

PROIECT DISCIPLINA POO

[Implementare Memory Game în C++]

Autor,
ZGHIBARTĂ ELENA

SUCEAVA . 2023

TEMA PROIECT

TEMA SI MOTIVATIA ALEGERII

Tema proiectului este crearea unui joc foarte popular Memory Puzzle sau cum i se mai spune Memory Game.

Am ales această temă de proiect deoarece îmi place să îmbin utilul cu plăcutul, astfel pot sa utilizez cunostințele de lucru cu elemente de POO pentru a realiza o versiune a jocului preferat din copilărie.

Deoarece pâna acum nu am mai realizat o aplicație de genul acesteia, mi s-a părut o adevărată șansa de a crea singură acest joc prin cod în limbaj C++.

Deoarece acest jos reprezenta una din ocupațiile mele din copilărie, am fost pasionată de felul în care mă făcea să gândesc, să am ocazia să mă pun în spatele logicii cu care a fost creat jocul și să îl implementez mi se pare un progres. Astfel implementarea acestei aplicații este o adevărată provocare pentru mine.

CUPRINS

Contents

TEMA SI MOTIVATIA ALEGERII	2
1. ELEMENTE TEORETICE	4
1.1. DESCRIEREA PROBLEMEI	4
1.2. ABORDAREA TEORETICA A PROBLEMEI	5
1.3. ELEMENTE SPECIFICE POO	5
2. IMPLEMENTARE.....	7
2.1. TEHNOLOGII FOLOSITE.....	10
2.2. DIAGRAMA DE CLASE, SCHEMA BLOC, WORKFLOW	11
3. ANALIZA SOLUTIEI IMPLEMENTATE	12
3.1. FORMATUL DATELOR DE I/O.....	12
3.2. STUDII DE CAZ.....	12
3.3. PERFORMANTE OBTINUTE.....	15
4. MANUAL DE UTILIZARE	16
5. CONCLUZII	17
6. BIBLIOGRAFIE	18
6.1. CARTI	18
6.2. SURSE BIBLIOGRAFICE DIVERSE	18

CAPITOLUL I

1. ELEMENTE TEORETICE

1.1. DESCRIEREA PROBLEMEI

Memory Puzzle este un joc clasic de puzzle în care trebuie să găsești perechi de imagini identice. Imaginile sunt afișate pentru 10 secunde, moment în care jucătorul încearcă să memoreze perechile, iar apoi sunt întoarse cu fața în jos. Jucătorul alege pe rând două imagini. Dacă cele două imagini sunt identice acestea sunt afișate și vor rămâne cu fața în sus până la finalul jocului. Dacă cele două imagini nu sunt identice jucătorul le poate vedea pentru 3 sec și apoi vor fi întoarse cu fața în jos și va alege următoarele 2 cartonașe.

Jocul se termină atunci când toate perechile au fost identificate și se află cu fața în sus, moment în care putem spune că jucătorul a câștigat.

Acest joc are drept scop ca utilizatorul să rețină cât mai multe informații în 10 secunde. „Cartonasele” vor fi stocate într-o matrice cu număr par de elemente. Aceasta va fi afișată timp de 10 secunde. După scurgerea timpului pe ecran va apărea o matrice goală iar utilizatorul va trebui să introducă o poziție din matrice. În acest moment elementul de pe poziția dată de utilizator va fi afișat pe ecran, iar utilizatorul este pus în situația în care să aleagă o altă poziție pe care acesta crede că ar fi elementul identic cu cel ales de pe poziția anterioară. În acest moment vor fi verificate cele 2 posibilități:

- Cazul în care utilizatorul a ghicit poziția elementului, elementele vor fi afișate și vor rămâne pe consola afișate în continuare până la finalul jocului.
- Cazul în care utilizatorul nu a ghicit poziția elementului, elementul va fi afișat pentru 3 secunde pentru ca utilizatorul să reușească să memoreze vizual poziția elementului, iar apoi acestea vor dispărea și utilizatorul va trebui să introducă din nou poziția a doua elemente pentru a găsi perechile.

A	#	8	\$
:	B	C	@
A	Z	#	=
8	B	=	Z
@	C	:	\$

?	?	?	?
?	?	?	?
?	?	?	?
?	?	?	?
?	?	?	?

?	?	?	?
?	B	?	?
?	?	?	?
?	B	?	?
?	?	?	?

Fig.1.

Fig.2.

Fig.3.

