UNIVERSITATEA "ȘTEFAN CEL MARE", SUCEAVA
FACULTATEA DE INGINERIE ELECTRICĂ ȘI ȘTIINȚA CALCULATOARELOR
SPECIALIZAREA CALCULATOARE

PROIECT DISCIPLINA POO

[Jocul X și O împotriva calculatorului în C++]

Autor

CIOBAN I. DANIEL

SUCEAVA . 2024

TEMA PROIECT

TEMA ȘI MOTIVAȚIA ALEGERII

Tema proiectului constă în realizarea jocului X și O împotriva calculatorului. Aceasta constă într-un joc de tip X și O în care jucătorul sau utilizatorul va concura împotriva calculatorului (un adversar virtual implementat printr-un algoritm) pentru a forma o linie continuă pe orizontală, diagonală sau verticală formată din trei simboluri ale utilizatorului (acest simbol poate fi X sau O, lucru ce depinde dacă utilizatorul uman sau calculatorul începe primul să completeze tabla de joc).

Motivul alegerii acestei teme constă în dorința mea de a-mi dezvolta abilitățile de programare prin construirea unui joc simplu bazat pe tehnica de programare orientată pe obiecte, dar și prin implementarea unui algoritm ce permite funcționarea unui adversar virtual (implementarea unei inteligențe virtuale de nivel foarte scăzut).

CUPRINS

Cuprins

CAPITOLUL I

1. ELEMENTE TEORETICE

1.1. DESCRIEREA PROBLEMEI

X și O (în română) sau tic-tac-toe (în engleză americană) este un joc de hârtie și creion pentru doi jucători care marchează pe rând spațiile într-o grilă de 3 pe 3 cu X sau O. Jucătorul care reușește să plaseze trei dintre semnele lor într-un rând continuu orizontal, vertical sau diagonal este câștigător.

Astfel aplicația dorită necesită următoarele implementări:

• Prezentarea grafică a jocului (tablă, simboluri) și a meniurilor (text, opțiuni/butoane) ;

• Preluarea datelor de intrare (input-urilor) de la tastatură;

• Prelucrarea datelor de intrare și actualizarea stării jocului ;

• Procesarea stării jocului (începere, terminare, ieșire);

Aplicația trebuie să ofere utilizatorului control asupra începerii jocului, opririi sau a terminării aplicației, astfel sunt necesare meniuri pentru aceste operații.

Având în vedere jocul în sine, putem stabili elementele principale care definesc jocul X și O. Prin urmare în componența jocului avem: o tablă, două simboluri. Aceste elemente au un rol semnificativ și bine determinat, astfel:

- Tabla:
  - reprezintă locul care va fi completată cu simbolurile jucătorilor pe parcursul jocului;
  - este sub formă matriceală. Dimensiunea este bine stabilită (trei linii a câte trei coloane fiecare).
- Simbolul:
  - reprezintă simbolul unui jucător (X sau O) și este atribuit la începerea fiecărui joc. Dacă jucătorul este primul atunci primește semnul X. Dacă este al doilea primește semnul O;
  - semnele nu au voie să se suprapună, fiecare ocupând câte o poziție pe tablă stabilită de datele de intrare ale jucătorului.
- Mișcările valide:
  - reprezintă mișcările posibile valide care rămân pe parcursul jocului. Cu cât tabla de joc este populată cu atât sunt mai puține mișcări valide posibile;

- sunt folosite pentru a genera mișcările calculatorului.

Mai multe detalii despre implementare și abordare sunt prezentate în subcapitolul următor.

1.2. ABORDAREA TEORETICĂ A PROBLEMEI

Privind problema din punct de vedere al datelor, putem asocia componentele principale enumerate în subcapitolul anterior cu: matrice, vectori , poziții și valori de tip caracter.

Prin urmare putem asocia elementele astfel:

- Tabla este asociată cu o matrice, unde valorile 0 reprezintă poziție liberă, 1 reprezintă jucătorul 1 și 2 reprezintă jucătorul 2. Un exemplu de tablă din timpul unui joc ar fi:

1 2 0

1 1 0

2 0 2

Din această matrice se observă că pozițile libere sunt linia 1 coloana 3, linia 2 coloana 3 și linia 3 coloana 2. Pozițile jucătorului 1 sunt linia 1 coloana 1, linia 2 coloana 2 și coloana 3, iar cele ale jucătorului 2 sunt linia 1 coloana 2, linia 3 coloana 1 și coloana 3. Astfel se pot vedea pozițile în care jucătorii își pot plasa simbolul, pozițile în care se va afișa simbolul jucătorului 1 și pozițile în care se va afișa simbolul jucătorului 2.

Astfel, matricea tablei este folosită pentru a salva pozițile mișcărilor din timpul jocului, pentru a verifica dacă o mișcare este validă și poate fi folosită la afișarea tablei grafice.

- Simbolul este o variabilă de tip char și memorează simbolul jucătorilor din timpul unui joc. De exemplu simbolul jucătorului 1 poate fi X și simbolul jucătorului 2 poate fi O, sau invers. Însă niciodată nu vor fi același simbol.

  Ele vor fi folosite în timpul afișării tablei de joc după valorile salvate în matricea tablei.

Pentru a implementa meniurile se folosesc expresii de tip switch case pentru a verifica input-ul utilizatorului uman. Dacă input-ul corespunde unei opțiuni din acel meniu atunci va/vor fi apelată/e funcția/funcțiile și instrucțiunile corespunzătoare.

