color = color

        self.speed = 10  # Increased speed

        self.direction = 0  # 0 for not moving, -1 for up, 1 for down

    def update(self):

        self.rect.y += self.direction \* self.speed

        if self.rect.top < 0:

            self.rect.top = 0

        elif self.rect.bottom > HEIGHT:

            self.rect.bottom = HEIGHT

    def draw(self):

        pygame.draw.rect(screen, self.color, self.rect)

class Puck:

    def \_\_init\_\_(self, x, y):

        self.rect = pygame.Rect(x, y, 20, 20)

        self.color = BLACK  # Initial color of the puck

        self.speed\_x = 7

        self.speed\_y = 7

        self.glow\_rate = 5  # Rate of color transition

        self.glowing\_up = True  # Flag to indicate if glowing is increasing or decreasing

        self.collision\_point = None  # Point of collision with paddle

    def update\_color(self):

        if self.glowing\_up:

            self.color = (min(self.color[0] + self.glow\_rate, 255),

                          min(self.color[1] + self.glow\_rate, 255),

                          min(self.color[2] + self.glow\_rate, 255))

            if self.color == (255, 255, 255):  # If reached maximum brightness, start decreasing

                self.glowing\_up = False

        else:

            self.color = (max(self.color[0] - self.glow\_rate, 0),

                          max(self.color[1] - self.glow\_rate, 0),

                          max(self.color[2] - self.glow\_rate, 0))

            if self.color == (0, 0, 0):  # If reached minimum brightness, start increasing

                self.glowing\_up = True

    def update(self):

        self.rect.x += self.speed\_x

        self.rect.y += self.speed\_y

        # Check for collisions with top and bottom walls

        if self.rect.top <= 0 or self.rect.bottom >= HEIGHT:

            self.speed\_y = -self.speed\_y

    def draw(self):

        pygame.draw.ellipse(screen, self.color, self.rect)

        if self.collision\_point:

            pygame.draw.line(screen, WHITE, self.rect.center, self.collision\_point, 2)

# Define game states

class GameState:

    START\_SCREEN = 0

    PLAYING = 1

    GAME\_OVER = 2

# Define the SCORE\_LIMIT constant

SCORE\_LIMIT = 5  # Adjust this value as needed

# Main game loop

def main():

    game\_state = GameState.START\_SCREEN

    player1\_paddle = Paddle(50, HEIGHT // 2 - 50, RED)

    player2\_paddle = Paddle(WIDTH - 70, HEIGHT // 2 - 50, BLUE)

    puck = Puck(WIDTH // 2 - 10, HEIGHT // 2 - 10)

    player1\_score = 0

    player2\_score = 0

    font = pygame.font.SysFont(None, 36)

    while True:

        for event in pygame.event.get():

            if event.type == pygame.QUIT:

                pygame.quit()

                sys.exit()

            elif event.type == pygame.KEYDOWN:

                if game\_state == GameState.START\_SCREEN:

                    game\_state = GameState.PLAYING

                elif game\_state == GameState.PLAYING:

                    # Player 1 controls

                    if event.key == pygame.K\_w:

                        player1\_paddle.direction = -1

                    elif event.key == pygame.K\_s:

                        player1\_paddle.direction = 1

                    # Player 2 controls

                    elif event.key == pygame.K\_UP:

                        player2\_paddle.direction = -1

                    elif event.key == pygame.K\_DOWN:

                        player2\_paddle.direction = 1

            elif event.type == pygame.KEYUP:

                if game\_state == GameState.PLAYING:

                    # Player 1 controls

                    if event.key == pygame.K\_w or event.key == pygame.K\_s:

                        player1\_paddle.direction = 0

                    # Player 2 controls

                    elif event.key == pygame.K\_UP or event.key == pygame.K\_DOWN:

                        player2\_paddle.direction = 0

        if game\_state == GameState.PLAYING:

