# **BTP5-500**

#### **BTP500 Project 2: Building a Game in Python Using Data Structures**

# **Jung ho Lee FROM KOREA**

**Michael K. Melles 111838223**

**Samin ----**

**Seneca Polytechnic College**

**Rani Gnanaolivu**

**April 10, 2025**

# **Game Documentation – *Advanced Rock-Paper-Scissors***

# **Technical Implementation & Algorithms – *Advanced Rock-Paper-Scissors***

This section outlines the core logic behind our game project, highlighting the data structures, algorithms, and object-oriented programming techniques used to build an interactive and intelligent Rock-Paper-Scissors game in Python with Pygame.

## **🎮 Game Design and Structure**

Our game is built using **Python 3** and the **Pygame** library. It provides two main game modes: Single Player (vs AI) and Tournament Mode (2+ players). The design follows a **state-based architecture** and uses object-oriented programming to separate gameplay logic from rendering and user interaction.

We used classes to represent the game (Game), each player (Player), and the tournament system (Tournament). The game loop handles user input, state transitions, and screen rendering.

## **🔁 Game Loop & State Management**

The game uses a **finite state machine** to manage screen transitions. Different screens are drawn depending on the value of game\_state, which can be:

MENU = 0  
PLAYING = 1  
TOURNAMENT = 3

Each screen (menu, gameplay, winner screen) has its own function like draw\_menu() or draw\_game(). The main loop checks the current state and calls the corresponding draw function.

if game\_state == MENU:  
 action = draw\_menu()  
elif game\_state == PLAYING:  
 action = draw\_game()

State transitions happen when the player clicks buttons or finishes a match.

## **Data Structures Used**

### **deque – Recent Move Tracking**

We used Python’s collections.deque with a maxlen of 10 to store the last 10 moves made by a player. This allows the AI to observe and react to recent patterns in a human player's behavior.

self.move\_history = deque(maxlen=10)

Using deque is efficient for this sliding window behavior, with fast appends and automatic trimming.

### **deque – Tournament Queue**

The tournament system uses two deques:

* self.players holds players waiting to play
* self.winners stores round winners

self.players = deque()  
self.winners = deque()

Players are dequeued in pairs. Winners are stored and used to generate the next round until a final champion is found.

### **list of dicts – Game History**

We store every completed match in a dictionary, logging players, moves, and results. These dictionaries are stored in a list.

self.game\_history.append({  
 'player1': self.player1.name,  
 'player2': self.player2.name,  
 'move1': move1,  
 'move2': move2,  
 'result': self.result  
})

The last 5 matches are shown after the tournament ends.

## **🤖 AI Algorithms (3 Levels)**

We implemented three levels of AI difficulty. Each level is progressively smarter, using increasingly advanced prediction techniques.

### **🔹 Level 1 – Random AI**

The easiest level picks a move at random. It’s unpredictable but not strategic.

python

CopyEdit

return random.choice(['rock', 'paper', 'scissors'])

This level requires no use of player history.

### **🔹 Level 2 – Pattern Counter AI**

This AI checks if the opponent has used the same move multiple times. If a move is repeated, the AI plays its counter.

python

CopyEdit

last\_move = opponent\_history[-1]  
if opponent\_history.count(last\_move) > 2:  
 return self.get\_counter(last\_move)

If no repetition is found, it makes a random choice.

### **🔹 Level 3 – Predictive AI**

This level uses a basic **Markov Chain**-like prediction. It looks at the last 3 moves of the opponent and searches for similar patterns in their history. If it finds a match, it predicts the next move and plays its counter.

pattern = tuple(opponent\_history[-3:])  
for i in range(len(opponent\_history)-3):  
 if tuple(opponent\_history[i:i+3]) == pattern:  
 possible\_next.append(opponent\_history[i+3])

This level uses short-term memory to make better predictions based on past behavior.

## **📦 Object-Oriented Structure**

We used **classes** to organize the logic and responsibilities of the game.