1.3. ELEMENTE SPECIFICE POO

Pentru problema descrisă anterior, în rezolvarea ei se vor crea clase si metode specifice pentru: „Game”, „Table” , „Player” și „Computer”. Pe lângă metodele specifice fiecărei clase, se vor folosi și constructorii și deconstructorii claselor.

Folosind o librărie pentru partea grafică, se vor folosi și clase deja făcute, de exemplu clasa „Vector2f”, care reprezintă un vector de lungime 2, cu parametrii „x” și „y”, de tipul „float”, clasa „Texture” care reprezintă o imagine, citită din memorie sau creată chiar din cod. Clasa „Sprite” este o clasă care se folosește de clasa „Texture”, poate chiar selecta doar porțiune din textură pentru afișare, obiectele acestei clase sunt afișate. Clasa „RenderWindow” reprezintă fereastra pe care afișăm sprite-urile, etc.

1. Clasa „Game”.

```
class Game
{
    private:
    Table* tabla;
    Player* jucator1;
    Computer* jucator2;
    char simbolJucator1;
    char simbolJucator2;

public:
    Game(); // Constructorul clasei
    void ruleaza(); // Meniul principal al jocului
    void afiseazaMeniuPrincipal(); // Afiseaza meniul principal
    void afiseazaMeniuPrincipal2(); // Afiseaza meniul la terminarea unui joc
    void incepeJocNou(); // Afiseaza meniul pentru a alege care jucator incepe primul
    void setareJocNou(); // Creaza o noua instanta a tablei si apeleaza incepeJocNou()
    void ruleazaJoc1(); // Ruleaza jocul cand jucatorul uman este primul
    void ruleazaJoc2(); // Ruleaza jocul cand computerul este primul jucator
    void afiseazaTabla(); // Afiseaza tabla din timpul jocului cu cateva detalii in plus
    void artaAscii(); // Afiseaza arta ascii din meniul principal
    void setari(); // Afiseaza meniul pentru setari si permite introducerea optiunii
    void setariCuloare(); // Permite alegerea temei jocului dintre cele enumerate
    void info(); // Afiseaza informatii despre proiect
    void castig(); // Ce se afiseaza in caz ca jucatorul uman a castigat
    void pierdere(); // Ce se afiseaza in caz ca jucatorul uman a pierdut
    void egalitate(); // Ce se afiseaza in caz de egalitate
    void centrare(std::string text); // Centreaza textul in consola
    void help(); // Afiseaza informatii despre cum se joaca jocul si cum se utilizeaza aplicatia
    ~Game(); // Destructorul clasei
};
```

Clasa „Game” avem atributele următoare:

- tabla – reprezintă instanța obiectului de tip „Table”;
- jucator1 – reprezintă instanța obiectului de tip „Player”;
- jucator2 – reprezintă instanța obiectului de tip „Computer”;
- simbolJucator1 – este de tip char și salvează simbolul jucătorului 1;
- simbolJucator2 – este de tip char și salvează simbolul jucătorului 2.

Metoda *rulează()* reprezintă meniul principal al aplicației. Metodele *afiseazaMeniuPrincipal(), afiseazaMeniuPrincipal2(), afiseazaTabla(), artaAscii(), info(), afiseazaTabla()* și *help()* afisează datele sau elementele grafice corespunzătoare. Metoda *setari()* și *setariCuloare()* afișează setările corespunzătoare și permite selectarea anumitor opțiuni. Metodele *ruleazaJoc1()* și *ruleazaJoc2()* sunt folosite pentru a rula jocul, pentru a verifica datele introduse, pentru a salva datele în tabel și pentru a verifica starea jocului. Metodele *castig(), pierdere()* și *egalitate()* sunt apelate la finalul unui joc după tipul de sfârșit al jocului și oferă anumite opțiuni utilizatorului. Metoda *centrare(std::string text)* este folosită pentru a afișa orice șir text la mijlocul consolei.

2. Clasa „Table”.

```
class Table
{
private:
    std::vector<std::vector<int>> tablaGoala;
    std::vector<std::vector<int>> tabla;

public:
    std::vector<int> miscariValide;
    Table(); // Initializeaza o tabla goala. Constructorul clasei
    Table (const Table& old); // Reseteaza tabla de joc. Constructorul cu argument
    std::vector<std::vector<int>> getTabla() const; // Returneaza tabla de joc
    bool esteMutareValida(int linie, int coloana); // Verifica daca mutarea este valida
    void plaseazaMutare(int linie, int coloana, int jucator); // Plaseaza in tabla de joc o miscare
    bool esteCastig(int jucator); // Verifica daca este castig
    bool esteEgalitate(); // Verifica daca este egalitate
    ~Table(); // Destructorul clasei
};
```

Clasa „Table” are următoarele atribute:

- tablaGoala – este o matrice goală ce este utilizată pentru a reseta matricea *tabla;*
- tabla – este matricea utilizată în timpul jocului pentru a salva mișcările jucătorilor;
- miscariValide – este vectorul în care sunt salvate mișcările valide pe care calculatorul le utilizează pentru a genera o mișcare validă.