            # Update game state

            player1\_paddle.update()

            player2\_paddle.update()

            puck.update()

            # Check for collisions with paddles

            if player1\_paddle.rect.colliderect(puck.rect) or player2\_paddle.rect.colliderect(puck.rect):

                puck.speed\_x = -puck.speed\_x

            # Check for goals

            if puck.rect.left <= 0:

                player2\_score += 1

                reset\_game(player1\_paddle, player2\_paddle, puck)

            elif puck.rect.right >= WIDTH:

                player1\_score += 1

                reset\_game(player1\_paddle, player2\_paddle, puck)

            # Check for game over condition

            if player1\_score >= SCORE\_LIMIT or player2\_score >= SCORE\_LIMIT:

                game\_state = GameState.GAME\_OVER

        # Render

        screen.blit(background\_image, (0, 0))  # Draw background image

        # Draw game elements

        player1\_paddle.draw()

        player2\_paddle.draw()

        puck.draw()

        # Draw scores with light gray color

        player1\_score\_text = font.render("Player 1: " + str(player1\_score), True, BLACK)

        player2\_score\_text = font.render("Player 2: " + str(player2\_score), True, BLACK)

        screen.blit(player1\_score\_text, (20, 20))

        screen.blit(player2\_score\_text, (WIDTH - 170, 20))

        pygame.display.flip()  # Update the display

        # Cap the frame rate

        pygame.time.Clock().tick(60)

        # Check for game over condition and reset the game if necessary

        if game\_state == GameState.GAME\_OVER:

            game\_over(player1\_score, player2\_score)

            player1\_score = 0

            player2\_score = 0

            reset\_game(player1\_paddle, player2\_paddle, puck)

def reset\_game(player1\_paddle, player2\_paddle, puck):

    puck.rect.center = (WIDTH // 2, HEIGHT // 2)

    player1\_paddle.rect.center = (50, HEIGHT // 2)

    player2\_paddle.rect.center = (WIDTH - 70, HEIGHT // 2)

    puck.speed\_x = 7 if random.randint(0, 1) == 0 else -7

    puck.speed\_y = random.choice([-7, 7])

def game\_over(player1\_score, player2\_score):

    # Load background image for game over screen

    game\_over\_background = pygame.image.load("3.jpg")  # Replace "3.jpg" with your image file

    game\_over\_background = pygame.transform.scale(game\_over\_background, (WIDTH, HEIGHT))  # Scale the image to fit the screen

    # Display game over screen with a background image

    screen.blit(game\_over\_background, (0, 0))  # Blit the background image onto the screen

    font = pygame.font.SysFont(None, 48)

    winner\_text = font.render("Player 1 Wins!" if player1\_score > player2\_score else "Player 2 Wins!", True, WHITE)

    player1\_score\_text = font.render("Player 1 Score: " + str(player1\_score), True, WHITE)

    player2\_score\_text = font.render("Player 2 Score: " + str(player2\_score), True, WHITE)

    # Calculate text positions

    winner\_text\_rect = winner\_text.get\_rect(center=(WIDTH // 2, HEIGHT // 2 - 50))

    player1\_score\_text\_rect = player1\_score\_text.get\_rect(center=(WIDTH // 2, HEIGHT // 2))

    player2\_score\_text\_rect = player2\_score\_text.get\_rect(center=(WIDTH // 2, HEIGHT // 2 + 50))

    # Blit text onto the screen

    screen.blit(winner\_text, winner\_text\_rect)

    screen.blit(player1\_score\_text, player1\_score\_text\_rect)

    screen.blit(player2\_score\_text, player2\_score\_text\_rect)