### **🧍 Player Class**

Each player has a name, a score, a move history, and optional AI settings. The AI logic is contained within the make\_move() method.

class Player:  
 def \_\_init\_\_(self, name, is\_ai=False, ai\_level=1):  
 self.name = name  
 self.is\_ai = is\_ai  
 self.ai\_level = ai\_level  
 self.move\_history = deque(maxlen=10)

If a player is AI, the game calls make\_move(opponent\_history) to get its move.

### **🎮 Game Class**

This is the core logic manager. It handles:

* Starting single/tournament matches
* Processing player moves
* Determining match results
* Storing match history
* Handling round animations

def determine\_result(self):  
 if move1 == move2:  
 self.result = 'draw'  
 elif (move1 == 'rock' and move2 == 'scissors') ...

It also plays sound effects and handles the "next round" or "next match" logic.

### **🏆 Tournament Class**

The tournament manager organizes player matchups and tracks progress through each round. It selects two players at a time and queues up winners.

def next\_match(self):  
 self.current\_match = (self.players.popleft(), self.players.popleft())

When only one player remains, that player is the tournament winner.

## **🖱️ UI Interaction & Input**

### **Button System**

We use a reusable draw\_button() function to handle drawing and interaction. It detects hover and click states and changes button color accordingly.

if x <= mouse\_pos[0] <= x + width and y <= mouse\_pos[1] <= y + height:  
 if clicked:  
 color = click\_color

### **Image Selection**

Players select rock, paper, or scissors by clicking image icons. Mouse position is checked against image rectangles:

if img\_rect.collidepoint(mouse\_pos):  
 if pygame.mouse.get\_pressed()[0]:  
 game.make\_move(1, move)

This makes gameplay interactive and intuitive.

## **🌀 Visual Animation**

Animations make the game feel more dynamic. When both players choose their moves, we animate the icons sliding from the sides toward the center.

progress = game.animation\_frame / game.max\_animation\_frames  
x = 150 \* progress

A “VS” text fades in during the animation, building suspense before showing the result.

## **Text Input Handling**

We support player name entry using basic keyboard input tracking:

if input\_active and event.type == pygame.KEYDOWN:  
 player\_name += event.unicode

For tournaments, users enter names separated by commas. These are parsed into a list using. split(",").

## **⚠️ Error Handling & Fallbacks**

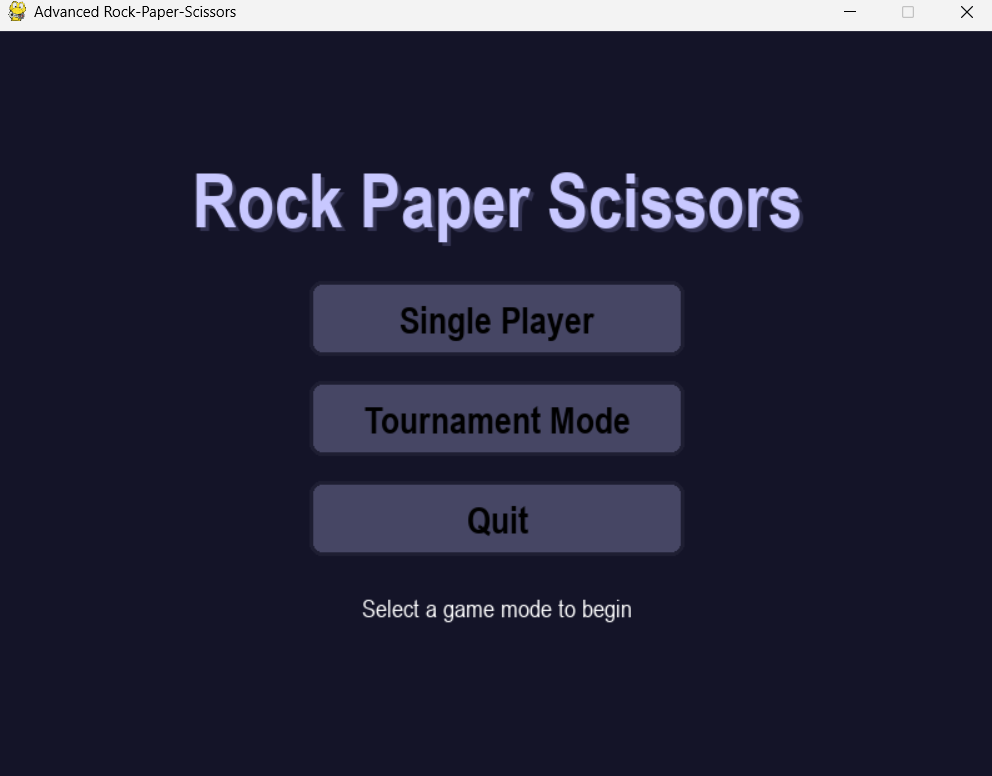
If assets like images or sounds are missing, we create visual or audio fallbacks.