În constructorul *Table()* sunt inițializate pentru prima dată matricea tablei și vectorul de mișcări valide. Constructorul *Table(const Table& old)* este utilizat la reînceperea unui joc nou după încheierea altui joc. Metoda *getTabla()* returnează matricea tablei ca o constantă. Metoda *esteMutareValida(int linie, int coloana)* verifică dacă coordonatele primite reprezintă o mișcare validă și returnează *true* dacă sunt sau *false* dacă nu sunt. Metoda *plaseazaMutare(int linie, int coloana, int jucator)* este folosită pentru a popula matricea tabla cu mișcările valide introduse de fiecare jucător în parte. Metoda *esteCastig(int jucator)* verifică dacă condiția de câștig a fost îndeplinită pentru jucătorul specificat ca parametru de intrare, și returnează *true* sau *false*. Metoda *esteEgalitate()* verifică dacă condiția de egalitate a fost îndeplinită și returnează *true* sau *false*.

3. Clasa „Player”.

```
class Player
{
    public:
        Player() {}; // Constructorul clasei
        std::pair<int, int> introducereMutare(); // Permite jucatorului uman sa introduca o mutare
        ~Player() {}; // Destructorul clasei
        void centru(std::string text); // Centreaza textul in consola
};
```

Metoda *introducereMutare()* este metoda folosită pentru ca utilizatorul uman să poată introduce de la tastatură coordonatele poziției dorite sau pentru a putea utiliza opțiunile „Meniu principal” și „Setari” în timpul jocului. Metoda *centru(std::string text)* este folosită pentru a centra textul afișat din metoda *introducereMutare()*. Această metodă va fi explicată în subcapitolul 2.3.

4. Clasa „Computer”.

```
class Computer
{

public:
    Computer();
    std::pair<int, int> generareMutare();
    std::pair<int, int> generareMutare2(std::vector<int>& miscariValide);
    ~Computer() {};
};
```

Metoda *generareMutare()* generează și returnează un set de poziții pentru calculator. Însă datorită faptului că nu utilizează informații în timp real despre tabla de joc există posibilitatea de a crea delay-uri în program încercând să genereze un set de coordonate valid. De acea nu este utilizată în aplicație. Este utilizată, în schimb, metoda *generareMutare2(miscariValide)*. Această metodă se folosește de vectorul de mișcări valide pentru a genera o mișcare care va fi validă.

CAPITOLUL II

2. IMPLEMENTARE

Modul de rezolvare a acestei probleme este prin crearea unei clase principale „Game”, care trebuie instanțiată doar o dată în „main”. Această clasă este folosită pentru a afișa și a gestiona orice acțiune care va avea loc pe parcursul rulării aplicației. Acest lucru include afișarea meniurilor, gestionarea stării jocului, aplicarea setărilor, introducerea datelor de către utilizator.

Clasa „Game” se folosește de obiecte ale claselor „Table”, „Player” și „Computer” pentru a gestiona întreaga funcționare a aplicației. Aceste obiecte sunt instanțiate în timpul utilizării aplicației. Mai exact la începerea unui joc nou și la pornirea aplicației.

La pornirea aplicației clasa „Game” afișează meniul principal cu opțiunile „Incepe joc nou”, „Setari”, „Paraseste aplicatia” și opțiunea pentru informații. Tot aici sunt instanțiate obiectele de tip „Player” și „Computer”. Dacă se selectează a doua opțiune atunci este afișat meniul cu setări. În meniul cu setări sunt opțiunile de modificare a tematicii aplicației, de afișare a informațiilor despre proiect și de întoarcere la meniul principal. Dacă se selectează a treia optiune din meniul principal atunci aplicația se oprește. Dacă se selectează opțiunea pentru informații din meniul principal atunci se afișează acele informații și utilizatorul se poate întoarce la meniul principal la introducerea unei taste. Dacă însă se selectează opțiunea de începere a unui joc nou atunci este afișat meniul pentru setarea începerii jocului. Acest meniu conține opțiunea ca utilizatorul să înceapă primul, opțiunea ca primul să fie calculatorul și opțiunea de întoarcere în meniul principal.

Când jocul rulează mai întâi este instanțiat obiectul de tip „Table” și datele sunt setate corespunzător (matricea tablei este goală și vectorul mișcărilor posibile este populat, simbolurile fiecărui jucător este setat ținând cont de cine va începe primul să introducă mișcarea).

După ce s-au instanțiat obiectele jocului, se populează matricea tablei cu pozițiile introduse de fiecare jucător pe rând. Pentru introducerea mișcărilor se folosesc funcțiile corespunzătoare ale obiectelor de tip „Player” și „Computer”. Obiectul de tip „Player” solicită utilizatorului uman să introducă poziția, iar obiectul de tip „Computer” generează câte o poziție validă folosindu-se de vectorul mișcărilor valide.

Când se termină jocul se afișează mesajul corespunzător stării de sfârșit (egalitate, câștig sau pierdere) și se oferă utilizatorului aceleași opțiuni ca în meniul principal, fără opțiunea pentru informații. În schimb i se oferă în schimb opțiunea de revenire în meniul principal.

2.1. TEHNOLOGII FOLOSITE

Pentru acest proiect s-a folosit limbajul de programare C++ și un mediu de lucru care suportă limbajul de programare C++, și anume Code::Blocks versiunea 20.03 și compilatorul GNU GCC. Aplicația este de tip consolă și este compatibilă cu windows.

