# Blackjack Digitale

Jonathan Figoni
Giacomo Crescenzio
Salvatore Gritto
Pietro Bigoni

24/8/2025

## 1 Tema

Il progetto ha lo scopo di simulare digitalmente il gioco d'azzardo di carte comunemente noto come blackjack. Il gioco si svolge tra il banco, solitamente rappresentato dal casinò e i giocatori. Dopo aver fatto la puntata, a ogni giocatore vengono assegnate due carte, a questo punto ciascuno di essi sceglie se richiedere carte aggiuntive o fermarsi. Dopodichè il banco fa lo stesso, continuando a pescare se e solo se il suo punteggio attuale è inferiore a 17: vincono i giocatori che realizzano un punteggio più alto del banco e non superiore a 21. I punteggi sono calcolati sommando il valore di ciascuna carta nella propria mano: le jack, regina e re valgono 10 mentre ogni altra carta ha il valore del numero a cui corrisponde, con la sola eccezione dell'asso che può valere 11 o 1 a seconda delle necessità.

## 2 Scelte progettuali e implementative

Per la realizzazione di questo progetto abbiamo scelto di creare diverse classi che potessero rappresentare i vari elementi del blackjack, fornite di metodi consoni al loro ruolo nel gioco. Tra esse le più importanti sono Card, Deck e Hand, che rappresentano rispettivamente una singola carta, un mazzo di carte e la mano di un giocatore. Tutte le classi e funzioni sono richiamate da un unico file, il Main, che costituisce lo scheletro del progetto e contiene la vera e propria dinamica del gioco. Per una migliore collaborazione tra i membri del gruppo ci siamo avvalsi dello strumento di Git Hub per mettere in comune i progressi e gli aggiornamenti in corso d'opera. Di seguito una descrizione più dettagliata dei contenuti di ciascuna coppia di file Source e Header.

### 2.1 Classe Card

La classe Card è la base del programma: il suo costruttore prende in input un valore (range), un seme (suit) e un valore di gioco (game_value). Per essa è inoltre definito l'operatore == che ritorna true se sono uguali sia il seme che il rango (e non game_value, poichè se anche due assi in un dato momento hanno valori di gioco diversi le carte devono essere considerate uguali). Inoltre sono presenti i getters per range_, suit_ e game_value, essendo questi ultimi definiti in private e perciò non accessibili direttamente.

### 2.2 Classe Deck

Negli stessi file è presente la classe Deck, che è un altro fulcro del programma. E' implementata come un vettore contenente oggetti di tipo Card e il suo costruttore inizializza direttamente tre mazzi tramite un ciclo for pescando valori e semi da una serie di array. La classe contiene inoltre diversi metodi fondamentali:

- size_t size(): ritorna il numero di carte del mazzo.
- void shuffle(): mescola il mazzo usando std::shuffle() con un generatore di numeri casuali basato su std::random_device.

- Card topCard(): ritorna l'ultima carta del mazzo e la rimuove dal mazzo tramite i metodi .back() e .pop_back() dei vettori. Inoltre, se sono rimaste meno di 30 carte lancia un'eccezione del tipo std::runtime_error.
- getDeck(): ritorna un vettore di carte come const ed è usato solo per l'implementazione dei test.

## 2.3 Classe Hand

La classe Hand rappresenta la mano di gioco e fornisce metodi per calcolare il punteggio ed eseguire le azioni tipiche di una partita a blackjack. È definita in maniera simile a Deck come un vettore dinamico di oggetti di tipo Card e, per ragioni di flessibilità, è dotata di due costruttori: uno vuoto e uno con già due carte, come nella condizione iniziale di una partita a blackjack. I suoi metodi rilevanti sono:

- int handScore() const: restituisce il punteggio del giocatore grazie all'algoritmo std::accumulate. Si occupa inoltre di gestire il punteggio degli assi che vengono fissati a 11 se il punteggio è inferiore a 21 e a 1 altrimenti. Viene inoltre gestito il raro caso in cui si abbia in mano più di un asso.
- void handDraw(Deck& deck): è il metodo che permette al giocatore di pescare una carta dal mazzo e fa uso del metodo topCard della classe Deck. Tale metodo impedisce al giocatore di richiedere carta se il suo punteggio è superiore a 21 lanciando un std::runtime_error.
- Card handElement(unsigned int numeber) const: restituisce la carta in posizione number all'interno della mano. E' fondamentale per le funzioni di disegno e i test.
- bool blackjack(): ritorna true nel caso in cui il player abbia in mano un blackjack (punteggio di 21 con due sole carte), è un metodo importante ai fini del gioco.
- int handSize(): restituisce il numero di carte nella mano di un giocatore: si tratta di un metodo fondamentale per il testing e alcune meccaniche di gioco.

