7Septica

12.01.2022

Student:

Nume, Prenume

(grupa/semigrupa)

Chirilă Gheorghe Viorel

gr. 224

Istoric Versiuni

Data	Versiune	Descriere	Autor
13/11/2021	1.0	Au fost create clasele : Carte, Pachet, Joc_s	
14/11/2021	1.1	Au fost creată metoda de amestecare a cărților din pachet și clasa P_Mana	
30/11/2021	1.2	Crearea clasei Masa Separarea jocului în fereastra de start si fereastra principala de joc	
15/12/2021	1.5	A fost creată clasa Scor	
30/12/2021	1.7	A fost creată clasa Jucator	

Cuprins

1 Specificarea cerințelor software

1.1 Introducere

Proiectul constă în creare unui joc de cărți în rețea. Jocul ales pentru dezvoltarea aplicației este jocul de Septica jucat în doi jucători prin intermediul rețelei. În joc se folosesc 32 de cărți : 7,8,9,10,J,Q,K și A .Jocul se joacă în doi jucători scopul fiind de a aduna cât mai multe puncte la finalul jocului. Punctele sunt reprezentate de cărțile 10 și A câte un punct pentru fiecare carte în total 8puncte. Au fost create o aplicație server si o aplicație client, aplicația client se conectează la aplicația server, iar jocul pornește după stabilirea conexiunii prin amestecarea cărților de joc din pachet și distribuirea către jucători. Atât cărțile cât si pachetul de cărți sunt reprezentate sub forma unor obiecte în cadrul proiectului. În aplicația server se găsește codul principal *"nucleul"* jocului, pachetul, masa, cărțile de joc se află în aplicația server aplicația client având funcția principală de a afișa către jucători schimbările transmise de server și de a recepționa și transmite mai departe către server modificările făcute de jucător (ex. punerea unei cărți pe masa).

1.1.1 Obiective

Obiective ce trebuiesc realizate pentru proiect:

- crearea cărților de joc
- crearea pachetului format din cărțile de joc
- amestecarea cărților din pachet
- crearea unei clase care să rețină cărțile jucătorilor
- crearea unei clase care să rețină cărțile puse pe masa
- afișarea cărților de pe masa și cărțile jucătorilor
- posibilitate jucătorilor de a alege să continue runda sau nu
- adăugarea unei animații la amestecarea pachetului
- adăugarea unei animații la distribuirea cărților către jucători
- adăugarea unei animații la punerea unei cărți pe masa
- crearea unei ferestre de personalizare a feței din spate a cărților si a mesei de joc
- crearea unei ferestre care sa conțină descrierea si regulile jocului
- realizarea conexiunii dintre aplicația server si cea client
- codificarea cărților sub forma unor mesaje text pentru transmiterea prin rețea
- decodificarea mesajelor transmise prin rețea
- crearea unei clase care să rețină scorul si cărțile câștigate
- afișarea scorului și a cărților câștigate la finalul jocului

1.1.2 Definiţii, Acronime şi Abrevieri

În realizarea proiectului:

- variabilele care încep cu litera 'c_' sunt variabile de tipul Carte
- variabilele care încep cu litera 'p_' sunt variabile de tipul Pachet
- variabilele care încep cu litera 'm_' sunt variabile de tipul P_Mana
- variabilele care încep cu litera 'ma_' sunt variabile de tipul Masa
- variabilele care încep cu litera 'j_' sunt variabile de tipul Jucator
- variabilele care încep cu litera 's_' sunt variabile de tipul Scor
- variabilele care au în componența numelui '_temp' sunt variabile ajutătoare folosite în interiorul metodelor.

Aceleași notații în ceea ce privește numele variabilelor se aplică la vectori si pointeri.
Definiții:

- clasa enum se folosește pentru a da nume simbolice unui sir de valori pentru a putea fi mai ușor de folosit valorile din sir având nume specifice. În cadrul proiectului se folosesc doua

enumerații Pereche și Numar cu valori de la 0 la 4 si respectiv 7 la 15 pentru reprezentarea perechii și numărului unei cărți.
- BLANK și BLANK2 perechea și numărul folosite pentru a inițializa o carte albă fără pereche si număr folosită in constructorul implicit și la înlocuirea temporară a cărților din mâna jucătorilor pentru a marca poziția cărților care au fost puse pe masă în timpul unei runde.
- funcția rand() – generează valori aleatoare. La fiecare apel al funcției rand() valoarea generată se calculează pe baza unei valori anterioare numita *sămânță(seed)*
- funcția srand(int s) – modifică valoarea *seed* folosită de funcția rand() cu valoarea lui s
- time(0) – returnează timpul curent al calculatorului

