**FACULTATEA DE INGINERIE ELECTRICĂ ȘI ȘTIINȚA CALCULATOARELOR**

**CALCULATOARE**

**Joc simplu Blackjack in consola**

Student: Aydogan Mert

Disciplina: Programare orientata pe obiecte

Grupa: 3121 A

# Cuprins

[Cuprins 1](#_Toc137055952)

[Introducere 2](#_Toc137055953)

[Teme si motivarea alegerii 3](#_Toc137055954)

[3 . CONSTRUIȚI ȘI INFORMAȚII DESPRE CODURI 5](#_Toc137055955)

[Player.cpp 5](#_Toc137055956)

[Human.cpp 6](#_Toc137055957)

[Game.cpp 7](#_Toc137055958)

[Stocarea fișierelor și procesul de păstrare a înregistrărilor statistice. 9](#_Toc137055959)

[Lucruri de imprimare 10](#_Toc137055960)

[4\*\*.    MANUAL  DE  UTILIZARE\*\* 11](#_Toc137055961)

[5.    CONCLUZII 12](#_Toc137055962)

[6.    BIBLIOGRAFIE 13](#_Toc137055963)

[6.1.                CARTI 13](#_Toc137055964)

[6.2.                SURSE BIBLIOGRAFICE DIVERSE 13](#_Toc137055965)

# Introducere

Această documentație se concentrează pe dezvoltarea unui joc de consolă folosind limbajul C++, aderând la principiile programării orientate pe obiecte (OOP). Jocul ales este Blackjack, un joc de cărți popular jucat în cazinourile din întreaga lume. Textul discută diverse aspecte și provocări potențiale asociate cu implementarea jocului, cum ar fi valorile cărților, mecanismele de joc, problema de a „epuiza”, numărarea cărților, avantajul dealer-ului și regulile casei.

Programarea orientată pe obiecte este o paradigmă de dezvoltare software care organizează codul în jurul obiectelor, încapsulând date și comportament. Articolul explorează modul în care jocul de blackjack poate fi analizat și implementat dintr-o perspectivă OOP, utilizând principii precum încapsularea, moștenirea și polimorfismul.

În plus, textul oferă informații despre detaliile de implementare a jocului, inclusiv clase precum Player, Human și Game în C++. Acesta acoperă funcții legate de pariuri, împărțirea cărților, acțiunile jucătorilor, acțiunile dealerului, condițiile de câștig și salvarea/încărcarea datelor de joc.

În general, această documentație servește ca un ghid cuprinzător pentru implementarea unui joc Blackjack OOP bazat pe consolă în C++.

# Teme si motivarea alegerii

**FORMULAREA TEMULUI**

Creați un joc de consolă folosind limbajul C + + în conformitate cu principiile programării orientate pe obiecte.

**MOTIVARE PENTRU ALEGEREA TEMA**

Blackjack, cunoscut și sub numele de douăzeci și unu, este un joc popular de cărți jucat în cazinourile din întreaga lume. Obiectivul jocului este de a avea o valoare a mâinii mai aproape de 21 decât mâna dealerului fără a depăși 21. Iată câteva aspecte cheie și potențiale probleme asociate jocului:

1. Valori card: În blackjack, cardurile numerice ( 2-10 ) își merită valoarea nominală, cărțile de față ( jack, queen, king ) valorează 10, iar Ace poate valora fie 1, fie 11, în funcție de preferințele jucătorului.
2. Gameplay: Jocul începe cu jucătorii care pun pariuri. Fiecare jucător și dealer primesc două cărți. Cărțile jucătorilor sunt de obicei distribuite cu fața în sus, în timp ce prima carte a dealerului este cu fața în sus, iar a doua carte este cu fața în jos. Jucătorii iau decizii, cum ar fi lovirea ( solicitarea unei alte cărți ), în picioare ( care nu mai solicită carduri ), dublarea ( dublarea pariului inițial și primirea încă o carte ), sau împărțirea ( împărțind o pereche în două mâini separate ). Dealerul respectă reguli specifice pentru lovire sau în picioare.
3. Problema bustului: O problemă comună în blackjack este „bustul”, ceea ce înseamnă depășirea unei valori de mână de 21. Dacă mâna unui jucător se aruncă, pierde pariul indiferent de mâna dealerului. Evitarea bustului este crucială în jocul de blackjack strategic.
4. Problema numărării cardurilor: numărarea cardurilor este o strategie folosită de jucători calificați pentru a obține un avantaj față de cazinou. Aceasta implică urmărirea numărului relativ de carduri înalte până la cardurile joase rămase pe punte. Cu toate acestea, numărarea cardurilor este considerată ilegală în multe cazinouri și folosesc diferite contramăsuri pentru prevenirea sau detectarea acesteia.
5. Problema avantajului dealerului: în blackjack-ul tradițional, dealerul are un ușor avantaj statistic față de jucători. Acest avantaj provine de la faptul că jucătorii trebuie să acționeze mai întâi, iar dacă au bust, pierd indiferent de mâna dealerului. Cu toate acestea, jucătorii au opțiunea de a lua decizii strategice, cum ar fi poziția pe valori mai mici atunci când cardul vizibil al dealerului este slab.
6. Problema regulilor casei: Cazinouri diferite sau variații de blackjack pot avea regulile lor specifice. De exemplu, unele cazinouri permit dublarea pe oricare două cărți, în timp ce altele o limitează la totaluri specifice. Înțelegerea regulilor și variațiilor jocului este importantă pentru a vă adapta strategia în consecință.

**Informații despre OOP**

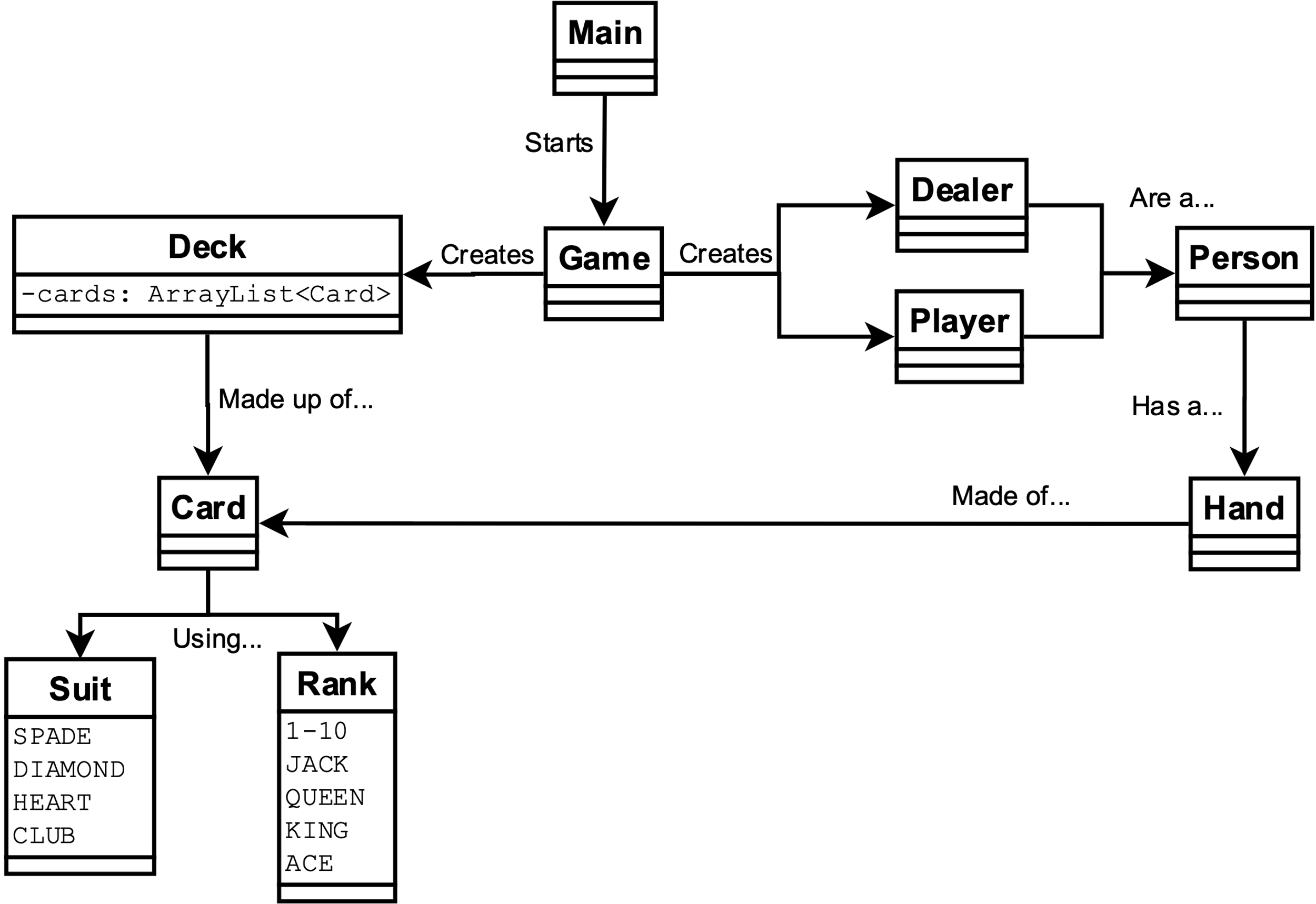
Programarea orientată pe obiecte ( OOP ) este o paradigmă de dezvoltare software care accentuează organizarea codului în jurul obiectelor, care sunt cazuri de clase care încapsulează datele și comportamentul. Principiile OOP, cum ar fi încapsularea, moștenirea și polimorfismul oferă o abordare structurată a proiectării și implementării sistemelor software. În acest articol, vom explora modul în care jocul de blackjack poate fi analizat dintr-o perspectivă OOP și vom discuta despre respectarea principiilor OOP.