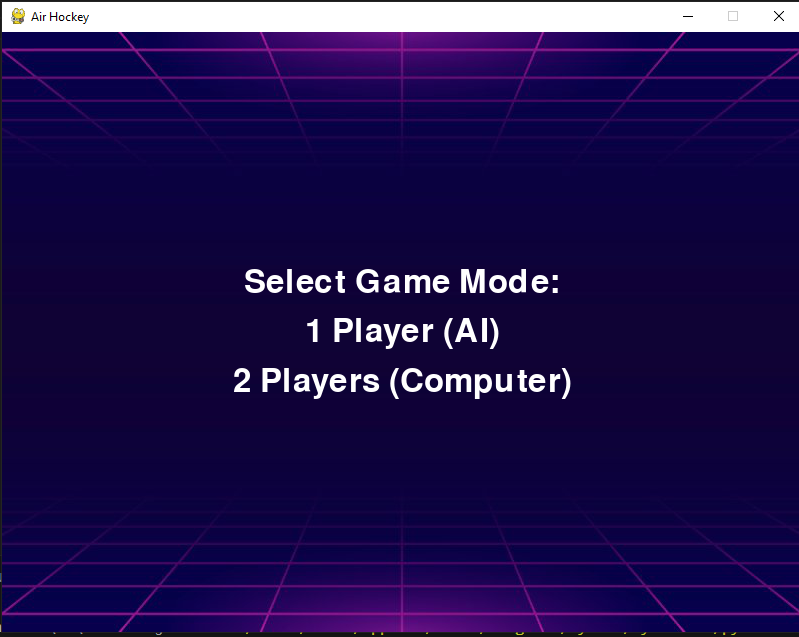
    pygame.display.flip()

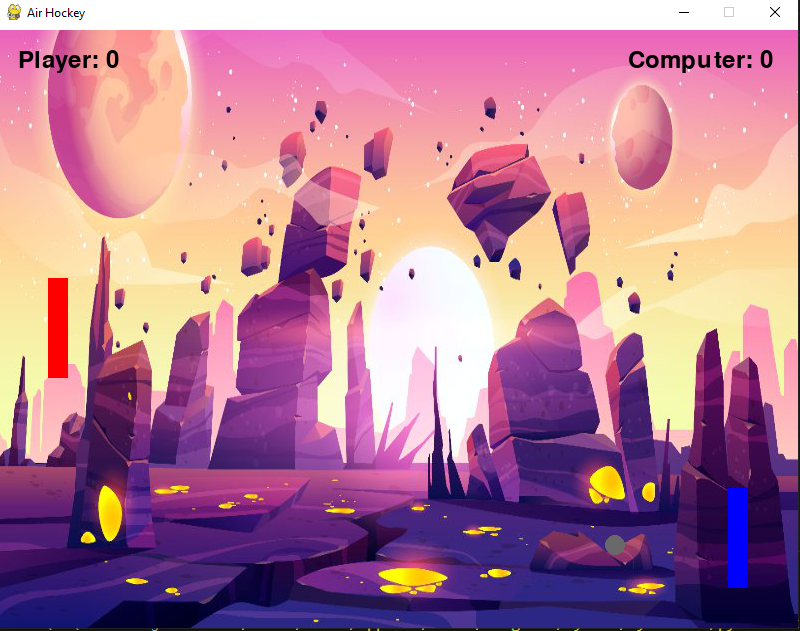
    pygame.time.delay(3000)  # Delay for 3 seconds to allow players to see the final scores