Pentru mesajele transmise pe rețea:
- "*" – început de mesaj
- "#" – sfârșit de mesaj
- "07" – codificarea unei cărți sub forma unui sir de caractere (0 – perechea INIMA; 7 – numărul SAPTE)
- "_" – marchează spațiile dintre cuvinte
- "cm" – urmează o carte ce trebuie afișată pe masă
- "Start" – mesajul de început transmis de către server urmat de cele patru cărți codificate pentru afișarea in aplicația client

1.1.3 Tehnologiile utilizate

Pentru realizarea proiectului au mai fost utilizate:
- aplicația *Paint* a sistemului de operare *Windows 10* pentru ajustarea dimensiunii imaginilor de fundal din fereastra de start și fereastra principală
- *textpro.me* pentru crearea imaginii de fundal pentru fereastra de start
- *Visual Paradigm* – folosit pentru crearea diagramei de clase

1.2 Cerințe specifice

Funcționalități ce au fost realizate in proiect:
- crearea cărților de joc – clasa Carte
- crearea pachetului format din obiecte de tipul Carte – clasa Pachet
- amestecarea într-o ordine aleatoare a cărților din pachet
- crearea unei clase care să rețină cărțile jucătorilor – clasa P_Mana
- afișarea cărților
- crearea unei clase care să rețină scorul
- crearea unei clase care să reprezinte un jucător – clasa Jucator - în această clasă se găsesc obiecte de tipul P_Mana și Scor – în aplicația server au fost create două instanțe de tipul Jucator una pentru server și una pentru client
- distribuirea cărților către jucători
- transmiterea cărților din mâna jucătorului server către masa la selectarea cărții dorite
- golirea mesei
- codificarea unei cărți sub forma unui șir de caractere în aplicația server
- codificarea unei cărți puse pe masă sub forma unui mesaj pentru a fi transmis la aplicația client
- codificarea si transmiterea unei cărți sub forma de mesaj pentru a fi afișată pe masa din aplicația client în aplicația server
- decodificarea unei cărți codificată sub forma unui șir de caractere – atât în aplicația server cât si în aplicația client
- codificarea primelor patru cărți extrase din pachet pentru jucătorul client, sub forma unui șir de caractere pentru a fi transmise prin rețea din aplicația server către aplicația client la începutul jocului
- decodificarea și afișarea primelor 4 cărți la începutul jocului în aplicația client.

2 Amestecarea cărților din pachet

2.1 Descriere

Deoarece proiectul presupune crearea unui joc de cărți importat este ca ordinea cărților din pachet să fie una aleatoare la fiecare rulare a aplicației pentru ca fiecare jucător să aibă șanse egale la extragerea cărților din pachet.Amesctecarea cărților se realizează de fiecare dată la începutul jocului asigurând o ordine diferită la fiecare rulare.

2.2 Fluxul de evenimente

2.2.1 Fluxul de bază

Amestecarea cărților din pachet este realizată de o metodă din clasa Pachet numită *Amesteca()*. În clasa Pachet se află un vector de pointeri către obiecte de tipul Carte – ‚c'. Acest vector este inițializat la început în constructorul implicit cu pointeri către obiecte de tipul carte într-o ordine prestabilită : prima dată pointeri la cărțile care au perechea cu index 0 și numere de la 7 la 14 apoi perechea cu index 1 până la perechea cu index 3.

În proiect se creează un singur pointer către un obiect de tipul Pachet care este membru privat al clasei Joc_S. În cadrul clasei Joc_S a fost creată o metodă *AmestecaPachet()* care apelează metoda *Amesteca()*. În forma fJoc a fost creat un pointer către obiecte de tipul Joc_S pointerul este inițializat în constructorul implicit al formei și tot în constructorul implicit este apelată metoda *AmestecaPachet()* care apelează mai departe metoda *Amesteca()* pentru a asigura amestecarea cărților chiar de la pornirea jocului. Utilizatorul nu trebuie să facă nimic pentru a amesteca cărțile deoarece metoda este apelată in constructorul implicit al formei.

