UNIVERSITATEA “ȘTEFAN CEL MARE”, SUCEAVA

FACULTATEA DE INGINERIE ELECTRICĂ ȘI ȘTIINȚA CALCULATOARELOR

SPECIALIZAREA CALCULATOARE

PROIECT DISCIPLINA POO

Autor: Maxemiuc Alina

Grupa: 3122B

An de studiu: II

# TEMĂ PROIECT

## Tema și motivația alegerii

Tema acestui proiect constă în crearea unui joc de șah. Voi folosi limbajul de programare C++, unde voi dori implementarea a cât mai multor noțiuni de programare orientată pe obiecte.

Jocul de șah a fost destul de popular în anii precedenți, chiar dacă mai există lume care preferă acest joc, procentul oamenilor care chiar încă o fac este mult mai mic în raport cu anii anteriori. Deși este mult mai “palpitant” să joci cu o altă persoană, pe o tablă reală de joc, ținând cont de secolul în care trăim, din ce în ce mai multe persoane preferă să își petreacă timpul liber în fața unui calculator. De aceea tema aleasă de mine constă în crearea unui joc de șah în C++.

Motivația alegerii acestei teme a fost dorința de a aprofunda limbajul de programare C++.

**Cuprins**

Contents

[TEMA PROIECT 2](#_Toc105281430)

[Tema si motivatia alegerii 2](#_Toc105281431)

[Capitolul 1 4](#_Toc105281432)

[1.Elemente teoretice 4](#_Toc105281433)

[1.1.Descrierea problemei. Reguli de joc 4](#_Toc105281434)

[1.2. Mutarea pieselor 5](#_Toc105281435)

[1.3. Abordarea teoretica a problemei 5](#_Toc105281436)

[1.4. Elemente specifice POO 6](#_Toc105281437)

[Capitolul 2 6](#_Toc105281438)

[2.Implementare 6](#_Toc105281439)

[2.1.Tehnologii folosite 6](#_Toc105281440)

[2.2.Clasele folosite 7](#_Toc105281441)

[Capitolul 3 10](#_Toc105281442)

[3.Analiza solutiei implementate/Manual de utilizare 10](#_Toc105281443)

[Capitolul 4 11](#_Toc105281444)

[4. Concluzii 11](#_Toc105281445)

[Capitolul 5 11](#_Toc105281446)

[5.Bibliografie 11](#_Toc105281447)

# Capitolul 1

## 1.Elemente teoretice

### 1.1.Descrierea problemei. Reguli de joc

Pentru a începe descrierea problemei, în primul rând, trebuie să ne asigurăm că regulile de joc sunt cunoscute tuturor.

Pasul 1 al jocului, ar fi să ne asigurăm că tabla de joc este pusă corespunzator, mai exact aceasta trebuie sa fie poziționată astfel încât fiecare jucător să aibe un pătrat de culoare albă ( sau culoare deschisă, depinde de culorile tablei de joc) în partea de dreapta jos.

Următorul pas face referire la modul de așezare al pieselor pe tablă. Piesele sunt de două culori (albe și negre), 16 pentru fiecare jucător. Cel mai ușor de așezat sunt pionii, aceștia ocupă al doilea rând/linie a tablei de joc. Pe primul rând vom așeza mai întâi turnurile(turele) (în colțurile tablei), urmate de cei doi cai, cei doi nebuni și regina (regina se așează de fiecare dată pe propria ei culoare, regina albă se va așeza pe alb, iar cea neagră pe un pătrat de culoare neagră), iar în cele din urmă vom așeza regele pe pătratul rămâs.



Scopul acestui proiect este de a crea un algoritm care ne poate permite folosirea jocului de șah, dar mai ales folosirea de programare orientată pe obiecte (POO).

### 1.2. Mutarea pieselor

Toate piesele jocului se mută diferit. Piesele nu se pot muta trecând prin alte piese și nu se pot muta pe un pătrat în care se află deja una dintre piesele proprii. Ele pot și mutate și pot lua locul unei piese de a adversarului care este capturată/ “mâncată”. De preferabil că piesele să fie mutate în așa fel încât să fie posibilă capturarea unei piese adverse.