**BLACKJACK ÎN C++**

**Diagrama de activitate și pași**

A diagram of activity diagram

Description automatically generated with low confidence



1. Start: Jocul începe.
2. Pariuri de loc: Jucătorii își plasează pariurile înainte de distribuirea cărților.
3. Distribuie carduri inițiale: dealerul oferă două cărți fiecărui jucător, de obicei cu fața în sus, și oferă două cărți pentru ei înșiși, cu una față în sus și cealaltă cu fața în jos.
4. Acțiuni jucător: Jucătorii iau decizii pe baza valorii mâinii și a cardului vizibil al dealerului. Aceștia pot alege să acceseze ( să primească o carte suplimentară ), să stea ( să încheie rândul lor fără să primească mai multe carduri ), să dubleze ( să își dubleze pariul și să mai primească încă o carte ), sau împărțiți ( dacă au o pereche, împărțiți-le în două mâini separate ).
5. Acțiuni de dealer: După ce toți jucătorii și-au încheiat acțiunile, este rândul dealerului să acționeze. Dealerul își dezvăluie cardul cu fața în jos și respectă reguli specifice pentru lovire sau în picioare, pe baza valorii totale a mâinii.
6. Verificați rezultatele: odată ce dealerul își finalizează acțiunile, mâinile sunt comparate pentru a determina câștigătorii. Jucătorii care au o valoare a mâinii mai mare decât mâna dealerului fără a depăși 21 de victorii, în timp ce jucătorii cu o valoare a mâinii care depășește 21 pierd.
7. Sfârșit: jocul se încheie, iar jucătorii pot alege să joace din nou sau să părăsească masa.