For example, if rock.png is missing, we draw a circle instead:

pygame.draw.circle(surf, (0, 0, 255), (50, 50), 45)

### **Single Player t Mode Gameplay Walkthrough with Player Interaction Flow**

### **Main Menu**



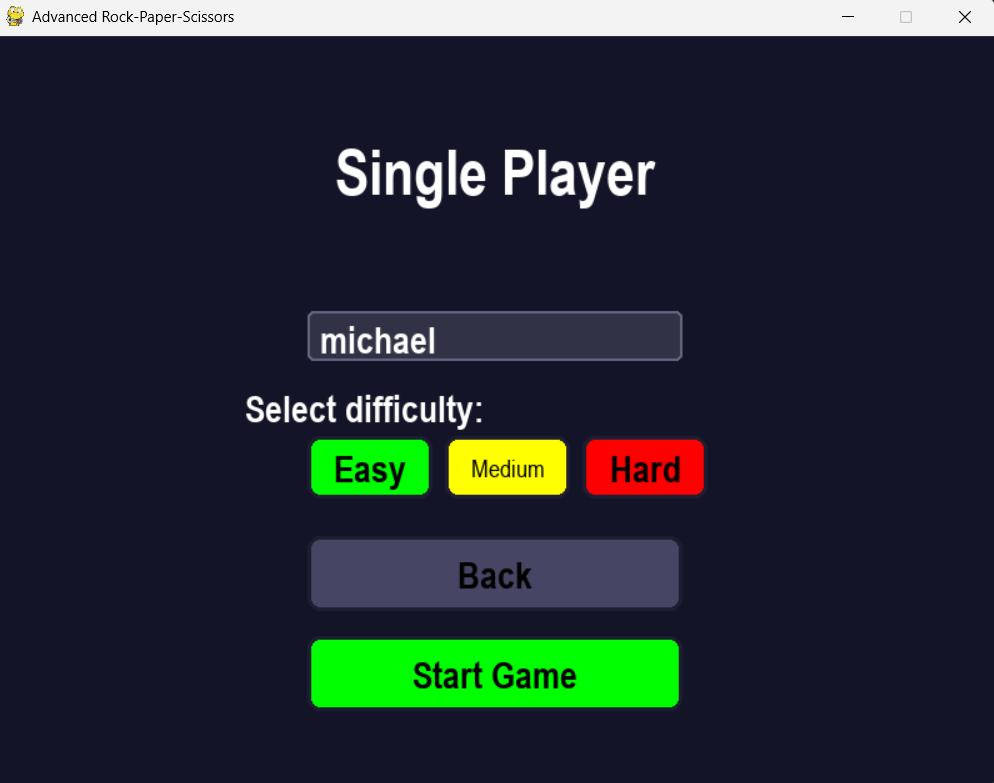
**Description:**

This is the **main menu screen** where players begin the game. It features three buttons:

* **Single Player** to play against the AI.
* **Tournament Mode** to play with multiple human or AI players.
* **Quit** to exit the game.

The clean design and large buttons make it easy for the player to start the game by selecting a mode.