Vorbind acum despre piesele folosite în joc, acestea sunt 16 la număr pentru fiecare dintre cele două culori: 8 pioni, 2 turnuri/ture, 2 nebuni, 2 caii, un rege și o regină.

Regele, cea mai importantă piesă, dar totuși și cea mai slabă, se poate muta doar câte un pătrat în orice direcție – în sus, în jos, în lateral și în diagonală. Acesta nu poate niciodată muta din “șah” – unde ar putea fi capturat. Dacă regele este atacat de vreo piesă se numește “șah”.

Regina este cea mai puternică piesă a jocului. Ea se poate muta în orice direcție -înainte, înapoi, lateral sau în diagonală – atât timp cât nu trece prin oricare dintre piesele proprii.

Turnul/Tura poate muta înainte, înapoi și în lateral, oricât de departe se dorește.

Nebunul se poate muta doar în diagonală oricât de departe se dorește. Fiecare nebun începe de pe o culoare (închisă sau deschisă) și trebuie să râmănă mereu pe acea culoare.

Deși am precizat anterior că piesele nu se pot muta trecând prin alte piese, Calul este singura piesa a jocului care poate sări peste alte piese. Acesta se poate muta doar în formă de “L”, adică merge două pătrate într-o direcție și apoi încă unul la un unghi de 90.

Ultimele piese despre care nu am vorbit încă sunt Pionii. Aceștia sunt mai diferiți față de celelalte piese deoarece se pot muta înainte doar un pătrat, dar pot captura pe diagonală. Pionii pot fi mutați cu două pătrate înainte doar la prima mutare și pot captura doar în pătratul din diagonala din fața lor, deoacere aceștia nu se pot muta înapoi. Dacă există o altă piesă direct în fața unui pion, el nu poate sări sau captura acea piesa.

### 1.3. Abordarea teoretica a problemei

În proiectul ales voi defini clasa “Piese” și voi crea o matrice care să rețină cele 32 de piese de pe tabla de joc. Caracteristicile piesei sunt: culoarea, formă și denumirea specifică, poziția inițială și modul în care poate mânca respectiva piesă, iar pentru fiecare piesă în parte se vor crea clase. Denumirea specifică a fiecărei piese se face după simbol, tip și culoare, poziția piesei de pe tablă va fi reținută de două variabile întregi care se referă la poziția piesei pe orizontală și poziția piesei pe verticală.

Jocul se va afișa în consolă, în fiecare dintre clasele făcute găsim funcții de genul ValideazaMutare, ValiteazaAtac, Atac,Muta, plus funcții care vor analiza problema ocupării pozițiilor și capturării pieselor.

### 1.4. Elemente specifice POO

Programarea orientată pe obiecte (POO) este o paradigmă de programare care se axează pe idea de încapsulare, de grupare a datelor și a codului într-o singură structură. Polimorfismul este un alt concept important asociat cu programarea orientată pe obiecte, concept care permite abstractizări printr-o descriere conceptuală mai simpla a soluției. Aceste legături vor reflecta relațiile ce se stabilesc între clasele și obiectele problemei. Moștenirea reprezintă o relație statică care face legatura dintre o clasă cu o alta, astfel încât clasa rezultată să poata prelua toate atributele clasei sau claselor pe care o/le moșteneste. Moștenirea este folosită pentru a reutiliza codul, acesta constituind și un avantaj al moștenirii. Clasa derivată (cea care moșteneste atributele altei clase, denumită astfel clasa de bază) va implementa doar metodele ei specifice, putând utiliza metodele de bază de la clasa principală.

# Capitolul 2

## 2.Implementare

În cele ce urmează se va găsi o scurtă prezentare a proiectului ales, a modului de implementare a noțiunilor.

