My Take on "Set" Card Game

1. Introduction:

   This project was undertaken as part of the Systems Programming course at Ben Gurion University. Its primary aim was to familiarize us with concurrent programming concepts. My enthusiasm for the assignment led me to re-implement it a few months later, resulting in a faster, more efficient, and modular version.

   In one sentence: Concurrent implementation of the card game "Set", Where CPU-controlled players compete against each other in real-time. The implementation also supports human players.

2. Objectives:

   a. **Thread Safety and Correctness –** Emphasizing critical sections and addressing edge cases to avoid concurrency issues.

   b. **CPU Optimization –** Employing thread synchronization techniques, including Readers-Writers locks, to enhance CPU resource utilization.

   c. Object-Oriented Programming – Utilizing inheritance to create abstract player classes, such as Human and Bot players, along with essential entities like the Table, Dealer, and Timer.

3. "Set" Card Game Overview:

   The "Set" card game involves a dealer distributing cards onto the table, and players aiming to assemble valid sets by placing tokens strategically. Each card possesses four attributes: *shape, color, number, and shading*. A valid set comprises three cards that satisfy conditions like having either the same or different attributes for each of the four characteristics. A dedicated function which determines the validity of a set has been given to us by the course staff. The dealer evaluates players' sets, retrieves their tokens, and imposes cooldowns based on set validity. A timer operates in the background, resetting the table upon reaching zero.

4. Project Components:

   The project consists of several integral components:

   a. **The Table:** Functioning as a singleton-like repository, housing shared resources such as current table cards and players' tokens.

   b. **The Player:** An abstract class serving as the foundation for both **Human** and **Bot** players. The next card selection mechanism is implemented within the inheriting classes.

   c. **The Dealer:** Responsible for table resets and set evaluations.

   d. **The Timer:** An additional thread contributing to countdown functionality.

5. Challenges and Resolutions:

   a. Managing Shared Resources – The dynamic nature of the game required meticulous handling of shared resources, accomplished by implementing a Readers-Writers lock for cards and tokens. Prioritization was assigned to the dealer for resource access, putting the Dealer in the Writer position and the players in the Readers position.

   Wrapping critical sections in a lock –

   Dealer: 1) Resetting table (letsPlay()). 2) Checking player's set (checkSets()).

   Player: 1) Laying a token on the table (executePress()).

   b. **Real-time notifications** – Effective inter-entity communication was established through interruptions, optimizing CPU utilization during thread waits.

   c. **Dealer instructs players** – A "state" data member facilitated real-time updates from the dealer to players, influencing their actions based on their respective states: *waiting, playing, point, penalty*.

6. Entity Roles – Each entity plays a crucial role in the system:

Dealer	Player	Table	Timer
Holds the card deck, manages set evaluation queues, and oversees critical functions such as table resets and set checks.	Manages individual scores, token count, and the selection of next tokens. Differentiated between Human and Bot players.	Stores cards and tokens, serving as a shared repository for active game elements.	Executes countdown operations, prompting table resets upon reaching zero.