Implementarea metodei *Amesteca()* este următoarea:

- se crează un pointer la un obiect de tipul Pachet *p_temp* cu constructorul implicit având cărțile ordonate așa cum a fost descris mai sus la instanțierea unui obiect cu constructorul implicit
- în cadrul obiectului de tipul Pachet curent este eliberat conținutul vectorului de cărți *c* cu ajutorul funcției *clear()* din biblioteca <vector>
- într-o structură repetitivă de tipul *while* cât timp mai sunt cărți in pachetul nou creat se extrage o carte aleator din pachet și se introduce în vectorul de cărți din pachetul curent
- verificarea numărului de cărți rămase în pachet se face prin intermediul metodei *NrCarti()* care returnează un număr întreg reprezentând numărul de elemente din vectorul de pointeri la obiecte de tipul Carte din clasa Pachet
- extragerea aleatoare a cărților din pachetul *p_temp* se face prin intermediul metodei *Extrage(int ind)* din clasa Pachet care primește ca parametru un număr întreg reprezentând indicele cărții care urmează să fie extrasă din vectorul de cărți din cadrul obiectului de tipul Pachet. În metodăse crează un pointer la Carte *c_temp* care este inițializat cu valoarea pointerului din vectorul de cărți *c* al cărui indice este *ind,* se șterge apoi acestă valoare din vector prin intermediul funcției *erase()* și se returnează pointerul creat la început.
- indicele pe care-l primește metoda *Extrage* este calculat cu ajutorul funcției *rand()* relativ la numărul de cărți rămase în pachet iar pentru a asigura ca la fiecare apel al funcției *rand()* aceasta să dea valori aleatoare diferite s-a folosit funcția *srand()* având ca parametru *time(0)* pentru ca funcția rand să aibă drept "*seed"* timpul sistemului.

În final cărțile din cadrul pachetului curent o să fie amestecate aleator

Amestecarea cărților se realizează o singură dată în aplicația server deoarece doar în aplicația server se regăsește un obiect de tipul Pachet , urmând ca aplicația server să codifice si să transmită mai departe cărțile ce trebuiesc afișate în aplicația client.

2.2.2 Pre-condiţii

Pentru această funcționalitate utilizatorul trebuie doar sa pornească jocul prin apăsarea butonului de Start și funcționalitatea se va realiza automat fără a mai fi nevoie de alte setări inițiale deoarece la crearea ferestrei de joc este apelată și metoda care amestecă cărțile din pachet.

2.2.3 Post-condiţii

La extragerea cărților din pachet și afișarea lor utilizatorul va observa că acestea nu respectă o anumită ordine. Dacă funcționalitatea nu ar fi funcționat corect jucătorii ar fi avut la început cele patru cărți de aceeași pereche (PICA)

3 Codificarea cărților pentru transmiterea prin rețea

3.1 Descriere

Codificarea cărților constă în crearea unui șir de caractere care să reprezinte un obiect de tipul carte pentru a putea fi transmis de la aplicația server la aplicația client și invers și pentru a putea fi decodificată și afișată cartea transmisă.

3.2 Fluxul de evenimente

3.2.1 Fluxul de bază

Codificarea unui obiect de tipul Carte se realizează printr-o metoda a clasei Joc_S numită *CodificaCarte()*. Această metodă primește ca parametru un pointer la obiect de tip Carte *c_temp* si returnează un șir de caractere care va reprezenta codificarea cărții *c_temp*.