### **Single Player Setup**



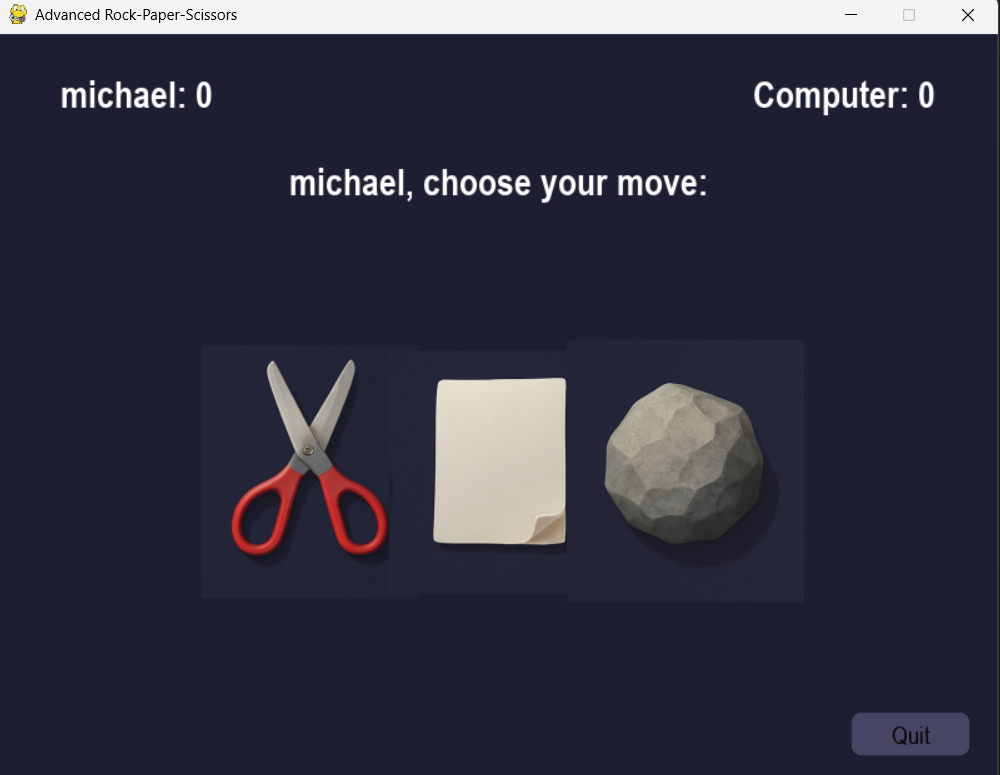
**Description:**

After selecting Single Player, the user is prompted to enter their **name** and choose a **difficulty level**:

* **Easy** (Green) uses a random AI.
* **Medium** (Yellow) uses pattern detection.
* **Hard** (Red) uses advanced predictive logic.

Once a difficulty is chosen, clicking **Start Game** begins the match. The **Back** button returns to the main menu.