2.2. DIAGRAMA DE CLASE, SCHEMA BLOC, WORKFLOW

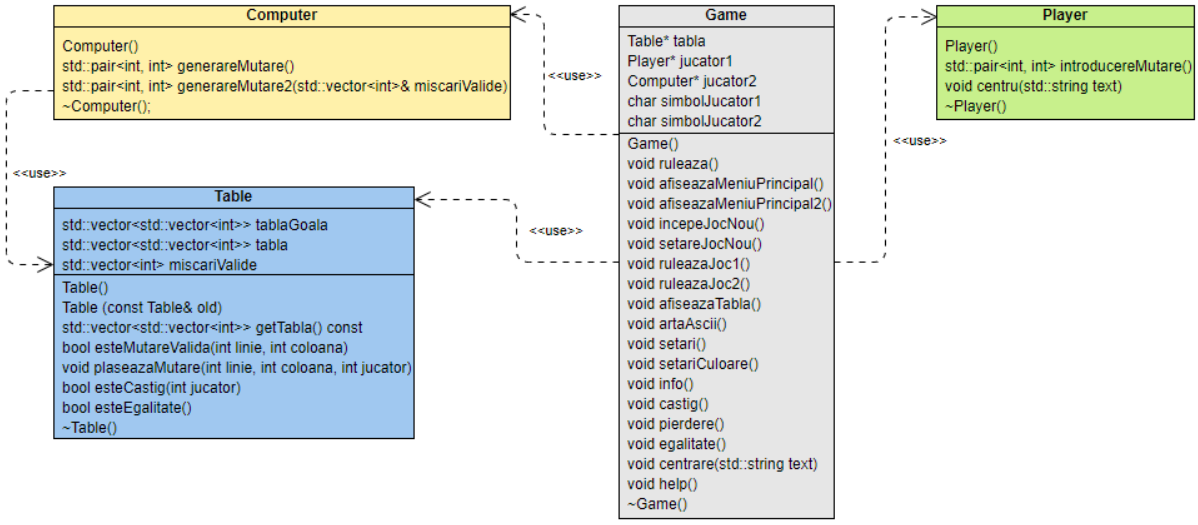

Fig. 1. Diagrama UML de clase a aplicației

2.3. AFIȘAREA ÎN CONSOLĂ ȘI INTRODUCEREA DATELOR

Elementele grafice ale aplicației consă în afișarea pe fiecare rând a câte unui șir de caractere. Însă, pentru a afișa aceste șiruri în mijlocul consolei a fost necesară implementarea unei metode pentru clasele „Game” și „Player” care să permită acest lucru. Pentru clasa „Game” a fost implementată metoda *centrare(std::string text)* și pentru clasa „Player” metoda *centru(std::string text)*. Însă ambele metode au același cod. Pentru a putea implementa aceste metode s-a folosit API-ul Windows introducând în proiect fișierul header *windows.h.* Codul metodelor este următorul:

```
void Game::centrare(std::string text) // Centreaza textul in consola
{
    // Va centra textul doar daca lungimea textului e mai mica decat lungimea consolei
    // Altfel va afisa textul asa cum e
    HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
    PCONSOLE_SCREEN_BUFFER_INFO informatiiEcran = new CONSOLE_SCREEN_BUFFER_INFO();
    GetConsoleScreenBufferInfo(hConsole, informatiiEcran);
    COORD dimensiuneNoua = informatiiEcran->dwSize;
    if (dimensiuneNoua.X > text.size())
    {
        int positieNoua = ((dimensiuneNoua.X - text.size()) / 2);
        for (int i = 0; i < positieNoua; i++) std::cout << " ";
    }
    std::cout << text << std::endl;
}

void Player::centru(std::string text) // Centreaza textul in consola
{
    // Va centra textul doar daca lungimea textului e mai mica decat lungimea consolei
    // Altfel va afisa textul asa cum e
    HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
    PCONSOLE_SCREEN_BUFFER_INFO informatiiEcran = new CONSOLE_SCREEN_BUFFER_INFO();
    GetConsoleScreenBufferInfo(hConsole, informatiiEcran);
    COORD dimensiuneNoua = informatiiEcran->dwSize;
    if (dimensiuneNoua.X > text.size())
    {
        int positieNoua = ((dimensiuneNoua.X - text.size()) / 2);
        for (int i = 0; i < positieNoua; i++) std::cout << " ";
    }
    std::cout << text << std::endl;
}
```

- HANDLE este un tip definit de Windows pentru a reprezenta un handle (un identificator unic) pentru resursele de sistem.
- GetStdHandle(STD_OUTPUT_HANDLE) este o funcție care returnează un handle către fluxul standard de ieșire (de obicei, ecranul consolei).
- hConsole stochează acest handle, care va fi folosit pentru a accesa și manipula ecranul consolei.
- PCONSOLE_SCREEN_BUFFER_INFO este un pointer către o structură CONSOLE_SCREEN_BUFFER_INFO.
- new CONSOLE_SCREEN_BUFFER_INFO() alocă dinamic o structură CONSOLE_SCREEN_BUFFER_INFO și returnează un pointer la aceasta.
- informatiiEcran stochează acest pointer. Structura CONSOLE_SCREEN_BUFFER_INFO conține informații despre ecranul consolei, cum ar fi dimensiunea bufferului, dimensiunea ferestrei consolei, etc.
- GetConsoleScreenBufferInfo este o funcție care obține informații despre bufferul ecranului consolei asociat cu hConsole și le stochează în structura indicată de informatiiEcran.
- hConsole este handle-ul către consola obținut anterior.
- informatiiEcran este pointerul către structura unde vor fi stocate aceste informații.
- COORD este o structură definită de Windows pentru a reprezenta coordonatele X și Y (de obicei, dimensiunea ecranului în caractere).