Implementarea metodei constă în:

- crearea unui șir de caractere *str* care va reține codificarea cărții
- crearea a două variabile de tip întreg *per_tem* și *num_temp*
- *per_temp* va reține indexul perechii cărții *c_temp* extras cu ajutorul metodei *GetPereche()* , metodă implementată în clasa Carte
- *num_temp* va reține indexul numărului cărții *c_temp* extras cu ajutorul metodei *GetNumar()* , metodă implementată în clasa Carte
- se realizează conversia explicită de tip de la întreg la șir de caractere pentru *per_temp* la fel si pentru *num_temp*, iar cele două șiruri rezultate se concatenează si se adaugă în șirul *str* inițial null
- tot la șirul *str* se mai concatenează si caracterul '_' care reprezintă un separator în ideea în care codificarea rezultată poate face parte dintr-un mesaj care să conțină mai multe codificări de acest fel separate între ele prin '_'
- se returnează apoi șirul rezultat salvat în *str*

3.2.2 Fluxuri alternative

3.2.2.1 < Primul flux alternativ >

În aplicația client codificarea unei cărți se realizează diferit. Deoarece aplicația client a fost gândită mai degrabă ca un "ecran" care să afișeze schimbările transmise de server și să permită utilizatorului să interacționeze cu interfața iar apoi să transmită mai departe schimbările făcute de utilizator sub formă de mesaje pentru a fi procesate in aplicația server cărțile sunt păstrate sub forma șirului de caractere codificat pe baza acestuia fiind afișate doar imaginile cărților. Astfel codificarea cărților in aplicația client se realizează în același moment cu crearea mesajului care urmează să fie transmis către server.

În clasa Joc_S din aplicația client avem membrul *str* de tip String care va reține mesajul ce urmează să fie transmis la server , iar codificarea se realizează prin extragerea codificării deja existente din mesajului transmis de server, mesaj păstrat în clasa Mana.

3.2.3 Pre-condiții

Funcționalitatea se realizează la distribuirea cărților la începutul jocului , se extrag cele patru cărți din pachet ale jucătorului client , iar în timpul extragerii se codifică cărțile extrase și se formează mesajul pentru a fi transmis la client.

Utilizatorul nu are de făcut nici o setare suplimentară , metoda care distribuie cărțile si implicit le codifică sub forma de mesaj este apelată în constructorul formei fJoc

Pentru aplicația client funcționalitatea se realizează la apăsarea unei cărți pentru a fi pusă pe masă. Se codifică cartea sub forma unui mesaj text iar apoi este transmis la server decodificat și afișat pe interfață

3.2.4 Post-condiţii

După realizarea funcționalității utilizatorul client ar trebui să poată vedea cartea care a fost transmisă de la server , iar utilizatorul client ar trebui să poată vedea cartea transmisă de la client.

4 Punerea unei cărți pe masă

4.1 Descriere

Este o funcționalitate importantă care constă în extragerea unei cărți din mâna jucătorului si punerea acesteia pe masă la apăsarea cu cursorul pe imaginea cărții dorite pentru a fi pusă pe masă.

4.2 Fluxul de evenimente

4.2.1 Fluxul de bază

În proiect masa de joc a fost reprezentată sub forma unei clase Masa care are următorii membri: un vector de pointeri de tipul Carte cu dimensiunea de opt elemente *c_masa* deoarece într-o rundă se pot pune pe masă maxim opt cărți , un vector de pointeri de tipul TImage *img* cu dimensiunea de opt elemente folosit pentru a afișa imaginile cărților puse pe masă și o variabila *nrCartiJos* de tip întreg care indică numărul de cărți puse pe masă.

Un obiect de tipul Masa este creat în clasa Joc_S care conține si metodele ce contribuie la realizarea funcționalității de punerea unei cărți pe masa. Tot în clasa Joc_S mai avem și două obiecte de tipul Jucător *j_Server* si *j_Client* ,clasa în care avem un obiect de tipul P_Mana care reține într-un vector de pointeri de tip Carte cele patru cărți ale unui jucător.

Implementarea funcționalității:

- se crează un pointer de tip Carte c_temp
- se extrage din mâna jucătorului cartea care urmează sa fie pusă pe masă pentru jucătorul server sau pentru jucătorul client , extragerea se face prin apelul metodei *PuneCarteJos(int i)* din clasa Jucator care apelează metoda *Extrage(int i)* din clasa P_Mana metodă care extrage din vectorul de cărți din obiectul de tip P_Mana cartea cu indexul i și o returnează
- pentru obiectul de tipul Masa din cadrul clasei Joc_S se apelează metoda *Adauga(Carte *c)*
- Metoda *Adauga(Carte *c)* presupune : verificarea numărului de cărți puse pe masă dacă este mai mic decât 8 se continuă execuția cu adăugarea cărții, în vectorul *c_masa* la indexul dat de *nrCartiJos* se adaugă cartea *c* în vectorul de imagini img la indexul dat de nrCartiJos este adăugată imaginea cărții *c* cu ajutorul metodei *GetIm()* care returnează un pointer de tipul TBitmap reprezentând imaginea cărții *c* se incrementează apoi *nrCartiJos* cu unu