### 2.1.Tehnologii folosite

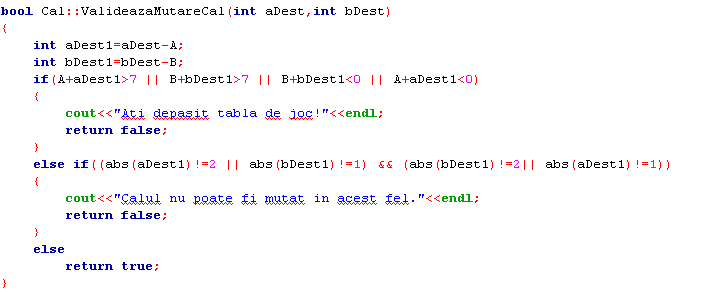
În realizarea proiectului am folosit limbajul de programare Code::Blocks 20.03. Acesta este un IDE gratuit care acceptă mai multe compilatoare, cum ar fi GCC, MinGW, Digital Mars, Microsoft Visual C++, compilatorul Intel C++ și altele. Deși IDE-ul a fost proiectat pentru limbajul C++, există suport și pentru alte limbaje, inclusiv Fortran și D. IDE-ul oferă evidențierea sintaxei și plierea codului, completarea codului C++, browser de clasă, un editor hexadecimal și multe alte utilitare. Debugger-ul are suport complet pentru punctele de întrerupere și permite utilizatorului să își depaneze programul având acces la simbolul funcției locale și afișarea argumentelor, stiva de apeluri, registrele CPU (procesorul) și interfața GNU Debugger. Code::Blocks utilizează un sistem de construcție personalizat, care stochează informațiile în fișiere de proiect bazate pe XML. Opțional, poate utiliza fișiere de tip makefile externe, ceea ce simplifică interfața cu proiectele care utilizează sistemele de construcție GNU sau [qmake](https://en.wikipedia.org/wiki/Qmake) .

Conceptele utilizate pe scară largă în dezvoltarea de software sunt polimorfismul și moștenirea (concepte majore în programarea orientată pe obiecte), diferența dintre acestea două în POO este că polimorfismul este o interfața comună pentru mai multe forme, fiind implementat în metode, iar moștenirea este de a crea o nouă clasă folosind proprietăți și metode ale unei clase existente.

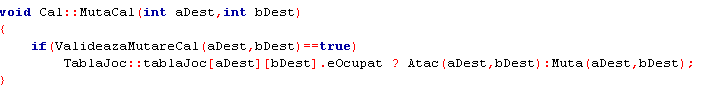
### 2.2.Clasele folosite

În primul rând trebuie să vorbim despre tema aleasa, aceasta fiind reprezentata de crearea unui joc în consola C++ a șahului. În rezolvarea temei am decis să creez clase pentru fiecare piesă în parte, care să derive din clasa principală Piese în care se vor găsi funcții precum Muta, Atac, Cauta, ValideazaMutare, ValideazaAtac, IncearcaMutare. Clasele care folosesc doar funcțiile din clasa principală Piese sunt Nebun, Rege, Regina, Tura, iar clasele Cal PionAlb si PionNegru mai au și alte funcții. Deci apare noțiunea de moștenire și polimorfism între clasele create.

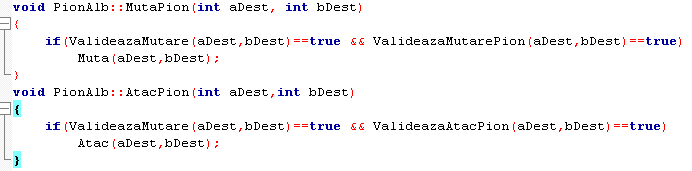
În Sah, calul se mută în forma unui “L” și a trebuit să explicităm acest lucru folosind funcțiia ValideazaMutareCal. Ca să ne fie mai ușor lucrăm cu sistemul de axe xOy (sau cum am notat mai jos aDest vine de la poziția x inițială, iar aDest1 reprezintă poziția finală a lui x, bDest vine de la poziția y inițiala, iar bDest1 de la poziția finală a lui y; A si B sunt indexi). “abs” este o funcție predefinită în C++ și returnează valoarea absolută a unui întreg, în cazul nostru verificăm să putem face mutarea calului cu 2 pătrățele în sus sau jos și doar un patrat în stânga sau dreapta.