### **Move Selection Screen**

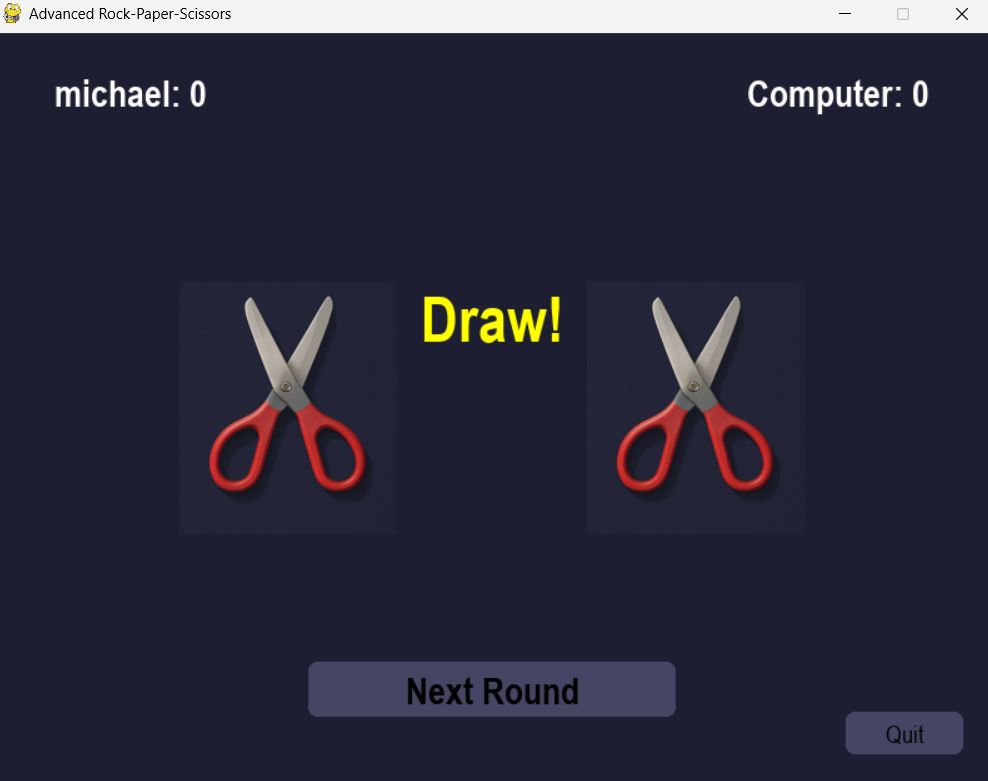


**Description:**

This screen shows the main gameplay for a single round. The player, **Michael**, is prompted to **choose a move**: scissors, paper, or rock.

The current **score** is displayed at the top (Michael vs. Computer), and the player selects a move by clicking the respective image. Once both moves are made, the result is shown on a new screen.

### **Draw Result**

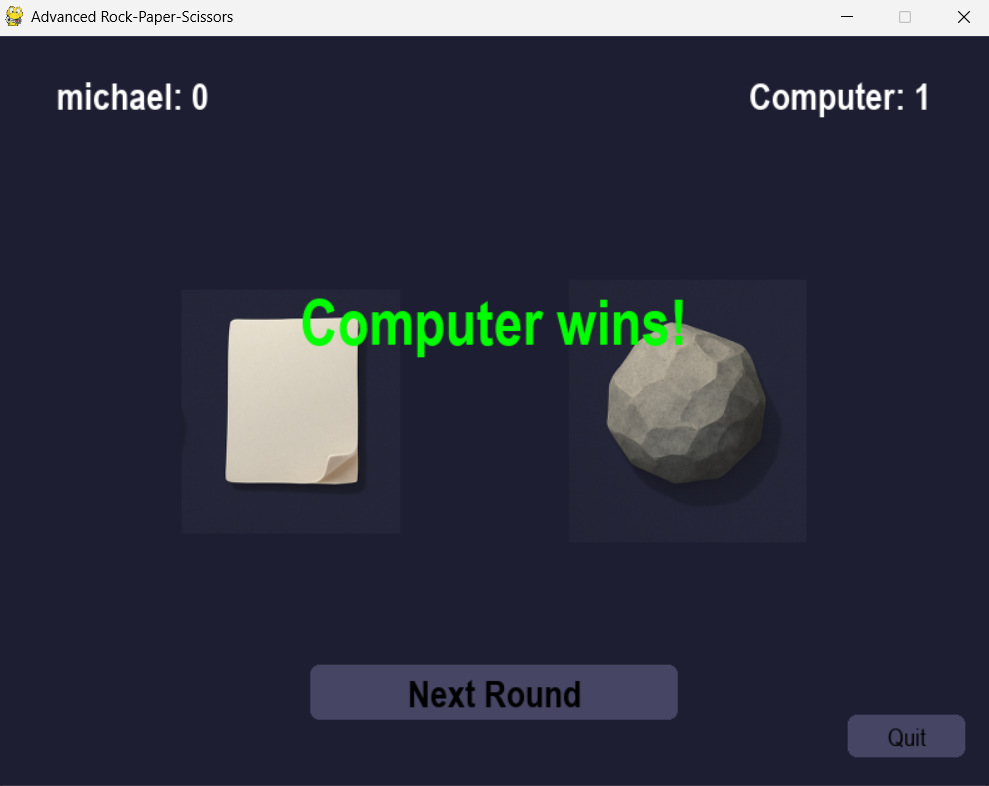


**Description:**

Both Michael and the computer chose the same move (scissors), resulting in a **draw**.