În figura 1 avem un exemplu de matrice de început în care avem 10 perechi de elemente identice. După cele 10 secunde, timp în care jucătorul are timp să le memoreze, va apărea pe ecran următoarea figură, figura 2, acolo avem matricea elementelor ghicite de jucător care la început este goală, eu am simbolizat pozițiile goale din matrice cu simbolul “?”. După introducerea a două poziții corecte pe care se află o pereche de elemente identice utilizatorul va putea vizualiza matricea din fig.3.

1.2. ABORDAREA TEORETICA A PROBLEMEI

Pentru realizarea aplicației propuse am utilizat CodeBlocks și limbajul C++ în care am lucrat cu clase și fișiere de tip header sau cod sursă pentru a organiza mai clar codul.

În primul rând, m-am gândit ce opțiuni aș dori să aibă utilizatorul în momentul în care rulăm aplicația. Astfel, am creat un meniu cu cinci opțiuni. Prima opțiune va fi aceea de a da start jocului, iar din momentul selectării opțiunii să înceapă jocul după regulile bine stabilite. O a doua opțiune este aceea de a salva jocul, această opțiune presupune salvarea nivelului la care a ajuns utilizatorul. A treia opțiune reprezintă încărcarea jocului, adică posibilitatea de a reveni la nivelul la care a rămas utilizatorul. O a patra opțiune ar fi “Help”, la accesarea acestei opțiuni pe consolă vor fi afișate regulile jocului pentru a veni în ajutor utilizatorului care nu cunoaște jocul. Ultima opțiune este aceea de a părăsi aplicația.

O primă condiție în începerea jocului este aceea de a da șansa utilizatorului să vizualizeze câteva secunde (în cazul meu am ales 10s) elementele din matrice, dar și poziția acestora în matrice. Apoi va începe propriu-zis jocul cu o matrice goală, inițial populată cu simbolul “?”, fiind sugestiv faptul că nu au fost încă găsite elementele pereche.

Utilizatorul va trebuie să introducă o poziție în matrice (rând și coloană), elementul va fi afișat, iar apoi va introduce alte coordonate pentru a încerca să identifice perechea elementului deschis anterior.

Cu ajutorul unor condiții de validare a găsirii unei perechi de elemente identice vor fi verificate toate posibilitățile și va fi completată matricea perechilor de elemente ghicite.

1.3. ELEMENTE SPECIFICE POO

Programarea orientată pe obiecte (POO) este o paradigma de programare care folosește concepte abstracte, aceasta este diferită de programarea procedurală. În cazul POO programele sunt alcătuite din module, acestea pot fi dezvoltate și testate independent. Aceste module pot fi asamblate pentru a forma un program complet.

În această situație module pot fi numite clase. Clasele ne permit să creem obiecte care modelează elementele din lumea reală.

Acest mod de programare ne permite să abstractizăm unele date, aceasta presupune să scoatem în evidență unele detalii mai importante și să le ascundem datele care sunt doar niște detalii.

Abstractizarea este procesul de definire a unei reprezentări abstracte a unui obiect sau concept într-un program. Scopul abstractizării este de a izola detaliile de implementare și de a se concentra pe caracteristicile esențiale ale obiectului sau conceptului. Abstractizarea

este realizată prin intermediul claselor și obiectelor. O clasă definește o structură abstractă care combină datele (variabilele membru) și comportamentul (funcțiile membru).

Încapsularea se referă la împachetarea datelor și funcțiilor care operează asupra acestora într-o unitate logică numită clasă. Ideea principală a încapsulării este de a ascunde detaliile interne ale clasei și de a oferi un set de interfețe publice prin intermediul cărora utilizatorii pot interacționa cu clasa. Încapsularea este realizată prin intermediul modificatorilor de acces. O clasă poate avea trei niveluri de acces pentru membrii săi: public, privat și protejat. Prin încapsulare, putem proteja datele clasei și putem controla modul în care acestea sunt accesate și modificate din exteriorul clasei.

Un alt avantaj al POO este încapsularea, care ne permite să ascundem unele informații sensibile de către utilizatori, prin utilizarea atributului "private".

În aplicația mea am utilizat mai multe clase, obiecte, metode, constructori, dar și lucrul cu fișiere. Pentru fiecare clasă am folosit 2 fișiere: unul .c și unul .h. Fișierul header conține corpul clasei în care se inițializează variabilele, constructorii și celelalte metode. Pe când în fișierul sursă .c se definesc corpul metodelor și al constructorilor. Prin directiva "#include" am inclus fișierele header în meniul principal, dar și fișierele .c în cele header, pentru fiecare clasă.

Pe lângă fișierele cu clase am utilizat și fișiere simple în care am scris unele funcții ajutătoare. Am folosit și mai multe librării pentru a putea accesa proprietățile acestora, cum ar fi culori, lucru cu fișiere, noțiuni de timp, thread-uri, vectori, algoritmi.