- informatiiEcran->dwSize accesează dimensiunea bufferului ecranului, care este un membru al structurii CONSOLE_SCREEN_BUFFER_INFO.
- dimensiuneNoua stochează această dimensiune.
- dimensiuneNoua.X este lățimea ecranului consolei în caractere.
- text.size() returnează lungimea textului ce trebuie centrat.
- Dacă lățimea ecranului (dimensiuneNoua.X) este mai mare decât lungimea textului (text.size()), se calculează numărul de spații necesare pentru a centra textul.
- positieNoua calculează numărul de spații necesare pentru a centra textul. Acest număr este obținut prin scăderea lungimii textului din lățimea ecranului și împărțirea rezultatului la 2.
- Bucla for afișează spațiile necesare pentru a centra textul.
- Afișează textul urmat de o linie nouă (std::endl).

În concluzie, acest cod:

- Obține un handle către consola curentă.
- Obține informațiile despre dimensiunea bufferului ecranului consolei.
- Calculează numărul de spații necesare pentru a centra un text.
- Afișează spațiile și apoi textul centrat pe ecranul consolei.

Bufferul este o zonă de memorie temporară folosită pentru a stoca date în timpul transferului între două locații.

```
HANDLE consoleHandle = GetStdHandle(STD_OUTPUT_HANDLE);
CONSOLE_CURSOR_INFO info;
info.dwSize = 100;
info.bVisible = FALSE;
SetConsoleCursorInfo(consoleHandle, &info);
```

Pentru a ascunde cursorul din consolă s-a folosit codul din imaginea de sus. Acest cod se folosește de fișierul header windows.h pentru a seta proprietățile cursorului astfel încât să fie invizibil.

Pentru a schimba tematica aplicației (culorile consolei) s-au folosit comenzi de tipul *system("Color 07")* utilizând fișierul header *windows.h*. Acest fișier header a fost folosit și pentru resetarea sau ștergera elementelor afișate în consolă, utilizându-se comanda *system("cls").*

windows.h este un fișier antet specific Windows pentru limbajele de programare C și C++ , care conține declarații pentru toate funcțiile din API-ul Windows , toate macrocomenzile comune utilizate de programatorii Windows și toate tipurile de date utilizate de diferitele funcții și subsisteme. Acesta definește un număr foarte mare de funcții specifice Windows care pot fi utilizate în C și C++.

Pentru a introduce un delay sau un timp de așteptare de o secundă după fiecare generare a poziției de către calculator s-a folosit comanda *sleep(1)* utilizând fișierul header *unistd.h.*

unistd.h este un antet specific sistemelor Unix (inclusiv Linux și macOS) care conține declarații pentru diverse funcții și constante care oferă acces la API-ul de nivel inferior al sistemului de operare. Aceste funcții sunt folosite pentru gestionarea fișierelor, procese, comunicații inter-procese, manipularea directoarelor, și multe alte operații de sistem.

Pentru a citi câte un singur caracter de la tastatură fără a-l afișa în consolă s-a folosit funcția *getch()* introducând în program fișierul header *conio.h.* Dacă se dorea afișarea caracterului citit se putea folosi funcția *getche().*

conio.h este un antet din bibliotecile standard ale limbajului de programare C care oferă funcții pentru manipularea consolei, inclusiv intrarea și ieșirea de caractere. Acesta este specific pentru sistemele DOS și Windows și nu este standardizat în ANSI C sau POSIX, ceea ce îl face mai puțin portabil. *conio.h* este folosit în principal pentru a facilita interacțiunea cu utilizatorul prin intermediul consolei în aplicațiile care rulează pe aceste platforme.

CAPITOLUL III

3. ANALIZA SOLUȚIEI IMPLEMENTATE

3.1. FORMATUL DATELOR DE I/O

Datele de intrare pentru aplicație sunt apăsările de taste de la tastatură. Acestea trebuie să corespundă cu opțiunile din meniuri sau cu pozițile valide din tabelul jocului.

Datele de ieșire sunt reprezentări grafice în consolă ale elementelor jocului, bazate pe operațiile corespunzătoare selectate și pe starea jocului din acel moment.

CAPITOLUL IV

4. MANUAL DE UTILIZARE

Rulare aplicație

Există următoarele două posibilități de a rula aplicația:

1) Aplicația poate fi lansată în execuție din *Command Prompt* mergând în directorul „./XsiO/” unde se află fisierul executabil folosind spre exemplu comanda *cd calea_catre_director\XsiO* , apoi rulând comanda *XsiO.exe* . Acest lucru lansează programul în execuție.

2) Un alt mod în care aplicația poate fi lansată este de a căuta vizual directorul programului în *File Explorer* sau în locul unde este salvat acesta (de exemplu pe Desktop) și de a efectua dublu-click pe fisierul *XsiO.exe* . Acest lucru lansează programul în execuție.

Interfața aplicației și utilizarea sa este prezentată în descrierea următoare:

- Meniul principal:

