BLACKJACK

Juego en C++

Nombre: Luis Sinning – T00078764

Profesor: Oscar Porto

¿Qué es?

El blackjack, también llamado veintiuno, es un juego de cartas, propio de los casinos con una o más barajas inglesas de 52 cartas sin los comodines, que consiste en sumar un valor lo más próximo a 21 pero sin pasarse. En un casino cada jugador de la mesa juega únicamente contra el crupier, intentando conseguir una mejor jugada que este. El crupier está sujeto a reglas fijas que le impiden tomar decisiones sobre el juego. Por ejemplo, está obligado a pedir carta siempre que su puntuación sume 16 o menos, y obligado a plantarse si suma 17 o más. Las cartas numéricas suman su valor, las figuras suman 10 y el as vale 11 o 1, a elección del jugador. En el caso del crupier, los ases valen 11 mientras no se pase de 21, y 1 en caso contrario. La mejor jugada es conseguir 21 con solo dos cartas, esto es con un As más carta de valor 10. Esta jugada se conoce como Blackjack o 21 natural. Un blackjack gana sobre un 21 conseguido con más de dos cartas.

Explicación de Código

El código se organizará de la siguiente manera:

Clase Card: Representa una carta con un valor y un palo.

Clase Deck: Representa un mazo de cartas.

Clase Player: Representa a un jugador.

Clase Game: Controla la lógica del juego, las interacciones entre los jugadores y el mazo.

Clase Card

- Representa una carta con tres atributos:
  - suit: El palo de la carta (corazones, diamantes, tréboles, espadas).
  - rank: El valor visible de la carta (2, 3, J, Q, etc.).
  - value: El valor numérico utilizado para calcular el puntaje de la mano en el juego de Blackjack (10 para J, Q, K, y 11 para A).

Atributos

- **string suit**: Representa el **palo** de la carta, que puede ser "Corazones", "Diamantes", "Tréboles" o "Espadas". Es de tipo string porque es una palabra.
- **string rank**: Representa el **valor visible** de la carta, como "2", "3", "J", "Q", "K", "A", etc.
- **int value**: Representa el **valor numérico** de la carta para fines de puntuación en Blackjack. Por ejemplo:
  - Los números del 2 al 10 mantienen su valor.
  - J, Q, K valen 10.
  - El As (A) tiene un valor de 11, aunque puede ser ajustado si es necesario durante el juego.

Métodos

- **Card(string s, string r, int v)**: Constructor que inicializa una carta con su palo, valor visible y valor numérico.

Clase Deck

- Contiene un **vector de cartas** que representa el mazo.
- El constructor crea un mazo con 52 cartas.
- La función shuffle mezcla las cartas aleatoriamente.
- La función dealCard reparte una carta (elimina la carta del mazo).

- La función remainingCards devuelve cuántas cartas quedan en el mazo.

Atributos

- **vector<Card> deck**: Es un vector que almacena todas las cartas del mazo. El tipo Card indica que cada elemento del vector es un objeto de la clase Card.

Métodos

- **Deck ()**: Constructor que **crea el mazo de cartas**. El mazo tiene 52 cartas (13 cartas para cada uno de los 4 palos). Este método llena el vector deck con las cartas.
- En este bloque, se iteran los palos (suits) y los valores (ranks y values), y se crean cartas con esas combinaciones. Por ejemplo, el primer ciclo crea cartas de **"Corazones"**, luego de **"Diamantes"**, y así sucesivamente.
- **void shuffle ()**: **Baraja el mazo** de cartas. Utiliza la función rand() para generar índices aleatorios y reordenar las cartas en el vector.
- cpp
- Copiar código
- El randIndex es el índice aleatorio que se usa para intercambiar cartas en el mazo, barajándolas aleatoriamente.
- Card **dealCard ()**: **Reparte una carta** (elimina una carta del mazo). Esta función devuelve el último elemento del vector deck (la carta más alta del mazo) y luego la elimina con pop_back(), lo que reduce el tamaño del mazo.

int remainingCards (): **Devuelve la cantidad de cartas restantes** en el mazo. Simplemente devuelve el tamaño del vector deck.

Clase Player

- Representa a un jugador.
- Tiene un nombre y un vector de cartas (hand), que son las cartas que el jugador tiene.
- La función addCard agrega una carta a la mano del jugador y actualiza su puntaje.