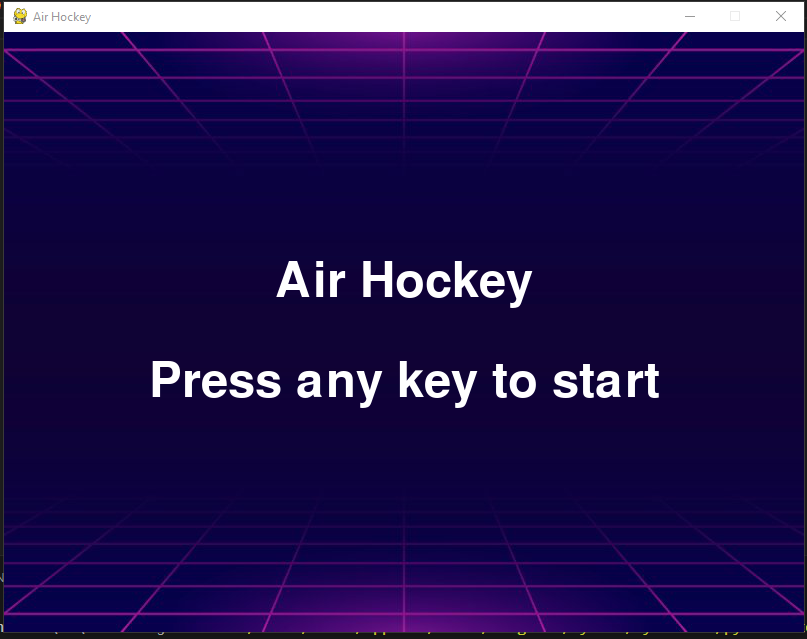
if \_\_name\_\_ == "\_\_main\_\_":

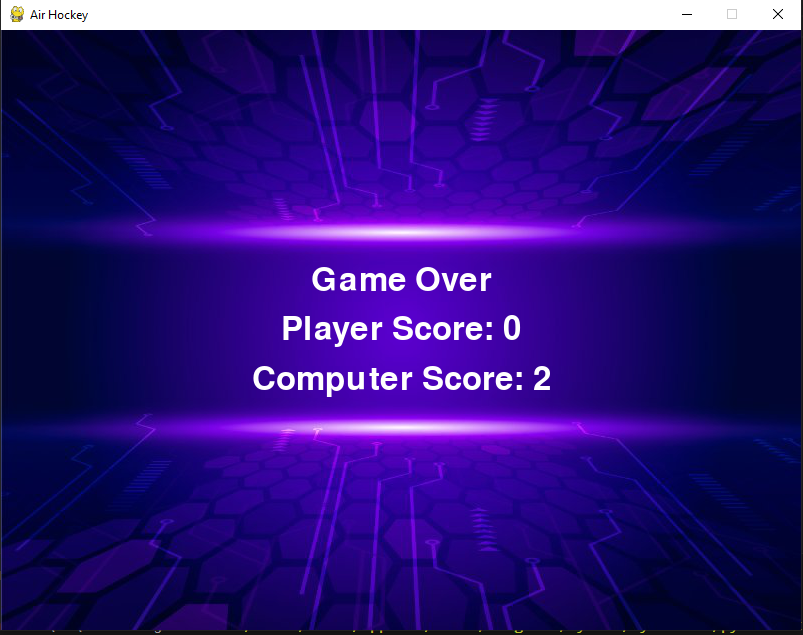
    main()

**Output:**









* **Conclusion:**

In conclusion, the Air Hockey game offers an exciting and immersive experience for players of all skill levels. With its intuitive controls, dynamic gameplay, and engaging visuals, the game provides hours of entertainment and challenges players to showcase their reflexes and strategic prowess.

Throughout the game, players are presented with various feedback mechanisms that enhance their understanding of the game mechanics, guide their progress, and celebrate their achievements. Visual and auditory cues highlight key events such as puck collisions and goals scored, while clear score updates keep players informed about their performance. Error messages offer constructive feedback to help players learn from their mistakes and improve their skills over time.

The inclusion of an AI opponent adds an additional layer of challenge and excitement, as players must adapt their strategies to outsmart their virtual adversary. The AI algorithm employed in the game utilizes heuristic functions to simulate intelligent paddle movements, creating a dynamic and responsive opponent that keeps players on their toes.

With its balanced gameplay, adaptive difficulty, and comprehensive feedback system, the Air Hockey game offers a rewarding and enjoyable experience for players of all ages. Whether competing against friends, challenging the computer, or honing their skills in practice mode, players are sure to find endless fun in the fast-paced world of Air Hockey.