CAPITOLUL II

2. IMPLEMENTARE

Implementarea jocului Memory Game a părut initial un lucru simplu, care a început de a o mica organizare a ideilor principale și a ceea de ce aş avea eu nevoie pentru a funcționa aplicația.

În prima etapa am realizat o diagrama a claselor pe care credeam eu ca le-aş folosi pe parcurs iar aceasta arata cam asa.

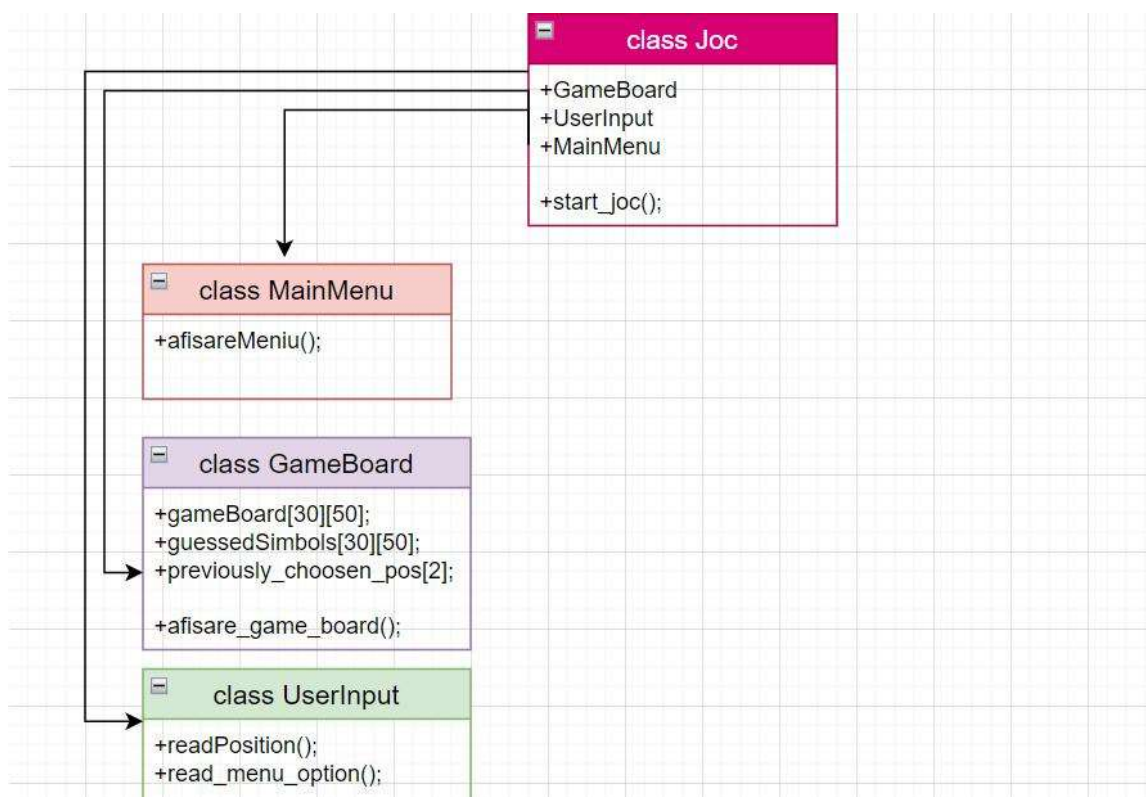


Figure 1. Diagrama inițială a claselor propuse.

La început mi-am construit corpul claselor de care aveam nevoie la început și câteva metode în interiorul acestora. Bineînțeles, mi-am construit și o clasă de bază în care se desfășoară toate operațiile, aceasta se numește clasa "Joc".

Ulterior am mai adăugat și alte clase cum ar fi clasa „Position”, care are rolul de a prelua o poziție în matrice(rând și coloană).

Fisierele cu funcții ajutătoare la fel au fost adăugate mai târziu în care am folosit o funcție specifică de curățare a terminalului în funcție de sistemul de operare, deoarece mi-am dorit ca aplicația să o lansez atât în Windows cât și în Linux.

Clasa joc conține o metodă importantă **start_joc()**, această metodă afișează matricea inițială, apoi într-un **while** citește poziția dată de utilizator, o verifică, verifică de asemenea

dacă utilizatorul a câștigat. O altă metodă este `castigareNivel()`, această metodă asigură trecerea la un nou nivel în momentul în care utilizatorul a câștigat. Apoi cele doua metode recent adaugate cu salvarea și încărcarea din fișier.