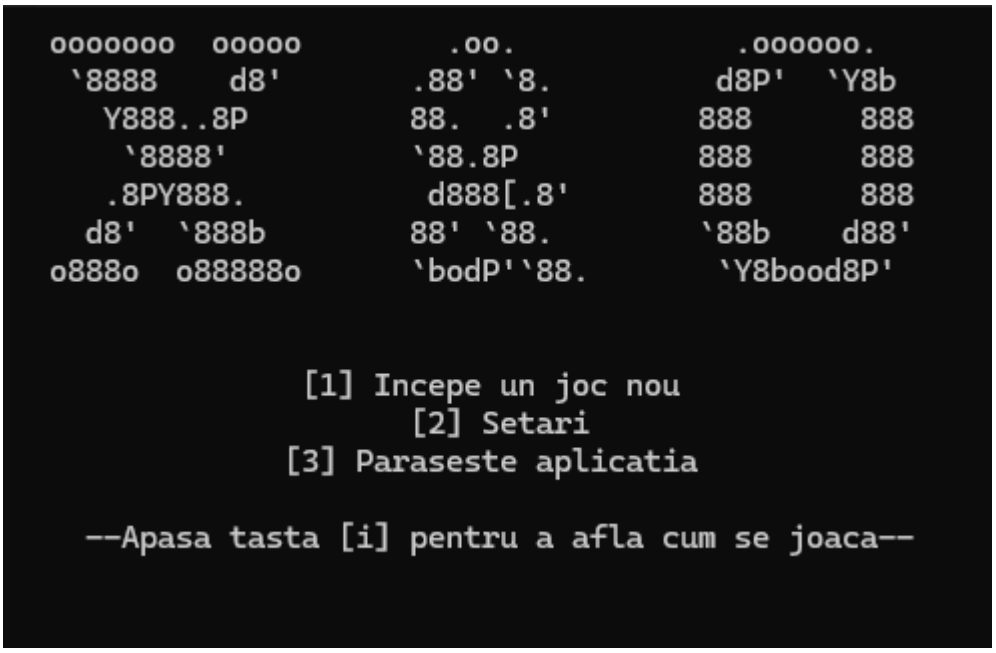

Fig. 2. Meniu principal

Meniul prezintă patru opțiuni: „[1] Începe joc nou”, „[2] Setări”, „[3] Părăsește aplicația” și „[i]”. Pentru a selecta o opțiune se introduce de la tastatură tasta corespunzătoare simbolului dintre parantezele pătrate. La apăsarea tastei corespunzătoare aplicația va efectua operația corespunzătoare fiecărei opțiuni. Introducerea unei taste ce nu corespunde nici unei opțiuni va fi ignorată de către program.

- Meniul cu informații despre cum se joacă:

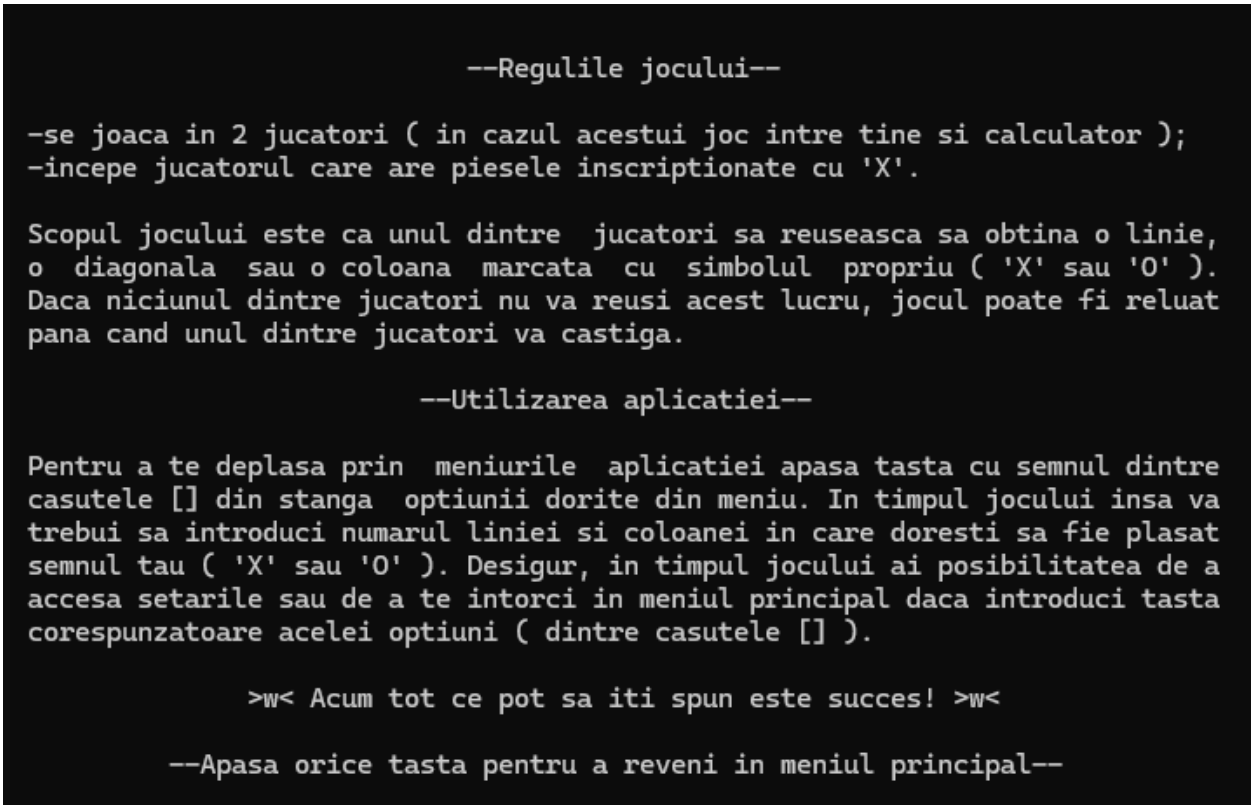

Fig. 3. Meniu cu informații despre cum se joacă

Acest meniu afișează regulile jocului și modul de utilizare al aplicației.