I metodi rimanenti sono utilizzati unicamente per la fase del testing.

## 2.4 Classe CardRenderer

Una volta avute a disposizione le classi Hand e Card era necessario una classe con metodi in grado di disegnare tali carte sulla schermata principale di SFML. I semi che vengono disegnati sulle carte non sono altro che immagini scaricate da internet e poste in una cartella suits. Il primo passo è stato dunque quello di permettere al programma di caricare il font e i semi dal percorso tramite funzioni di tipo loadFromFile che ci sono state suggerite dall'intelligenza artificiale insieme alla relativa sintassi. A questo punto abbiamo associato due metodi alla classe:

- generateCardTexture(const Card& card): prende in input un oggetto di tipo Card e genera una nuova renderTexture composta da un rettangolo bianco dotato del valore in alto a sinistra e del seme al centro.
- drawCard(sf::RenderWindow&, const Card&, float x, float y, float angle): non fa altro che creare o recuperare dalla cache la texture, creare con essa un oggetto di tipo sf::sprite e disegnarla sulla schermata alle coordinate inserite come x e y e ruotandola di un angolo specificato come float angle (in gradi).

Negli stessi file abbiamo inserito anche le funzioni:

- void drawAdditionalCards(sf::RenderWindow& window, el::CardRenderer& renderer, GameState& state), tale funzione disegna le carte che vengono pescate dai giocatori durante il turno.
- void drawInitialCards(sf::RenderWindow& window, el::CardRenderer& renderer, GameState& state), tale funzione disegna le carte già presenti nelle mani dei giocatori a inizio turno.

## 2.5 Base graphics

Basegraphics raccoglie una serie di metodi di base per disegno e scritte, in particolare:

- void drawText(sf::RenderWindow& window, sf::Font& font, const std::string& str, float x, float y, int size, sf::Color color, float angle_of_rotation): è un metodo piuttosto semplice in grado di disegnare una stringa che gli viene passata in argomento, con spefiche posizioni, colore e dimensioni e ruotata dell'angolo assegnato.

- void drawCircle(sf::RenderWindow& window, float x, float y, float raggio, sf::Color sfondo, float thickness, sf::Color bordi): metodo molto simile al precedente, riceve nell'ordine la finestra dove essere disegnato, le coordinate del centro, il raggio, il colore dello sfondo, lo spessore e il colore dei bordi.

- sf::Text getText(sf::RenderWindow& window, sf::Font& font, const string&, float x, float y, unsigned int size, sf::Color& color, float angle_of_rotation): è un metodo quasi uguale al già menzionato drawText, ma a differenza di esso ritorna un elemento di testo. E' utile per generare testi dinamici e interattivi.

- void drawRect(sf::RenderWindow& window, float x, float y, float w, float h, sf::Color sfondo, float thickness, sf::Color bordi, float angle_of_rotation): disegna sulla finestra un rettangolo centrato in x,y di larghezza w e altezza h, del colore specificato, con una rotazione pari ad angle_of_rotation e i bordi di spessore pari a thickness e del colore specificato.

- sf::RectangleShape rectangularButton(sf::RenderWindow& window, float x, float y, float w, float h, sf::Color sfondo, float thickness, sf::Color bordi, float angle_of_rotation): è apparentemente uguale al metodo precedente, ma la differenza sta nel return statement. Tale funzione infatti disegna sì un rettangolo, ma ritorna anche un oggetto di tipo sf::RectangleShape. Ciò è reso necessario dal fatto che si tratta di un bottone e il suo utilizzo è necessario nel main.