Pentru a înțelege cum ar trebui să construiesc algoritmul și logica de completare a matricei cu elementele ghicite am construit clasa `Game Board`. Inițial am utilizat o matrice de 3x2 pentru a înțelege ce am de făcut și cum ar funcționa lucrurile. Clasa `GameBoard` este la fel de importantă ca și clasa menționată mai sus, deoarece se află în stransă legătura cu clasa `Joc`. Metodele clasei sunt impotante deoarece stau la baza funcționării algoritmului.

```
#ifndef GAMEBOARD_H
#define GAMEBOARD_H
#include "Position.h"
#include <random>
#include "Culori.h"
class GameBoard
{
public:
    GameBoard(int nivel);
    void afisare_game_board(int afisare_totala, Position noua);
    void Proccesed_turn(Position poz_noua);
    int verify_if_user_won();
    void Afisare_initiala();
    virtual ~GameBoard();

protected:

public:
    int nivelCurent;
    int numarLinii, numarColoane;
    char** gameBoard;
    char** guessedSymbols;
    Position previously_chosen_pos=Position(-1,-1);
};

#endif // GAMEBOARD_H
```

Figure 2. Clasa `GameBoard` din cadrul proiectului

Aceasta conține mai multe variabile necesare, precum matricea `gameBoard` și cea necesară pentru completare pe parcursul jocului `guessedSymbols`, numărul de linii, numărul de coloane, nivelul curent și o variabilă de tipul `Position` (rand și coloana). Jocul creat de mine are 6 nivele. În funcție de acestea în constructor este construită matricea `gameBoard` și matricea `guessedSymbols`. Apoi am utilizat o variabilă de tip vector pentru a reține toate pozițiile din matrice. Cu ajutorul unui dispozitiv aleatoriu am folosit `shuffle` pentru a umple vectorul cu perechi de caractere pentru a popula ulterior matricea.


```

vector<Position*> vectorPozitii;

for(int i=0; i<numarLinii; i++)
    for(int j=0; j<numarColoane; j++)
    {
        Position* pozitie=new Position(i,j);
        vectorPozitii.push_back(pozitie);

        guessedSymbols[i][j]=' ';
    }

// Use a random device as the seed for the random number generator
std::random_device rd;
std::mt19937 gen(rd());

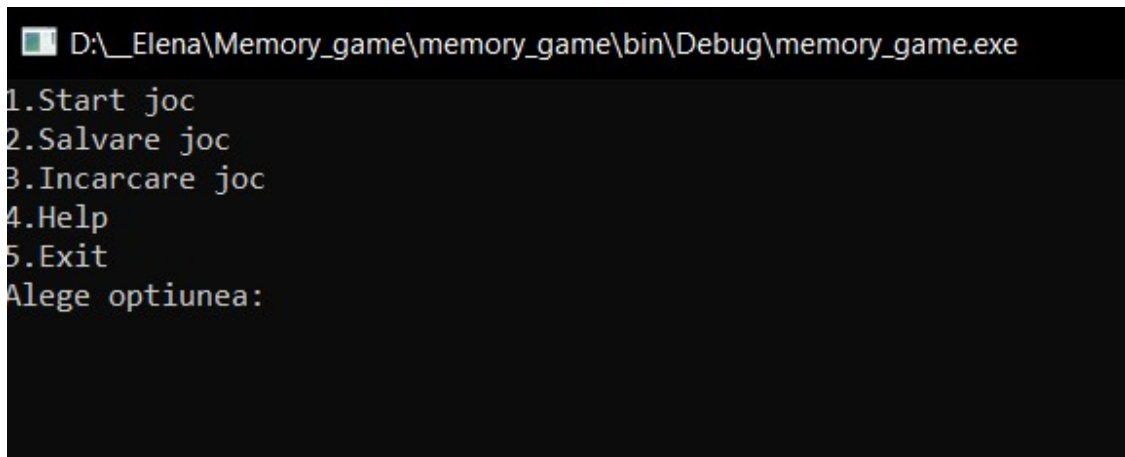
// Shuffle the vector using std::shuffle
std::shuffle(vectorPozitii.begin(), vectorPozitii.end(), gen);
char* sirCaractere="ABCDEFGHIJKLMNOPQRSTUVWXYZ";
int caracterCurent=0;

for(int i=0; i<numarColoane*numarLinii; i=i+2)
{
    gameBoard[vectorPozitii[i]->i][vectorPozitii[i]->j]=sirCaractere[caracterCurent];
    gameBoard[vectorPozitii[i+1]->i][vectorPozitii[i+1]->j]=sirCaractere[caracterCurent];
    caracterCurent++;
}

```

Figure 3. Dispozitivul de generare aleatorie a perechilor de simboluri

Aplicația este realizată sub forma unui meniu, pentru care am creat o clasă separată. În momentul rulării programului, în consolă se afișează meniul aplicației și posibilitatea de a alege opțiunea dorită.



```

D:\_Elena\Memory_game\memory_game\bin\Debug\memory_game.exe
1.Start joc
2.Salvare joc
3.Incarcare joc
4.Help
5.Exit
Alege optiunea:

```

Figure 4. Meniul aplicației

O altă clasă implementată este `UserInput()` care are două metode, una pentru a citi opțiunea din meniu și cealaltă metodă pentru a citi poziția din matrice. Metoda `readPosition()` validează datele, adică dacă a fost introdus un număr greșit al coloanei sau a rândului.