The game displays both selected icons and the word **"Draw!"** in bold yellow text. The player can click **Next Round** to continue or **Quit** to return to the menu.

### **Win/Loss Result**



**Description:**

Michael chose **paper**, and the computer chose **rock**, resulting in a **win for the computer**. The screen shows the selected moves and displays **"Computer wins!"** in green. The scores update accordingly, and the **Next Round** button allows the player to play again immediately.

**Tournament Mode Gameplay Walkthrough with Player Interaction Flow**

**Description:**

In Tournament Mode, players are prompted to enter names separated by commas. This screen supports both human and AI players. For example, entering samin,jung\_ho sets up a 1v1 match between two human players.

After entering names, clicking **Start Tournament** initiates the first round.

### **Player 1 Move Selection**

**Description:**

This screen shows the move selection phase for **Player 1 (samin)**. The screen presents three options — **scissors, paper, and rock** — shown twice vertically for each player.

At this stage:

* Only **samin** is supposed to interact with the game.
* After samin selects a move, their choice is **hidden**.
* The screen is then **handed over to jung\_ho** to maintain secrecy.

This manual passing of the device ensures fair play and prevents the second player from seeing the first player's move.

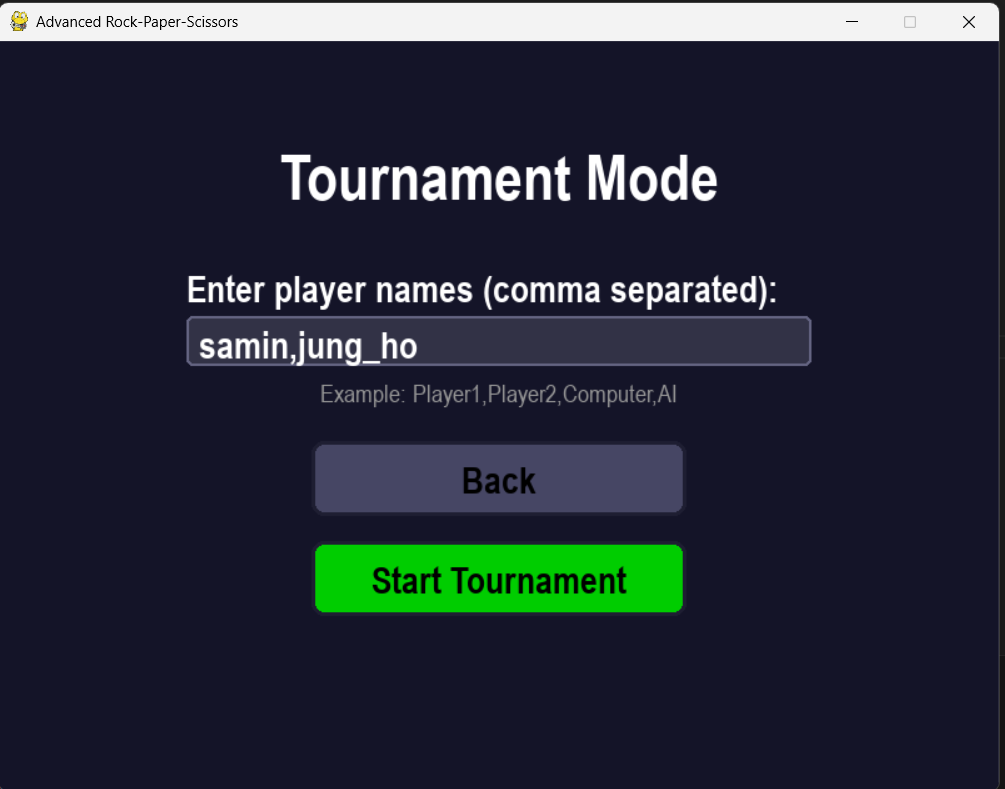
### **Match Result Screen**

**Description:**

After both players have selected their moves, the result is displayed. In this example, both samin and jung\_ho chose **scissors**, resulting in a **draw**.