- std::vector <sf::Text> createCurvedText(const std::string& text, const sf::Font& font, unsigned int charSize,sf::Vector2f center, float radius, float startAngleDeg, float totalAngleDeg): questo metodo è necessario per creare scritte curve ed è stato suggerito dall'AI. Per prima cosa calcola lo step angolare dividendo l'angolo totale per il numero di intervalli tra una lettera e l'altra (n lettere - 1). In seguito per ogni carattere calcola l'angolo corrispondente, lo converte in radianti e tramite operazioni di seno e coseno trova le coordinate x e y. Ognuno di essi è trasformato in un sf::text con font e dimensione specificati, coordinate ricavate come sopra e ruotato dell'angolo corrispondente; in seguito esso è aggiunto al vettore di output. La funzione getLocalBounds() serve a ottenere le dimensioni del carattere per centrarlo meglio. Il metodo ritorna un vettore di oggetti di tipo sf::Text pronti per essere disegnati.

- sf::Sprite getSprite(sf::RenderWIndow& window, float x, float y, float x_scale, y_scale, sf::Texture texture): restituisce una sprite utilizzabile dalla finestra SFML.

## 2.6 Classe Buttons

Questa classe è stata creata separatamente dal resto delle funzioni di grafica perchè tali oggetti hanno una funzione duplice: devono sì essere disegnati alla pari di altri elementi grafici, ma, essendo bottoni che devono garantire al giocatore la possibilità di interagire con il programma, è necessario che abbiano un nome ben definito. La struttura della classe è molto semplice: i fields privati sono i bottoni e il font e il costruttore inizializza tali bottoni servendosi del metodo RectangularButtons definito in BaseGraphics. La funzione membro drawFirstButtons(sf::RenderWindow& window) permette di disegnare i 3 bottoni della schermata di base tutti insieme, ed in modo analogo la funzione void drawPayingModeButtons(sf::RenderWindow& window) disegna i riquadri della fase di puntata. Inoltre sono presenti dei getter che, se chiamati, restituiscono il relativo bottone by reference per soddisfare le esigenze di interazione.

## 2.7 Classe GameState

Questa classe è il fulcro del programma e contiene tutti i metodi necessari al corretto svolgimento della partita e gestisce lo stato globale del gioco. Essa presenta una serie di variabili membro fondamentali:

- Il mazzo e le mani di gioco;
- std::vector<bool>phases: è un vettore di booleani il cui stato indica in che fase di trova il gioco;
- int your_score: indica il punteggio attuale del giocatore umano;
- una serie di flag di stato che tengono traccia dello stato della partita e delle azioni del giocatore.

La classe è dotata due costruttori diversi, uno di default che viene chiamato al primo avvio del gioco, e uno che, basandosi sul metodo resetRound(vedi sotto) prende in input le fish rimaste al giocatore e viene chiamato all'inizio di ogni round. La classe presenta inoltre un ampio numero di metodi pubblici:

- void resetRound(float your_fishes): Viene chiamato prima di ogni turno, reinizializza il mazzo, distribuisce le carte iniziali e azzera i flag di stato.
- void advanceTurn(int current_turn): aggiorna il vettore phases, ponendo a false l'elemento nella posizione che gli viene passata e impostando a true il successivo.
- void dealerPlay(), void bot1Play() e void bot2Play(): gestiscono in automatico la logica di gioco dei giocatori gestiti dal "computer", secondo dei criteri molto semplici.
- void handleBetting(sf::RenderWindow& window, Buttons& buttons, sf::Font& font, float& fishes_left, int& actual_bet, std::string& bet_input, std::string& error_message, sf::Clock& error_timer, sf::Text& bet_text, sf::Clock& stop, std::function¡void()¿& afterWait, bool bettingMode, const sf::Event& event): Questo metodo gestisce la fase di puntata che avviene nella schermata di overlay. Controlla che l'input di bet sia accettabile (un intero positivo minore delle fish con cui si entra) e, premendo il tasto "ok" sulla schermata o il tasto Enter della tastiera, permette di iniziare a giocare chiamando resetRound e advanceTurn;
- void playerHandleEvent(sf::RenderWindow& window, sf::Clock& stop, std::function <void() >& afterWait, float& fishes_left, int& actual_bet, Buttons& buttons, sf::Font& font, sf::Text& bet_text, const sf::Event& event): gestisce ed elabora gli eventi di gioco del giocatore umano. In particolare gestisce gli input dei bottoni hit, stand e double e controlla che il punteggio del giocatore non ecceda 21;
- void drawPlayerUI(sf::RenderWindow& window): disegna un pallino verde nello slot del giocatore umano per segnalare che è il suo turno;
- void drawError(sf::RenderWindow& window, const std::string& message, sf::Font& font): gestisce l'eccezione che viene lanciata se il giocatore non ha fish sufficienti per la scommessa che intende fare.
- void calculatePayOut(float& fishes_left, int actual_bet): gestisce la logica dei pagamenti a seconda dei punteggi del giocatore umano e del banco.