- La función displayHand muestra las cartas del jugador y su puntaje.
- La función isBusted verifica si el puntaje del jugador supera 21, en cuyo caso se considera "busted".
- La función resetHand limpia la mano del jugador para la próxima ronda.

Atributos

- **string name**: El nombre del jugador.
- **vector<Card> hand**: El **vector de cartas** que contiene las cartas en la mano del jugador.
- **int score**: El puntaje acumulado del jugador. Se actualiza cada vez que se agrega una carta.

Métodos

- **Player (string n)**: Constructor que **inicializa** el nombre del jugador y establece el puntaje en 0.
- **void addCard (Card card)**: **Añade una carta** a la mano del jugador y actualiza su puntaje.
- Este método agrega la carta al vector hand y aumenta el puntaje del jugador con el valor de la carta.
- **void displayHand ()**: Muestra en pantalla las cartas que tiene el jugador y su puntaje.
- Muestra el nombre del jugador, las cartas de su mano y el puntaje total.
- bool **isBusted ()**: Verifica si el jugador se ha "pasado" (es decir, si su puntaje supera 21).
- Devuelve true si el puntaje del jugador es mayor que 21, lo que significa que el jugador ha "perdido" esta ronda.
- void **resetHand ()**: **Reinicia** la mano del jugador, es decir, limpia las cartas y el puntaje.
- Este método es útil para **preparar a los jugadores para una nueva ronda**, limpiando sus cartas y restableciendo el puntaje.

Clase Game

- Controla la lógica del juego.
- En el constructor, se crea un mazo y se piden los nombres de los jugadores.

- La función playRound reparte las cartas y maneja los turnos de los jugadores.
- La función start permite jugar múltiples rondas hasta que el jugador decida terminar o no haya cartas restantes.

Función main

- Pide al usuario cuántos jugadores participarán en el juego y luego inicia el juego.

Atributos

- **Deck deck**: Un objeto de la clase Deck que maneja el mazo de cartas.
- **vector<Player> players**: Un vector que almacena los jugadores del juego.

Métodos

- **Game (int numPlayers)**: Constructor que **inicia el mazo**, lo baraja y crea a los jugadores. Además, solicita los nombres de los jugadores.

 void **playRound ()**: Ejecuta una **ronda de juego**. En esta función:

- Se reparten **dos cartas iniciales** a cada jugador.
- Se muestra la mano de cada jugador.
- Los jugadores toman turnos para decidir si quieren otra carta o plantarse.
- Después de los turnos, se determina el ganador comparando los puntajes de los jugadores que no se han "pasado".

El flujo básico de esta función es:

1. **Repartir cartas** a todos los jugadores.
2. **Mostrar la mano** de cada jugador.
3. Cada jugador puede decidir si quiere más cartas o plantarse.
4. Verificar si algún jugador se ha "pasado" y mostrar el resultado.
5. **Reiniciar** las manos para la próxima ronda.

 void **start ()**: Inicia el juego, permitiendo jugar rondas múltiples hasta que el mazo se quede sin cartas o el jugador decida terminar. En cada ronda se llama a playRound ().

Instrucciones

 Inicio: El juego comienza pidiendo el número de jugadores.
 Reparto: Cada jugador recibe dos cartas. Los jugadores ven sus cartas y su puntaje.
 Turnos de los jugadores: Cada jugador decide si quiere tomar otra carta (por defecto, "n" para no, "s" para sí).
 Condiciones de derrota: Si un jugador supera los 21 puntos, se considera "busted" y pierde su turno.
 Finalización de ronda: Después de que todos los jugadores terminan su turno, el jugador con el puntaje más alto sin haber pasado de 21 gana la ronda.
 Nueva ronda: Después de cada ronda, los jugadores pueden elegir continuar jugando o terminar.

Interacción entre los elementos

- **Mazo (Deck) y Jugadores (Player)**:
  - El mazo reparte cartas a los jugadores mediante deck. dealCard (), y los jugadores reciben las cartas con player. addCard ().
  - El mazo se **baraja** antes de empezar con deck. Shuffle (), asegurando que el juego sea impredecible.
- **Juego (Game) y Jugadores**:
  - El juego gestiona las rondas, los turnos de los jugadores y la determinación del ganador.
  - Los jugadores deciden si quieren más cartas a través de la entrada del usuario (sí/no) durante su turno.
- **Lógica de puntaje y "pasarse"**:

- Los jugadores acumulan puntajes al recibir cartas, y la condición de "busted" se controla con el método isBusted () de la clase Player.