- Meniul pentru setări:

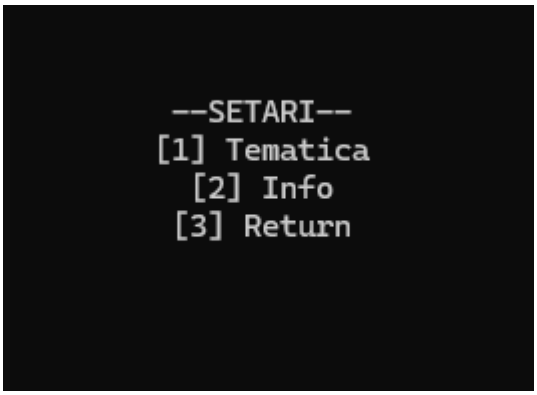

Fig. 4. Meniu pentru setări

Acest meniu afișează setările aplicației. Utilizatorul are posibilitatea de a selecta tematica aplicației, de a vedea informațile despre proiect și de a reveni înapoi în meniul precedent.

- Meniul pentru alegerea tematicii jocului:

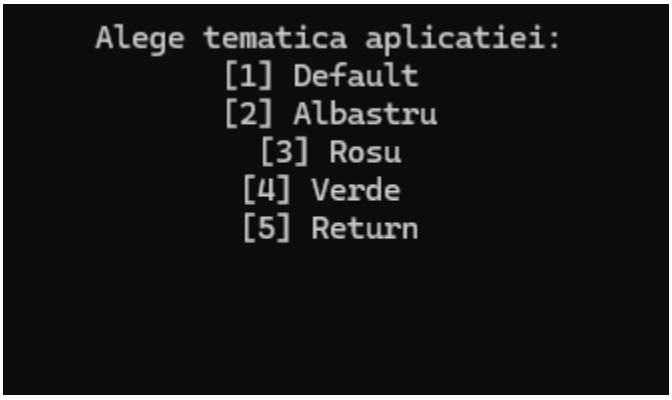

Fig. 5. Meniu pentru tematica aplicației

Acest meniu permite utilizatorului să selecteze tematica dorită a aplicației și să revenă înapoi în meniul pentru setări.

- Meniul cu informații despre proiect:

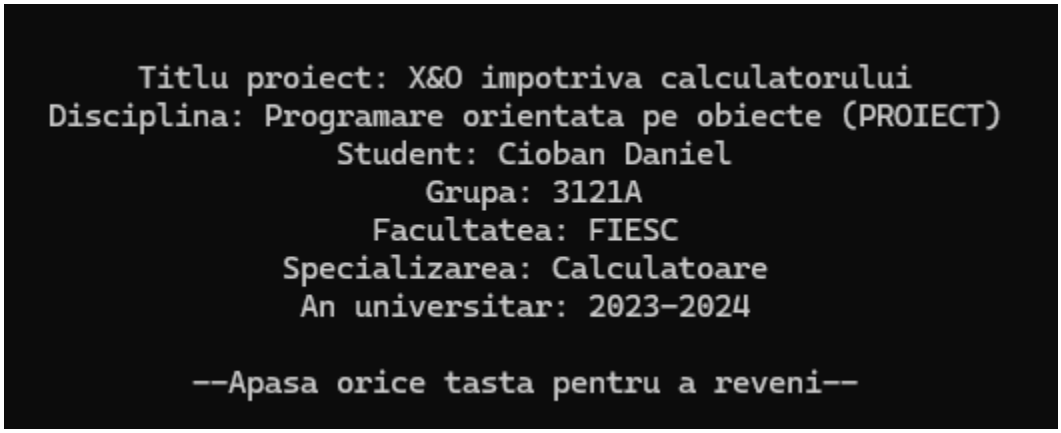

Fig. 6. Meniu cu informații despre proiect

Acest meniu afișează informații despre proiect.

- Meniul pentru începerea unui joc nou:

Fig. 7. Meniu pentru începerea unui joc nou

Acest meniu permite utilizatorului să aleagă cine va introduce setul de coordonate primul pe tabla la începerea unui joc nou.

- Interfața jocului:

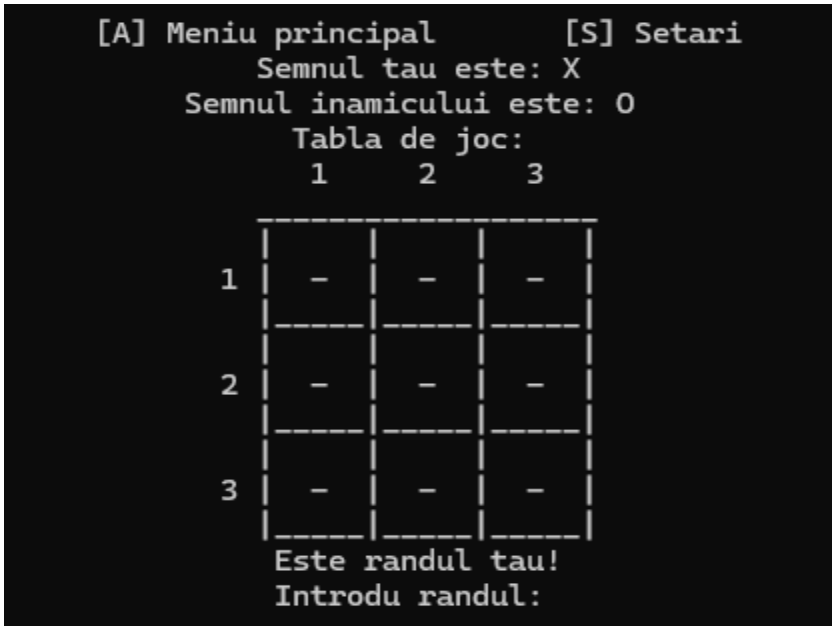

Fig. 8. Interfața din timpul unui joc

În figură este prezentată interfața din timpul unui joc. Sunt afișate opțiunile de întoarcere în meniul principal și de accesare a setărilor. Pentru a accesa aceste opțiuni utilizatorul trebuie să introducă semnul dintre parantezele pătrate corespunzătoare fiecărei opțiuni. De asemenea sunt afișate informații despre semnul utilizatorului și al inamicului/calculatorului, este afișată tabla de joc ce va fi populată cu X și O pe parcursul jocului, cât și informații despre rândul jucătorului.