## 2.8 StaticTable

StaticTable contiene tre funzioni:

- void first_window(sf::RenderWindow& first_window, sf::Font& font), che disegna la schermata con le regole e le fish iniziali;
- void drawStaticTable(sf::RenderWindow& window, sf::Font& font, float fishes_left, int score, const sf::Texture texture1, const sf::Texture texture2, std::vector <sf::Text >& allLetters), che disegna tutte le parti statiche del tavolo da gioco tramite una combinazione delle funzioni esposte in base_graphics.
- void overlay(sf::RenderWindow& window, Buttons& buttons, GameState& state, sf::Font& font, sf::Text& bet_text, const std::string& bet_input): gestisce la schermata semitrasparente che appare quando ci si trova nella fase di scommessa. Inoltre alla fine del gioco controlla lo stato del gioco e ne stampa a schermo l'esito di conseguenza.

### 2.9 First Window

First Window contiene una singola funzione:

- float firstwindow(sf::RenderWindow& window, sf::Font& font), che si occupa di gestire la prima schermata, disegnandola e gestendo il box di input, compresi i casi d'eccezzione lanciando errori a schermo.

### 2.10 Main

L'architettura logica del main ha dovuto tenere conto delle forti limitazioni imposte dal ciclo while necessario a tenere aperta la finestra grafica SFML, perciò la sua implementazione è stata tenuta il più semplice possibile. Le fish del giocatore sono inizializzate tramite un input inserito dallo stesso nella schermata iniziale, aperta dalla funzione firstWIndow di cui sopra. In seguito si entra nel ciclo principale nel quale vengono richiamate le funzioni di gioco, definite in methods. La prima ad essere chiamata è bettingMode che gestisce la schermata di scommessa. Questa si chiude quando viene premuto il tasto "ok" e il turno ha inizio. Vengono dunque chiamate le funzioni che gestiscono i turni. Il gioco procede grazie al metodo advanceTurn della classe GameState e chiamando la funzione propria del giocatore che gioca in quel momento tramite un if sullo stato del gioco. Una volta terminato il turno del dealer, il gioco arriva allo stato di pagamento chiamando la funzione calculatePayout che al suo termine riporta il gioco allo stato iniziale, permettendo di introdurre una nuova scommessa.

## 3 Librerie Esterne

Per creare l'interfaccia grafica ci siamo avvalsi della libreria SFML, scaricabile con il seguente comando:

%sudo apt install libsfml-dev

Come testing è stato usato doctest, come formattatore clang-format.

## 4 Compilazione

Per compilare ed eseguire il programma si è usato il build system Cmake. Con l'aiuto dell'AI ne abbiamo scritto un file Cmake.txt che oltre a compilare il programma e

## 5 Implementazione dei Test

Per verificare la correttezza del programma abbiamo implementato vari test, utilizzando gli strumenti forniti da Doctest. Tali test sono suddivisi in vari file mirati a testare singolarmente i vari file che compongono il progetto: essi sono card.test.cpp, hand.test.cpp e methods.test.cpp. In questi test abbiamo verificato il corretto funzionamento del programma controllando innanzitutto l'operatore == di Card, la corretta generazione del mazzo di carte (Classe Deck) e i suoi vari metodi, come "top_card()" e "size()". In modo analogo sono stati verificati tutti i metodi di Hand, così come i costruttori di ciascuna classe. Abbiamo posto particolare attenzione ai casi limite, controllando che venissero emessi i giusti messaggi di errore: un paio di esempi sono il pescare una carta da una mano anche quando il punteggio di essa ha già raggiunto il 21 e il pescare carte da un mazzo con troppe poche carte.