2.1. TEHNOLOGII FOLOSITE

Aplicația care a fost realizată este o aplicație de tip consolă realizată în mediul de programare Code::Blocks. Limbajul folosit este C++ care este utilizat pentru a programa orientat pe obiecte.

Code::Blocks este un mediu de dezvoltare integrat (IDE) gratuit și open-source, conceput pentru programarea în diferite limbaje, inclusiv limbajul C++. Este disponibil pe multiple platforme, inclusiv Windows, Linux și macOS.

Caracteristici principale ale Code::Blocks:

1. Editor de cod: Code::Blocks oferă un editor de cod cu funcționalități precum evidențierea sintaxei, indentarea automată și completarea inteligentă a codului, facilitând scrierea și editarea codului sursă.
2. Compilator și depanator integrat: IDE-ul include un compilator C++ și un depanator (debugger) integrat, permițându-vă să compilați și să rulați aplicațiile C++ direct în interfața IDE-ului. Acest lucru facilitează testarea și depanarea codului.
3. Proiecte și gestionarea fișierelor: Code::Blocks vă permite să creați și să gestionați proiecte, adăugând fișiere, organizându-le în dosare și facilitând gestionarea și compilarea proiectului într-un mod structurat.
4. Suport pentru programarea orientată pe obiecte (OOP): Code::Blocks este un IDE flexibil și compatibil cu programarea orientată pe obiecte. Vă permite să creați și să lucrați cu clase, obiecte, moștenire, polimorfism și alte concepte OOP specifice limbajului C++.
5. Plugin-uri și extensibilitate: Code::Blocks poate fi extins prin intermediul plugin-urilor pentru a adăuga funcționalități suplimentare sau pentru a se integra cu alte instrumente și biblioteci.

În ceea ce privește limbajul C++ utilizat pentru programarea orientată pe obiecte, este un limbaj de programare puternic și versatil care permite dezvoltarea de aplicații complexe. POO în C++ se bazează pe concepte precum clase, obiecte, moștenire, polimorfism și încapsulare.