The game presents the chosen objects clearly and displays **“Draw!”** in bold yellow text. Players then click **Next Match** to move on, or **Quit** to return to the main menu.

### **Tournament Setup Screen**

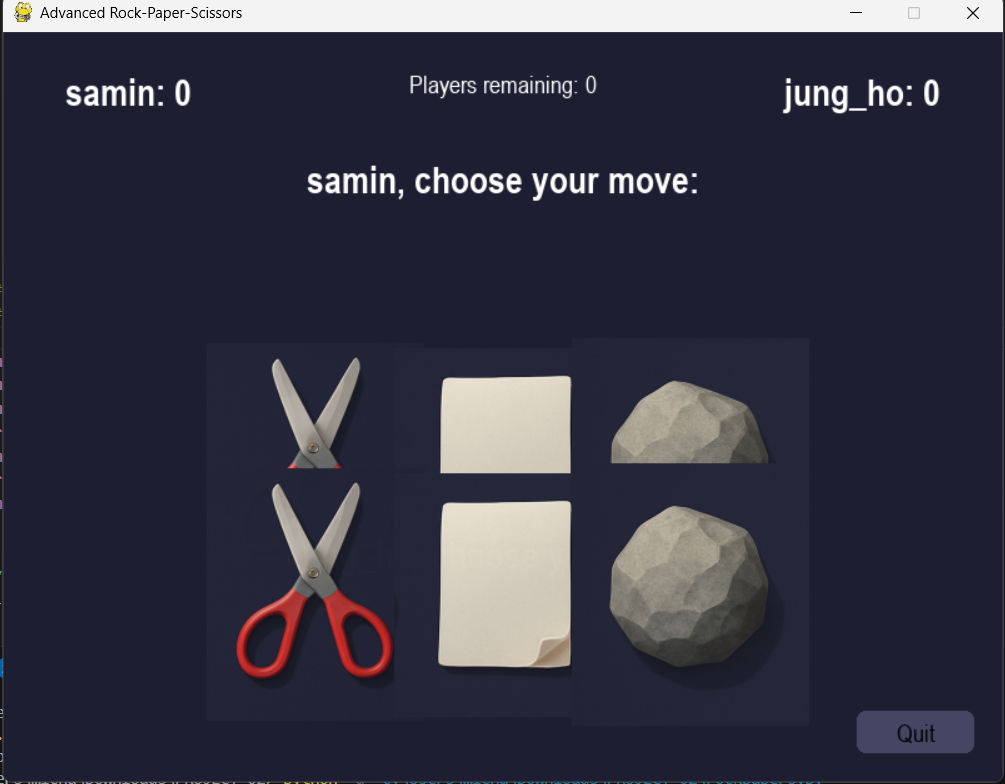


**Description:**

In Tournament Mode, players start by entering their names in the input field, separated by commas. For example, typing samin,jung\_ho sets up a two-player head-to-head match. Players can also include "Computer" or "AI" as one of the names to play against an AI opponent.

Once the names are entered, pressing the **Start Tournament** button begins the bracket and loads the first matchup.