- Interfața jocului pentru un final al unui joc:

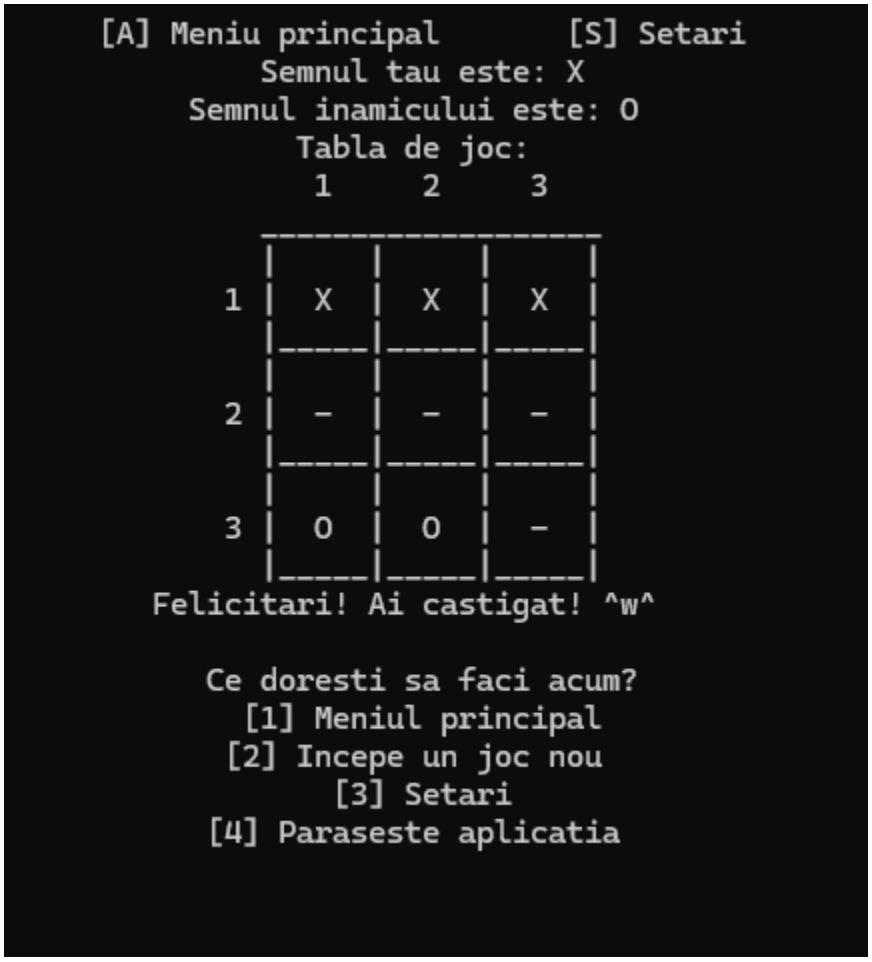

Fig. 9. Interfața jocului pentru câștig

Interfața este la fel ca și cea anterioară însă este menționată starea jocului final pentru utilizator și sunt afișate opțiunile pe care utilizatorul le poate accesa.

CAPITOLUL V

5. CONCLUZII

Modul în care problema a fost abordată este unul simplu. Însă acest lucru nu înseamnă că este și eficient. Structura și funcționalitățile claselor ar putea fi îmbunătățite pentru a utiliza memoria într-un mod eficient și pentru a realiza operațiile într-un timp mai scurt.

Proiectul ar putea fi îmbunătățit prin implementarea următoarelor funcționalități:

- adăugarea sunetelor (de exemplu sunet pentru câștigarea/pierderea jocului, pentru începerea unui joc, etc.) cu mai multe setări pentru a oferi jucătorului posibilitatea de a customiza/modifica jocul după preferințe;
- Adăugarea unui cronometru de la începerea unei noi remize până la încheierea ei;
- adăugarea unui istoric al remizelor cu diferite informații despre joc salvate local într-un fișier (câștigătorul, durata, numărul de mișcări etc.) și crearea/adăugarea unui scoreboard;
- alte modalități de a interacționa cu jocul pe lângă introducerea de la tastatură (de exemplu prin folosirea mouse-ului sau alte modalități);
- adăugarea diferitelor moduri de joc. Pe lângă modul de joc împotriva calculatorului să existe modul de joc împotriva altui jucător pe același dispozitiv/computer, modul de a juca pe o tablă mai mare (de exemplu 9x9) sau/și moduri de dificultate diferite.

CAPITOLUL VI

6. BIBLIOGRAFIE

SURSE BIBLIOGRAFICE DIVERSE

[RCP.1] http://www.cplusplus.com
[RCP.2] https://cplusplus.com
[RCP.3] https://www.geeksforgeeks.org
[RCP.4] https://stackoverflow.com