2.2. DIAGRAMA DE CLASE, SCHEMA BLOC, WORKFLOW

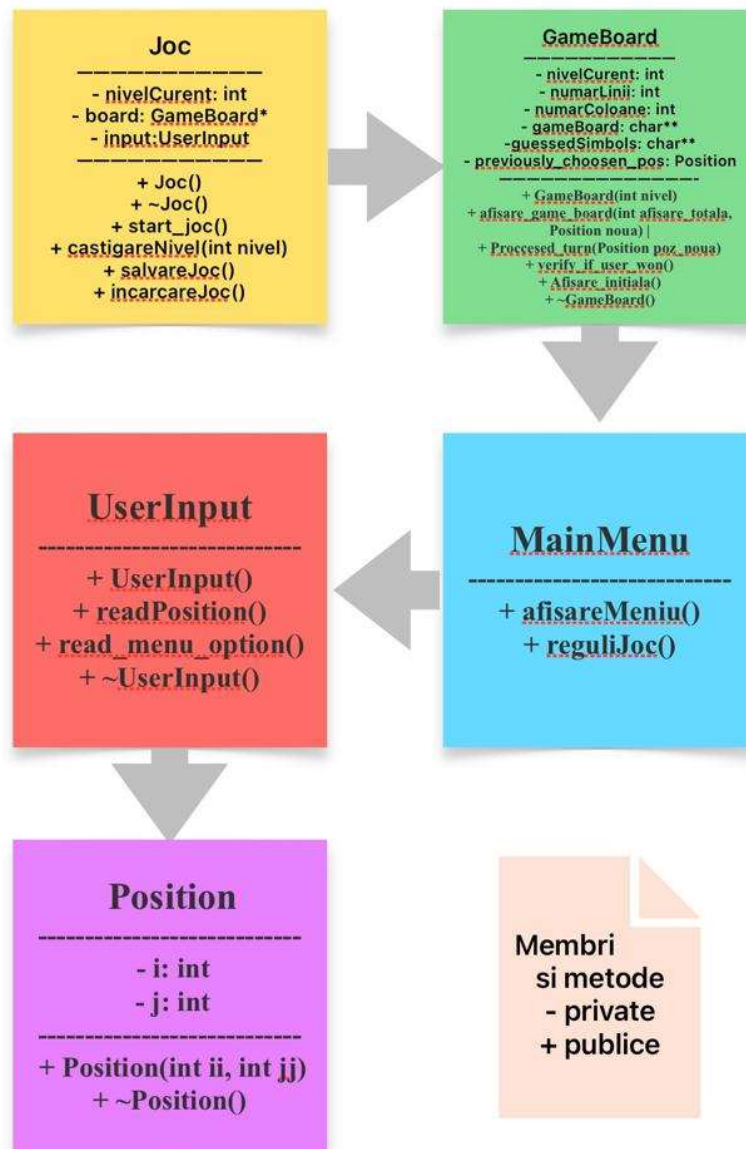


Figure 5. Digrama UML de clase a aplicatiei

CAPITOLUL III

3. ANALIZA SOLUTIEI IMPLEMENTATE

3.1. FORMATUL DATELOR DE I/O

Datele de intrare sunt sub form[de cifre. Pentru a alege opțiunea utilizatorul va introduce o cifra de la 1 la 5 pentru a selecta una din cele 5 optiuni. În momentul în care a selectat prima opțiune se va da start jocului astfel utilizatorul va fi pus în postura de a introduce o poziție în matrice. Indexarea matricelor începe de la 0, de exemplu pentru o matrice de 4x5 numar de linie va putea fi de la 0 la 3, iar cel de coloana între 0 și 4. La opțiunea a doua de salvare in fisier utilizatorul un trebuie sa introduca date, doar este nevoie de a tasta o oricare tasta si de a da enter penrtu a il conduce inapoi la meniul principal. La fel ca in cazul optiunii a doua va fi si pentru opțiunea a treia. Alte date de intrare un sunt necesare decât cele de mai sus.

De exmplu pentru o matrice 3x2:

```
1(optiunea aleasa)
0 0(primul rand prima coloana elementul,elementul guessedSymbols[0][0])
1 2
.....s.a.m.d.
```

Datele de ieșire sunt cele 2 matrici. Prima matrice este matricea de dimensiunile specifice nivelului la care se afla jucătorul și aceasta este populată random cu perechi de caractere dintr-un șir menționat. Această matrice este utilizata pentru a se afișa utilizatorului pentru primele 10 secunde ca acesta să poată memora visual cum arată matricea. La fel aceasta este reperul după care se completează matricea cu simbolurile ghicite. Cealaltă matrice este cea pe care utilizatorul trebuie să o populeze cu perechi de elemente identice pentru ca acesta să câștige.

3.2. STUDII DE CAZ

În rândurile de mai jos voi prezenta primul nivel de joc și datele respective de intrare și de ieșire. În secțiunea meniu voi tasta 1 pentru a alege prima opțiune, iar apoi voi încerca să memorez în cele 10 secunde locul caracterelor în tabla de joc.

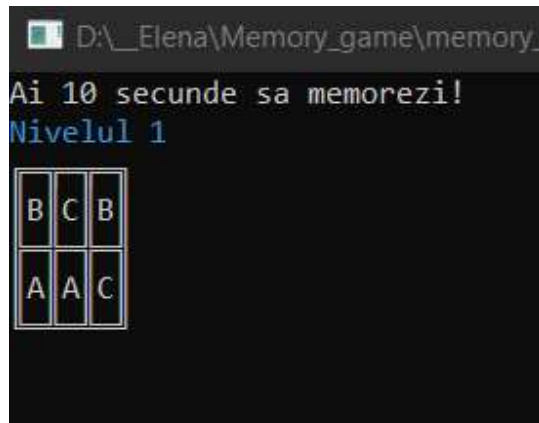


Figure 6. Un exemplu de nivelul 1 al jocului.