### **Player 1 Move Selection (Private Turn)**

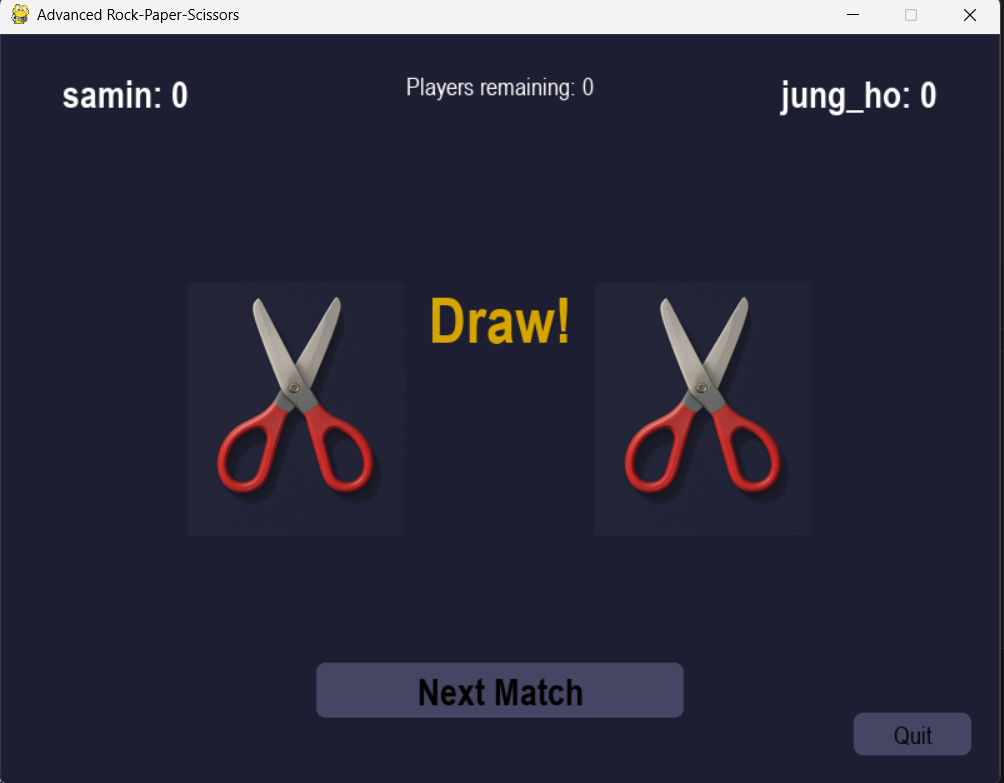


**Description:**

During a tournament round, each player takes their turn **privately** to select a move. In this screen, **samin** (Player 1) is prompted to choose between scissors, paper, or rock. The UI shows two rows of selectable objects to visually divide the player turns.

After samin selects a move, the game **hides their choice** and passes control to **jung\_ho** (Player 2). This ensures that Player 2 does not see the first player's move, keeping the game fair and competitive.

### **Result Reveal Screen**



**Description:**

After both players have selected their moves, the result is revealed. The screen displays each player’s choice (in this case, both selected scissors) along with the round outcome. In the example shown, the match ends in a **Draw**, highlighted in bold yellow.

Scores are shown at the top, and the number of players remaining in the tournament is also indicated. Pressing **Next Match** advances the tournament to the next round.

**Project Summary – *Advanced Rock-Paper-Scissors***

* **Project Type:** Python-based game using the Pygame library
* **Goal:** Reimagine the classic Rock-Paper-Scissors game with intelligent AI, animated visuals, and multiplayer functionality

### **🔹 Key Features:**

* **Two Game Modes:**
  + **Single Player** vs AI (3 difficulty levels)
  + **Tournament Mode** with 2+ players (human and/or AI)
* **AI System:**
  + Easy (Random), Medium (Pattern Detection), Hard (Move Prediction)
* **Interactive UI:**
  + Clickable buttons and image-based move selection
  + Responsive interface with visual feedback
* **Match Results:**
  + Displays both player moves, outcomes, and scores
  + Draw, win, and loss clearly animated with sound effects
* **Tournament Logic:**
  + Bracket-style elimination with queue-based matchups
  + Supports private, turn-based selection to ensure fairness

### **🧰 Technologies Used:**

* **Language:** Python 3
* **Libraries:** Pygame, collections (deque), random
* **Design Approach:** Object-Oriented Programming (OOP) with custom Player, Game, and Tournament classes

This project demonstrates our understanding of game logic, user interface design, AI behavior, and interactive programming using Python. It is fun, competitive, and structured to support both solo and group play.