## 3 . CONSTRUIȚI ȘI INFORMAȚII DESPRE CODURI

### Player.cpp

1. Codul include fișierul antet "player.h" care conține, probabil, declarația clasei de jucători și orice dependențe necesare. De asemenea, include biblioteca **iostream** pentru operațiuni de intrare / ieșire.
2. Codul definește un constructor implicit pentru clasa de jucător. Acest constructor inițializează obiectul jucătorului cu valori implicite. Valorile implicite sunt următoarele:
   * Nume: „Necunoscut”"
   * Pariu: 0
   * Numerar: 1000
   * Câștigări: 0
   * Pierde: 0
3. În urma constructorului, există mai multe funcții de getter și setter definite pentru accesarea și modificarea atributelor jucătorului:
   * getName ( ): Returnează numele jucătorului ca șir.
   * getBet ( ): Returnează suma pariului jucătorului ca număr întreg.
   * getCash ( ): returnează suma de numerar pe care o are jucătorul ca număr întreg.
   * getWins ( ): Returnează numărul de victorii pe care jucătorul le are ca număr întreg.
   * getLoses ( ): Returnează numărul de pierderi pe care jucătorul le are ca număr întreg.
   * setName ( std :: string nm ): Setează numele playerului folosind șirul furnizat.
   * setBet ( int b ): Setează pariul pentru joc prin deducerea sumei pariului din numerarul jucătorului și adăugarea acestuia la pariu.
   * addCash ( int c ): adaugă suma specificată de numerar la suma de numerar existentă a jucătorului.
   * incrementWins ( ): Crește numărul de victorii cu unul singur.
   * incrementLoses ( ): Crește numărul pierderilor cu una.
4. În cele din urmă, există o funcție de joc numită clearCards ( ) definită. Această funcție șterge mâna jucătorului și resetează suma pariului la 0. Este probabil ca clasa de jucător să moștenească de la o altă clasă numită „umană” și să-și extindă funcționalitatea adăugând capacitatea de a șterge cărțile și de a reseta pariul.

În general, acest cod reprezintă o implementare de bază a unei clase de jucători pentru un joc. Oferă metode de setare și regăsire a atributelor jucătorului, precum și de a efectua acțiuni legate de joc, cum ar fi setarea pariurilor, actualizarea numerarului și urmărirea câștigurilor și pierderilor.

### Human.cpp

1. Codul include fișierul antet "\*\* human.h \*\*" și biblioteca \*\* iostream \*\* pentru operațiuni de intrare / ieșire.
2. Codul definește un constructor implicit pentru clasa umană. Acest constructor inițializează atributul „sumă” la 0.
3. Codul oferă o funcție getter numită "getSum ( )" care returnează valoarea atributului "sumă. Înainte de a returna valoarea, apelează funcția „switchAce ( )”, care ajustează valoarea atributului „sumă”, dacă este necesar.
4. Funcția "switchAce ( )" este responsabilă de manipularea cazului special al unui card Ace în mână. Dacă suma curentă a mâinii depășește 21, iterează prin cărțile din mână și verifică dacă există un As care nu a fost trecut anterior la o valoare de 1. Dacă se găsește un astfel de Ace, își schimbă valoarea la 1 ( scăzând 10 din suma ) și o marchează ca „blocată” pentru a preveni modificările ulterioare ale valorii.
5. Funcția "addCard ( )" ia un obiect de card ca intrare și îl adaugă la mâna omului. De asemenea, ajustează valoarea cardului dacă este o carte de față ( mai mare de 10 ) sau un Ace ( își stabilește valoarea la 11 ). Suma este actualizată în consecință.
6. Funcția „clearCards ( )” șterge mâna omului prin eliminarea tuturor cărților și setarea sumei la 0.
7. Funcția „printCards ( )” tipărește cărțile umane pe consolă într-un format stilizat. Folosește bucle cuibărite pentru a itera printre rândurile și coloanele afișajului cardului. Numărul și costumul fiecărei cărți sunt tipărite într-un mod vizual atrăgător.

În general, acest cod reprezintă implementarea clasei umane, care include metode de adăugare a cărților la mână, ajustarea sumei în funcție de prezența Așilor, ștergerea mâinii, și imprimarea cardurilor într-un format stilizat.