Urmează în secțiunile de mai jos exemplificarea funcționalității atât pentru aplicația client cât și pentru aplicația server

4.2.2 Fluxuri alternative

4.2.2.1 < Primul flux alternativ >

Pentru aplicația server punerea unei cărți jos pe lângă operațiile explicate mai sus mai implică și alte operații. Această funcționalitate se realizează prin intermediul metodei *PuneCarteJosServer(int i)* din clasa Joc_S

Implementarea metodei:

- se execută pașii explicați mai sus cu diferența că metoda *PuneCarteJos(int i)* este apelată pentru obiectul *j_Server*
- clasa Joc_S are ca membru public variabila str de tipul String folosită pentru transmiterea de mesaje prin rețea
- variabila str este inițializată cu șirul “*cm_”
- la variabila str se concatenează codificarea cărții *c_temp* codificată cu ajutorul metodei *CodificaCarte(Carte *c_temp)*

- ultimul caracter din șir care reprezintă separatorul "_" este înlocuit cu '#' pentru a marca sfârșitul mesajului

4.2.2.2 < Al doilea flux alternativ >

În aplicația client punerea unei cărți jos este implementată diferit. Aici nu mai avem clasa Jucator care să aibă membru un obiect de tip P_Mana ci obiectul de tip Mana este creat în clasa Joc_S.

Clasa Masă aici conține doar vectorul de imagini *img[8]* și numărul de cărți maxime. Metoda *Adauga(String str)* primește acum un șir de caractere care reprezintă numele imaginii din folderul de imagini din cadrul proiectului ce urmează să fie încărcată în vectorul de imagini. În folderul de imagini fiecare imagine are numele format din combinația index pereche index carte.

Clasa Mana are un singur membru un șir de caractere care reține codificarea cărților trimisă de server.

Metoda *PuneCarteJos(int i,TImage *timg[])* implementează funcționalitatea astfel:

- crează o variabilă de tipul String str_temp
- crează o variabilă de tipul întreg k inițial 0
- un vector de patru elemente de tipul String st[4]
- în str_temp se salvează șirul de caractere din m_Joc cu ajutorul metodei Arata() din clasa Mana șir care reprezintă cele patru cărți din mâna jucătorului
- se parcurge pe rând fiecare caracter din *str_temp* dacă caracterul este diferit de separator se adaugă în *st[k]* st va reține astfel indecșii perechii și numărului fiecărei cărți din mână ; dacă caracterul curent este egal cu separatorul se incrementează *k*
- se apelează apoi metoda *Adaugă()* din cadrul clasei Masa cu parametrul *st[i]* unde *i* este indexul cărții care vrem să fie afișată
- se codifică apoi în membrul *str* din clasa Joc_S cartea ce se va afișa pe masă sub forma unui mesaj pentru a fi transmis la server ,decodificat și pentru a afișa și acolo cartea , pentru a actualiza cărțile din mâna jucătorului client deoarece atunci când o carte a fost pusă pe masa aceasta trebuie extrasă și din mâna ; codificarea se realizează prin concatenarea la șirul "*cm_" *st[i]* și terminatorul "#"

4.2.2.3 < Al treilea flux alternativ >

În aplicația server mai există încă o implementare a funcționalității atunci când trebuie pusă pe masă o carte de la jucătorul client pe baza mesajului primit de la aplicația client funcționalitate ce se realizează prin intermediul metodei *PuneCarteJosClient(String str)* din cadrul clasei Joc_S.