Apoi după cele 10 secunde pe consolă se afișează matricea `guesesdSimbols`. Iar utilizatorul este pus în situația de a alege o poziție în matrice.

```

Nivelul 1
  ? ? ?
  ? ? ?

Dati pozitia din matrice(rand si coloana)
0 0
  
```

```

Nivelul 1
  B ? ?
  ? ? ?

Dati pozitia din matrice(rand si coloana)
  
```

```

Nivelul 1
  B ? B
  ? ? ?

Dati pozitia din matrice(rand si coloana)
  
```

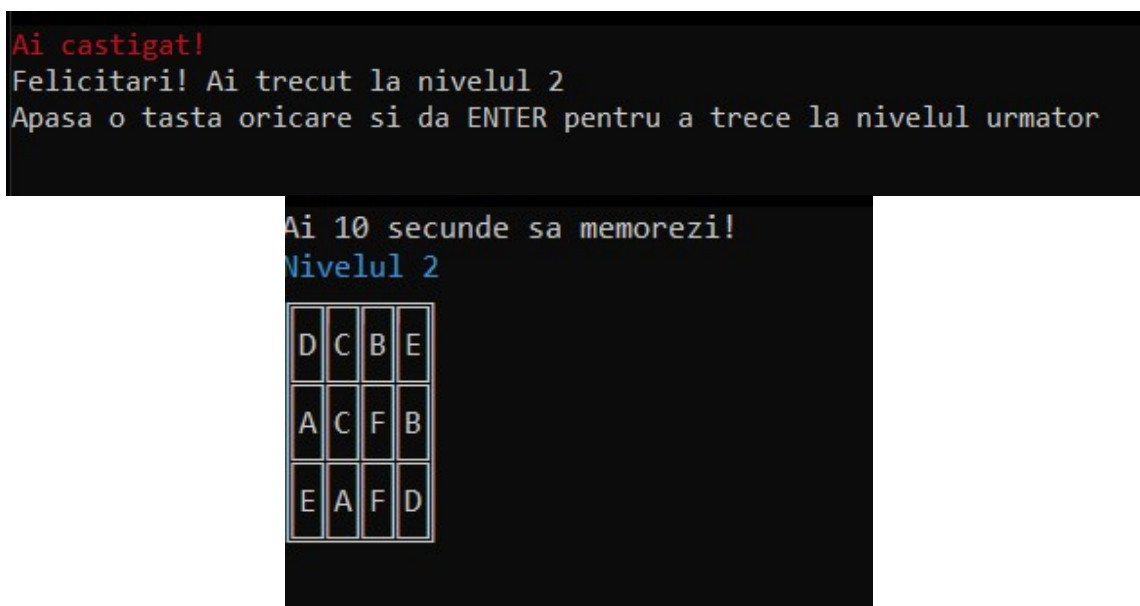


Figure 7. Exemplu de date introduse și rezultatul trecerii de primul nivel.

Imaginile de mai sus reprezintă pașii pe care utilizatorul îi face în timpul jocului și cum înainteză lucrurile pe parcursul rulării aplicației.

Pentru lansarea în execuție a programului în Linux am folosit o mașină virtuală Virtual Box.

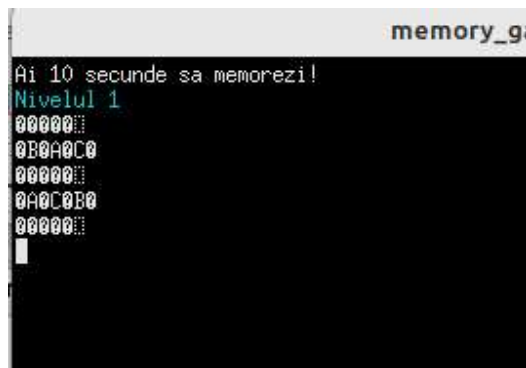


Figure 8. Aplicația în Linux

Din cauză că am folosit anumite simboluri pentru a desena conturul tablei de joc sau al matricei, în Linux acestea nu mai sunt valabile și un îmi recunoaște codul lor ASCII, astfel se afișează alte caractere în locul acestora. Acesta este un aspect negativ care l-am scăpat din vedere de la începutul efectuării proiectului astfel a fost peră târziu să remediez problema. În rest aceasta funcționează perfect normal exact cum am gândit-o și în Linux.

3.3. PERFORMANTE OBTINUTE

Pentru a măsura timpul de rulare al jocului meu am folosit în main funcții a timpului, cum ar fi "clock()". Am pus codul următor în main și am rulat programul.

Codul utilizat pentru a măsura timpul rulării aplicației:

```
clock_t start = clock();

// Codul tău pentru jocul de memorie

clock_t end = clock();
double duration = static_cast<double>(end - start) / CLOCKS_PER_SEC;

std::cout << "Durata de rulare a jocului: " << duration << " secunde\n";
```

Timpul de execuție al jocului este 0.476s

CAPITOLUL IV

4. MANUAL DE UTILIZARE

Manual de Utilizare – Memory Game

Bun venit la Memory Game! Aceasta este o aplicație interactivă care îți testează abilitățile de memorie. Scopul jocului este să găsești toate perechile de cartonașe identice. Urmărește instrucțiunile de mai jos pentru a începe și a te bucura de joc.