### Game.cpp

1. Codul include mai multe fișiere antet și biblioteci necesare implementării jocului.
2. Codul definește un constructor implicit pentru clasa Game, care inițializează un obiect de punte.
3. Funcția "dealDealer ( )" oferă carduri dealerului până când suma lor depășește 17 sau suma dealerului devine mai mare decât suma jucătorului. Se returnează adevărat dacă dealerul termină de a face și este fals dacă este îndeplinită o condiție de câștig.
4. Funcția "compareSum ( )" compară sumele jucătorului și dealerului și determină câștigătorul. Tipărește rezultatul și returnează un personaj reprezentând rezultatul ( 'p' pentru câștigul jucătorului, 'd' pentru câștigul dealerului, 'n' pentru o remiză ).
5. Funcția "checkWins ( )" verifică condițiile de câștig apelând funcția "checkEnd ( ). Crește contoarele câștigului / pierderii și ajustează în consecință numerarul jucătorului. Se întoarce adevărat dacă este îndeplinită o condiție de câștig.
6. Funcția "checkEnd ( )" verifică condițiile de bust sau blackjack atât pentru dealer, cât și pentru jucător. Tipărește rezultatul și returnează un personaj reprezentând rezultatul ( 'd' pentru câștigul dealerului, 'p' pentru câștigarea jucătorului, 'f' pentru nicio condiție de câștig ).
7. Funcția "startBet ( )" permite jucătorului să își plaseze pariul prin creșterea sau scăderea valorii pariului. Se returnează adevărat dacă procesul de pariu este finalizat.
8. Funcția "startGame ( )" începe jocul ocupând două cărți fiecare jucătorului și dealerului. Apoi, îl solicită pe jucător să lovească sau să stea până când jucătorul stă în picioare sau are busturi. Se întoarce adevărat dacă jucătorul stă fără să se plictisească.
9. Funcția "beginGame ( )" gestionează bucla principală a jocului. Initializează jocul, permite jucătorului să plaseze un pariu, începe jocul și gestionează condițiile de câștig. De asemenea, oferă o opțiune pentru a salva jocul.
10. Funcția „beginMenu()” este punctul de intrare pentru joc. Afișează un meniu pentru ca jucătorul să aleagă opțiuni, cum ar fi pornirea unui joc nou, încărcarea unui joc, vizualizarea statisticilor, citirea instrucțiunilor sau părăsirea jocului.
11. Codul include funcții pentru salvarea și încărcarea datelor de joc folosind gestionarea fișierelor binare.
12. Codul include diverse funcții pentru imprimarea informațiilor legate de joc, cum ar fi statistici, instrucțiuni și ecranul jocului.

În general, acest cod reprezintă implementarea unei clase de joc numită „Joc”, care gestionează jocul, condițiile de câștig, pariurile, salvarea/încărcarea datelor de joc și tipărirea informațiilor legate de joc.

A screenshot of a computer program

Description automatically generated with medium confidenceA screenshot of a computer program

Description automatically generated with medium confidence

1. **goliți jocul :: loadGame ( )**: Această funcție este responsabilă de încărcarea unui joc salvat dintr-un fișier binar. Acesta solicită utilizatorului să introducă un nume de fișier și adaugă „.bin” la numele de fișier introdus pentru a forma calea fișierului. Apoi încearcă să deschidă fișierul în modul de introducere cu format binar folosind **std :: fstream**. Dacă fișierul este deschis cu succes, citește numele jucătorului, numerar, câștiguri și pierderi date din fișier folosind **f1.read ( )** și le stochează în variabilele corespunzătoare. Apoi închide fișierul și actualizează atributele obiectului jucătorului cu datele încărcate folosind funcții de membru precum **player.setName ( )**, **player.addCash ( )** , \*\* player.incrementWins (**, și \*\* player.incrementLoses ( )**. Dacă fișierul nu există, acesta apelează funcția **beginMenu ( )** cu un mesaj de eroare.
2. **game goid :: printStatistics ( )**: Această funcție tipărește statisticile jocului. Șterge ecranul, tipărește titlul jocului și afișează statisticile folosind funcția **s.print ( )**, care tipărește probabil datele statistice stocate. După tipărirea statisticilor, solicită utilizatorului să apese orice cheie pentru a continua.
3. **goliți jocul :: printInstructions ( )**: Această funcție tipărește instrucțiunile jocului. Șterge ecranul, tipărește titlul jocului și afișează instrucțiunile folosind **Print :: instrucțiuni ( )** funcție, care returnează probabil un șir care conține instrucțiunile. După tipărirea instrucțiunilor, solicită utilizatorului să apese orice cheie pentru a continua.
4. **goliți jocul :: printTop ( )**: Această funcție tipărește secțiunea superioară a ecranului jocului. Șterge ecranul, tipărește titlul jocului și afișează informații precum numărul de cărți din punte, numerarul jucătorului, pariul jucătorului și numele jucătorului.
5. **golit Game :: printBody ( )**: Această funcție tipărește secțiunea principală a corpului ecranului jocului. Apelează **printTop ( )** pentru a imprima secțiunea superioară, apoi imprimă granița dealerului, prima carte a dealerului folosind \*\* dealer.printFirstCard ( ) \* , granița jucătorului, și cărțile jucătorului folosind **player.printCards ( )**. În cele din urmă, tipărește suma cărților jucătorului.