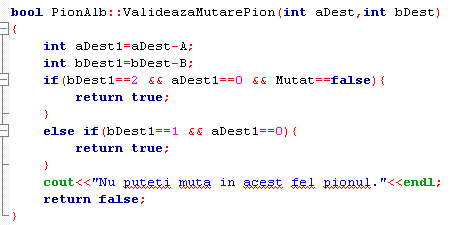
În ceea ce privește și funcția de mutare a acestei piese, va trebui mai întâi să vedem dacă mutarea acestuia se poate face cum dorim prin apelarea funcției prezentată anterior, ca mai apoi să “arătăm” programului unde dorim să efectuăm mutarea, adică pe tabla de joc (există clasă creată numită TablaJoc). Vom verifica dacă locul pe care dorim să ne mutăm este liber sau nu, dacă se află o piesă de altă culoare vom ataca, iar în caz că pătratul în cauză este liber, ne vom muta doar.



Pionii sunt de două culori,astfel am ales să fac două clase separate care să moștenescă clasa principală Piese, una din ele va fi PionAlb, iar cealalată PionNegru. Asemănător ca la clasa Cal, clasa PionAlb are funcții care vor asigura mutarea, atacul, validarea mutării și validarea atacului pionilor. Funcțiile de atac și mutare vor fi folosite din clasa Piese, deoarece avem nevoie de ele în mutarea și atacul pionilor, după cum se observă și mai jos (ValideazaMutare aparține tot clasei Piese):

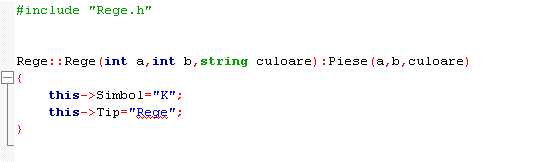


Clasa PionAlb conține funcțiile ValideazaMutarePion și ValideazaAtacPion. Asemănător clasei Cal, și în interiorul acestei clasei ne vom folosi de sistemul de coordonate xOy ( aDest vine de la poziția x inițială, iar aDest1 reprezintă poziția finală a lui x, bDest vine de la poziția y inițială, iar bDest1 de la poziția finală a lui y; A și B sunt indexi ). Deoarece la început pionul are voie să se mute cu două căsuțe mai în față, am folosit bDest1==2.



La clasa PionNegru este asemănător ca la cea numită PionAlb.

În ceea ce privesc restul pieselor, clasele speciale pentru acestea conțin doar constructorul care ne dă tipul și modul în care va fi simbolizată în consolă. În interiorul constructorului am folosit “this->” (un pointer care ne permite să utilizăm membrii altei clase, în cazul nostru să ne folosim de două (variabile) șiruri de caractere care ne arată simbolul și tipul piesei folosite în clasa Piese):

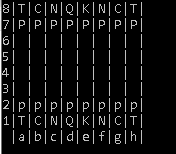


# Capitolul 3

## 3.Analiza soluției implementate/Manual de utilizare

În acest capitol voi încerca să acopăr orice neclaritate în vederea folosirii jocului creat. O primă întrebare la care ar trebui să răspund este referitoare la introducerea datelor în consolă și rezultatul după „aplicarea” acestora.

Tabela tablei de șah este vizualizata ca o matrice de 8x8, având linii de la a la h și coloane de la 1 la 8.



Pentru mutarea pieselor vom folosi formatul în care vom alege mai întâi linia și apoi coloana, ca in exemplul ce urmează:



Din câte se poate observa, pionul p (pion alb, în șah albul mută de obicei primul) s-a deplasat din poziția sa inițială în cea dorită de noi.

# Capitolul 4

## 4. Concluzii

Jocul de șah este un joc de gândire, perspicacitate și răbdare. Regulile acestui joc nu sunt neapărat cele mai complicate reguli, dar atenția trebuie să fie sporită. Codul care face posibilă jucarea sahului, n-a fost tocmai cel mai ușor de implementat, a fost nevoie de înțelegerea jocului în totalitate.

# Capitolul 5

## 5.Bibliografie

1. https://codereview.stackexchange.com/

2. https://www.blogdeit.ro/cancel-save-changes-cele-4-principii-ale-oop-abstractizare-incapsulare-mostenire-polimorfism/

3. <https://wikipedia>.com/