Implementarea metodei:

- se crează o variabilă de tipul String str_temp
- parametrul str trebuie să conțină un mesaj de forma "*cm_07#"
- se extrage codificarea cărții parcurgând șirul de la indexul 5 până la întâlnirea terminatorului de mesaj si se adaugă în str_temp
- se crează un pointer de tipul Carte c_temp care va reține cartea decodificată
- se apelează metoda DecodifcaCarte(String str) cu str_temp ca parametru metodă care va returna un pointer de tip Carte și care va fi salvat in c_temp
- într-un for de la 0 la 4 se parcurg cărțile din mâna jucătorului client pentru a se extrage cartea ce urmează să fie pusă pe masă ; se crează un pointer de tip Carte care va reține cartea curentă din mâna jucătorului client , se compară cartea curentă cu cartea decodificată (doua cărți sunt egale dacă au aceeași pereche și același număr) când se găsește cartea dorită se execută operațiile de la fluxul de baza cu diferența că *PuneCarteJos(int i)* se va apela pentru obiectul *j_Client*

4.2.2.4 < Al patrulea flux alternativ >

În aplicația client mai există încă o implementare a funcționalității atunci când trebuie pusă pe masă o carte de la jucătorul server pe baza mesajului primit de la aplicația server funcționalitate ce se realizează prin intermediul metodei *PuneCarteJosServer(String str)* din cadrul clasei Joc_S. Metoda apelează metoda *Adauga(String str)* din cadrul clasei Masa care afișează imaginea cărții pe masă

4.2.3 Pre-condiţii

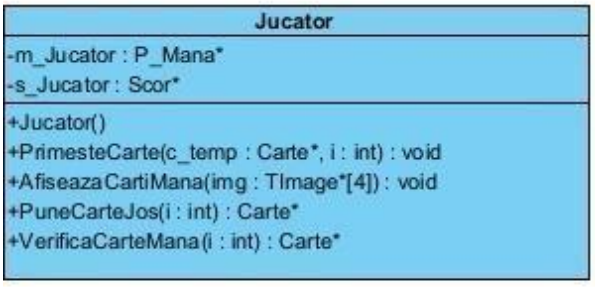
Pentru a porni această funcționalitate utilizatorul trebuie să dea click pe imaginea cărții dorite. Utilizatorul poate pune pe masă o singură dată una dintre cele patru cărți. O problemă ce nu a fost momentan rezolvată este transmiterea cărții puse pe masă dinspre server spre client. Cărțile puse pe masă de către client se pot vedea atât in aplicația server cât și în aplicația client dar cărțile puse de server se văd doar în aplicația server deoarece mesajul nu se transmite către client pentru a fi decodificat și afișat.