Aceste funcții sunt probabil parte dintr-o implementare de clasă mai mare pentru un joc de blackjack de linie de comandă. Ele se ocupă de diferite aspecte ale jocului, cum ar fi încărcarea unui joc salvat, tipărirea statisticilor și instrucțiunilor și afișarea ecranului jocului cu informații relevante.

### Stocarea fișierelor și procesul de păstrare a înregistrărilor statistice.

### 1. Declarați un flux de fișiere \*\*`std::fstream f1;`\*\* și o variabilă șir \*\*`std::string filename;`\*\* pentru fișier și, respectiv, nume de fișier.

### 2. Setați calea în care va fi localizat fișierul folosind \*\*`std::string path = "data/";`\*\*. În acest exemplu, este folosit directorul „data/”.

### 3. Solicitați utilizatorului să introducă un nume de fișier folosind \*\*`std::cout<<"Introduceți numele fișierului: ";`\*\* și stocați-l în variabila \*\*`filename`\*\* folosind \*\*`std::cin> >nume fișier;`\*\*.

### 4. Transformați numele fișierului introdus în minuscule folosind \*\*`std::transform(filename.begin(), filename.end(), filename.begin(), ::tolower);`\*\*.

### 5. Utilizați o buclă (\*\*`do-while`\*\* buclă) pentru a solicita în mod repetat un nume de fișier până când numele de fișier introdus nu este „statistici”.

### 6. Când numele fișierului introdus se potrivește cu „statistici”, ieșiți din buclă și construiți calea fișierului prin concatenarea \*\*`filename`\*\* și adăugând extensia „.bin”: \*\*`path+=filename+.bin";` \*\*.

### 7. Deschideți fișierul folosind \*\*`f1.open(cale, std::ios::in | std::ios::binary);`\*\*. Este folosită calea fișierului specificată, iar \*\*`std::ios::in`\*\* indică faptul că fișierul este deschis pentru citire, în timp ce \*\*`std::ios::binary`\*\* indică faptul că fișierul conține binar date.

### 8. Dacă fișierul este deschis cu succes (\*\*`if(!f1.fail()) { ... }`\*\*), continuați cu următorii pași. În caz contrar, dacă fișierul nu se deschide, afișați un mesaj de eroare și apelați funcția \*\*`beginMenu`\*\* pentru a reveni la meniul de pornire al programului.

### 9. Creați variabile pentru a stoca numele jucătorului, suma în numerar, numărul de câștiguri și numărul de pierderi: \*\*`std::string sName; int sCash; int sWins; int sLoses;`\*\*.

### 10. Creați o variabilă \*\*`nameSize`\*\* pentru a stoca dimensiunea numelui și citiți valoarea acestuia din fișier folosind \*\*`f1.read((char\*)&nameSize, sizeof(nameSize));`\*\*.

### 11. Redimensionați variabila \*\*`sName`\*\* pentru a avea o lungime de \*\*`nameSize`\*\*: \*\*`sName.resize(nameSize);`\*\*.