1. Descărcare și instalare

- Descarcă sursa aplicației Memory Game de pe [<https://github.com/Zghibarta-Elena/Memory-Game>] .
- Deschide Code Blocks pe dispozitivul tău.
- Crea un nou proiect în Code Blocks.
- Importă sursa aplicației Memory Game în proiectul nou creat.

2. Compilare și rulare

- Asigură-te că proiectul este configurat corect pentru compilarea surselor C/C++ în Code Blocks.
- Salvează și compilează proiectul utilizând opțiunile de compilare din Code Blocks.
- După ce compilarea este finalizată cu succes, poți rula aplicația Memory Game făcând clic pe butonul "Run" din Code Blocks.
- Consola va afișa jocul de memorie și vei putea începe să joci.

3. Jocul

- Vei vedea o grilă de cartonașe așezate cu fața în jos în consolă.
- Folosește cifre pentru a selecta poziția în tabla de joc, de exemplu 0 1 pentru a accesa elementul al doilea de pe primul rând.
- Apasă tasta Enter pentru a întoarce cartonașul și a dezvălui imaginea din spatele său.
- Alege un alt cartonaș și încearcă să găsești perechea sa identică.
- Scopul este să găsești toate perechile.
- Dacă două cartonașe se potrivesc, ele vor rămâne întoarse. Dacă nu se potrivesc, ele vor fi întoarse înapoi.
- Continuă să întorci cartonașe și să găsești perechi identice până când toate cartonașele sunt potrivite.

4. Finalul jocului

Când ai găsit toate perechile de cartonașe, jocul se va încheia și vei primi un mesaj de felicitare și posibilitatea de a trece la nivelul următor.

CAPITOLUL

5. CONCLUZII

Memory Game este o aplicație de joc de memorie simplă, implementată într-un mediu de consolă utilizând Code Blocks. Jocul a demonstrat abilități de bază ale algoritmilor de memorie și oferă o experiență interactivă pentru utilizatori. Iată câteva avantaje și dezavantaje ale aplicației Memory Game:

Avantaje:

Simplitate: Jocul este simplu și ușor de înțeles, oferind o experiență accesibilă pentru toți utilizatorii.

Dezvoltarea abilităților de memorie: Memory Game contribuie la îmbunătățirea și antrenarea abilităților de memorie a utilizatorilor, în special a celor legate de retentarea și recunoașterea modelelor.

Performanță: Datorită implementării într-un mediu de consolă, jocul se poate executa eficient și rapid pe diferite dispozitive.

Dezavantaje:

Interfață vizuală limitată: Deoarece aplicația este în consolă, utilizatorii nu beneficiază de o interfață vizuală grafică atrăgătoare sau de animații avansate. Aceasta poate reduce atractivitatea și experiența de utilizare.

Limitări de interacțiune: Fiind un joc de consolă, utilizatorii interacționează folosind tastatura, ceea ce poate fi mai puțin intuitiv și mai puțin captivant decât utilizarea unui interfață tactile sau cu mouse.

Direcții de dezvoltare viitoare

În viitor, pentru a îmbunătăți și extinde proiectul Memory Game, pot fi luate în considerare următoarele direcții:

Interfață grafică: Dezvoltarea unei interfețe grafice mai atractive și interactive poate spori atractivitatea jocului și poate oferi o experiență mai captivantă utilizatorilor. Folosirea unor biblioteci grafice precum SFML sau SDL poate facilita dezvoltarea unei interfețe grafice mai complexe.

Diversificarea nivelelor de dificultate: Putem adăuga un timp după care să stabilim performanțele utilizatorului sau un timp maxim de completare cu cartonașele găsite, poate crește complexitatea jocului și poate oferi o provocare mai mare utilizatorilor avansați.

Sistem de înregistrare și comparare a scorurilor: Implementarea unui sistem de înregistrare a scorurilor și clasamentului poate încuraja competiția între jucători și poate oferi o motivație suplimentară pentru a juca jocul.

Adăugarea de funcții suplimentare: Încorporarea de funcții suplimentare, cum ar fi sugestii, sunet sau bonusuri.

CAPITOLUL VI

6. BIBLIOGRAFIE

6.1. CARTI

- [RCP.1] B.Stroustrup, The C++ Programming Language, Addison-Wesley, 1991
- [RCP.2] Knuth, D. E. (1973). The Art of Computer Programming, Volume 1: Fundamental Algorithms. Addison-Wesley.

6.2. SURSE BIBLIOGRAFICE DIVERSE

- [RCP.3] <http://www.cplusplus.com>
- [RCP.4] <https://stackoverflow.com/>
- [RCP.5] <https://www.tutorialspoint.com/index.htm>