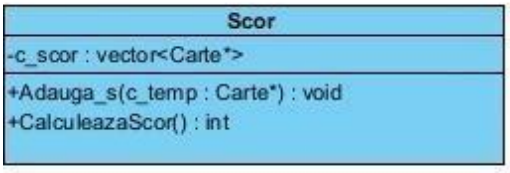
4.2.4 Post-condiţii

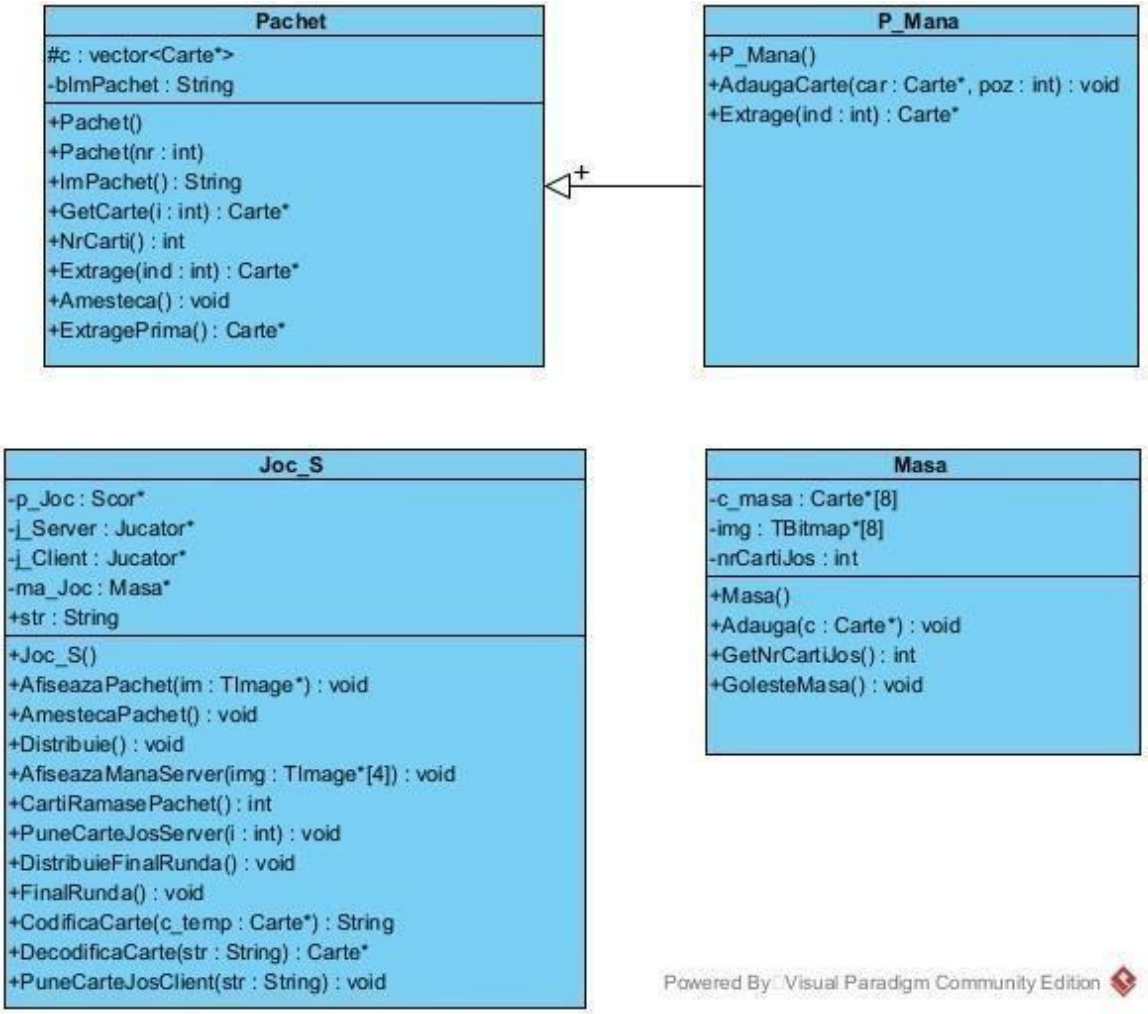
La apăsarea unei cărți utilizatorul va vedea cartea pusă pe masă între imaginile cu cărțile lui și imaginile cu spatele cărților adversarului

5 Implementare

5.1 Diagrama de clase






Pachet
#c : vector<Carte*>
-blmPachet : String
+Pachet()
+Pachet(nr : int)
+ImPachet() : String
+GetCarte(i : int) : Carte*
+NrCarti() : int
+Extrage(ind : int) : Carte*
+Amesteca() : void
+ExtragePrima() : Carte*
P_Mana
+P_Mana()
+AdaugaCarte(car : Carte*, poz : int) : void
+Extrage(ind : int) : Carte*
Joc_S
-p_Joc : Scor*
-j_Server : Jucator*
-j_Client : Jucator*
-ma_Joc : Masa*
+str : String
+Joc_S()
+AfiseazaPachet(im : TImage*) : void
+AmestecaPachet() : void
+Distribuie() : void
+AfiseazaManaServer(img : TImage*[4]) : void
+CartiRamasePachet() : int
+PuneCarteJosServer(i : int) : void
+DistribuieFinalRunda() : void
+FinalRunda() : void
+CodificaCarte(c_temp : Carte*) : String
+DecodificaCarte(str : String) : Carte*
+PuneCarteJosClient(str : String) : void
Masa
-c_masa : Carte*[8]
-img : TBitmap*[8]
-nrCartiJos : int
+Masa()
+Adauga(c : Carte*) : void
+GetNrCartiJos() : int
+GolesteMasa() : void
Powered By Visual Paradigm Community Edition

6 Bibliografie

- c++builder - Dynamic alocation an array of Images in C++ Builder - Stack Overflow
- c++builder - Load images from file (any pc works) C++ Builder - Stack Overflow
- http://www.functionx.com/cppbuilder/
- Embarcadero/IDERA Documentation Wiki
- C++Builder Developer's Guide - RAD Studio (embarcadero.com)
- C++ Builder 6 Developer’s Guide – Satya Say Kolachina
- C++ Builder - VCL Components, Dialogs and Forms - YouTube