### 12. Citiți șirul din fișier în variabila \*\*`sName`\*\* folosind \*\*`f1.read(&sName[0], sName.size());`\*\*.

### 13. Citiți variabilele \*\*`sCash`\*\*, \*\*`sWins`\*\* și \*\*`sLoses`\*\* din fișier folosind \*\*`f1.read()`\*\*.

### 14. Odată ce citirea fișierului este completă, închideți fișierul cu \*\*`f1.close()`\*\*.

### 15. Setați numele jucătorului folosind \*\*`player.setName(sName)`\*\*.

### 16. Ajustați suma de numerar a jucătorului cu diferența dintre suma de numerar încărcată și suma de numerar curentă folosind \*\*`player.addCash(sCash - player.getCash())`\*\*.

### 17. Creșteți numărul de câștiguri al jucătorului până când acesta se potrivește cu numărul de câștiguri încărcat folosind o buclă: \*\*`while(player.getWins()!=sWins){ player.incrementWins(); }`\*\*.

### 18. Creșteți numărul de pierderi al jucătorului până când acesta se potrivește cu numărul de pierderi încărcat folosind o buclă: \*\*`while(player.getLoses()!=sLoses){ player.incrementLoses(); }`\*\*.

### 19. Dacă fișierul nu a fost găsit sau a întâmpinat o eroare în timpul procesului de deschidere a fișierului, afișați un mesaj de eroare adecvat și reveniți la meniul de început folosind funcția \*\*`beginMenu`\*\*.

### Lucruri de imprimare

## 1. \*\*`void Game::printStatistics()`\*\*: Această funcție este responsabilă pentru tipărirea statisticilor jocului. Pașii implicați sunt următorii:

## - Ștergeți ecranul folosind funcția \*\*`clearscr()`\*\*.

## - Imprimați titlul jocului în galben folosind \*\*`std::cout<<yellow<<Print::title\_blackjack()<<def<<"\n";`\*\*.

## - Tipăriți cuvântul „Statistici” în culoarea verde deschis folosind \*\*`std::cout<<"\n"<<lightGreen<<Print::statistics()<<def<<"\n";`\*\*.

## - Apelați funcția \*\*`s.print()`\*\* pentru a imprima statisticile.

## - Afișați un mesaj pentru a apăsa orice tastă pentru a continua să utilizați \*\*`std::cout<<"\n\n\t(Apăsați orice tastă pentru a continua)\n";`\*\*.

## - Așteptați apăsarea tastei folosind \*\*`getch()`\*\*.

## 2. \*\*`void Game::printInstructions()`\*\*: Această funcție este responsabilă pentru tipărirea instrucțiunilor de joc. Pașii implicați sunt următorii:

## - Ștergeți ecranul folosind funcția \*\*`clearscr()`\*\*.

## - Imprimați titlul jocului în galben folosind \*\*`std::cout<<yellow<<Print::title\_blackjack()<<def<<"\n";`\*\*.

## - Tipăriți cuvântul „Instrucțiuni” în culoarea verde deschis folosind \*\*`std::cout<<"\n"<<lightGreen<<Print::instructions()<<def<<"\n";`\*\*.

## - Așteptați apăsarea tastei folosind \*\*`getch()`\*\*.

## 3. \*\*`void Game::printTop()`\*\*: Această funcție este responsabilă pentru tipărirea secțiunii de sus a jocului, inclusiv informații precum cărți, numerar, pariu și numele jucătorului. Pașii implicați sunt următorii:

## - Ștergeți ecranul folosind funcția \*\*`clearscr()`\*\*.

## - Imprimați titlul jocului în galben folosind \*\*`std::cout<<yellow<<Print::title\_blackjack()<<def<<"\n";`\*\*.

## - Imprimați numărul de cărți din pachet în culoare roșu deschis, urmat de suma în numerar a jucătorului în culoarea verde deschis, pariul jucătorului în culoarea magenta deschisă și numele jucătorului în culoarea albastru deschis.

## 4. \*\*`void Game::printBody()`\*\*: Această funcție este responsabilă pentru tipărirea corpului principal al jocului, inclusiv cărțile dealer-ului și ale jucătorului. Pașii implicați sunt următorii:

## - Apelați funcția \*\*`printTop()`\*\* pentru a imprima secțiunea de sus a jocului.

## - Imprimați un chenar pentru secțiunea dealer-ului în culoare roșu deschis folosind \*\*`std::cout<<lightRed<<Print::dealer\_border()<<def;`\*\*.

## - Imprimați primul card al dealer-ului folosind funcția \*\*`dealer.printFirstCard()`\*\*.

## - Imprimați un chenar pentru secțiunea jucătorului în culoarea cyan deschis folosind \*\*`std::cout<<lightCyan<<Print::player\_border()<<def;`\*\*.

## - Imprimați cărțile jucătorului folosind funcția \*\*`player.printCards()`\*\*.

## - Imprimați suma cărților jucătorului în verde deschis folosind \*\*`std::cout << lightGreen<< "\nSum: "<<lightRed<< player.getSum()<<def<<"\n"; `\*\*.

## 4.    MANUAL  DE  UTILIZARE

Programul funcționează ca o consolă. Când utilizatorul intră în program pentru prima dată, apare un meniu și acest meniu acceptă doar valori între 1 și 5. Dacă utilizatorul nu introduce o valoare între 1-5, apare un mesaj de eroare care spune că utilizatorul a introdus o valoare incorectă.

În meniu; începe un joc nou, încarcă jocul, statistici, cum se joacă și ieși.

Jocurile a căror parte de statistici este înregistrată au informații precum bani, jocuri câștigate și pierdute.

Când creăm un joc nou, utilizatorului i se cere să introducă un nume de jucător. După introducerea numelui de utilizator, se apasă tasta „Enter” și apoi intră în joc. În această secțiune, utilizatorul: „Tasta W = intrare pariu”, „Tasta S = reducere pariu” și „Tasta R = OK”. Produce trei chei, „S” și „R” din aceste chei și acceptă un pariu fix de 5 USD.

După ce utilizatorul introduce un pariu, apare fața principală a jocului, care arată cărțile Dealer și User pe consolă cu o interfață grafică limitată pentru utilizator. Valoarea cardului utilizatorului este, de asemenea, indicată în partea de jos a cardului său.

Este de așteptat ca utilizatorul să apese „tasta H” pentru Hit și „tasta S” pentru Stand.

După acest proces, dacă utilizatorul câștigă, se adaugă 1 număr la secțiunea de scor câștigător de mai jos, dacă a pierdut, se adaugă 1 punct la secțiunea pierzătoare. (Banii pe care îi câștigi sau îi pierzi în fiecare joc vor cădea în sistem).

Utilizatorului i se prezintă opțiunea tasta Y dacă dorește să continue, tasta N dacă nu dorește să continue. După acest proces, apare secțiunea Doriți să salvați jocul, unde utilizatorul poate efectua operația dorită apăsând tasta Y sau N. Dacă utilizatorul apasă tasta Y, utilizatorului i se solicită un nume de fișier pentru salvare. Acest nume îl ajută pe utilizator să reia de unde a rămas mai târziu.

După înregistrare, programul se închide.

## 5.    CONCLUZII

Jocul blackjack poate fi analizat și implementat în conformitate cu principiile OOP. Încapsularea permite organizarea datelor și a comportamentului în obiecte coezive, în timp ce moștenirea și polimorfismul facilitează reutilizarea și extensibilitatea codului. Aplicarea principiilor OOP pe blackjack îmbunătățește mentenabilitatea codului, modularitatea și separarea preocupărilor. Vizualizarea blackjack-ului printr-un obiectiv OOP, dezvoltatorii pot crea un sistem bine structurat și adaptabil, care să se alinieze celor mai bune practici din industrie.

## 6.    BIBLIOGRAFIE

### 6.1.                Cărți

**C++ in the Lab: Lab Manual Paperback – Import, ISBN 0-13-03478-X, 16 September 2002**

**Object Oriented Programming (Oop) Using C++  –** • **ISBN-13 ‏ : ‎** 978-3846515860, **30 September 2011**

### 6.2.                Surse Bibliografice Diverse

<https://cplusplus.com>

<https://www.blackjackinfo.com>