Proiect Disciplina POO

Problema Labirint

student:

**Pușcașu Emanuele-Nicholas**

grupa:

**3121a**

# Tema proiectului și motivația alegerii

Tema proiectului este rezolvarea unei probleme de tip labirint.

Am ales acest proiect deoarece, pentru mine pare o adevarată provocare să lucrez în cod astfel încât să am ca rezultat o consolă ce afișează grafic un labirint generat care poate fi interpretat și sub forma unui joc clasic 2D.

Un alt motiv pentru care am ales acest proiect se datorează filmului Maze Runner, în care personajele sunt blocate în mijlocul unui labirint care se schimba în fiecare zi sau pe măsura ce aceștia cauta o cale de ieșire;

# Descrierea programului

Obiectivul acestui „joc” pe consola este ca jucătorul (Reprezentat de caracterul „P”) sa parcurga labirintul de la punctul „S” (Start) pana la punctul „E”(End).

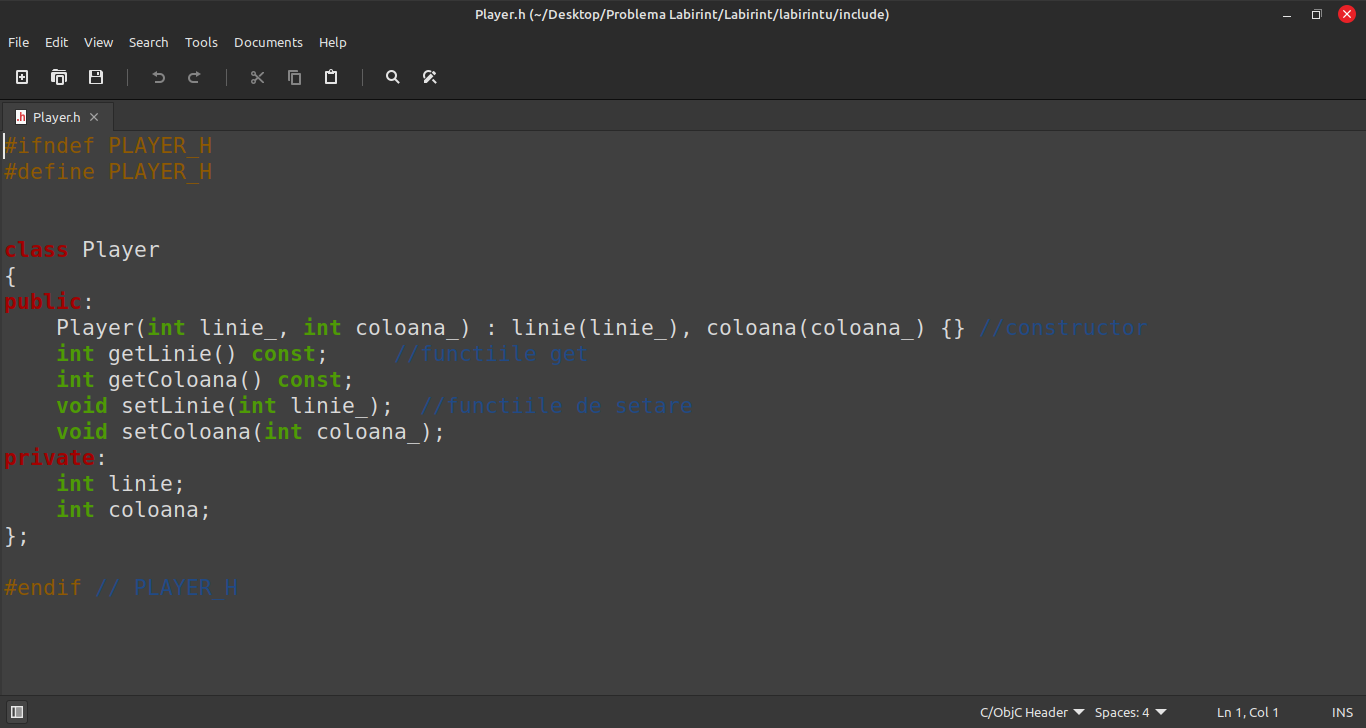
Dificultatea jocului depinde strict de mărimile setate la început de jucător. (5x5 fiind cel mai mic și mai ușor labirint iar 50x85 fiind cel mai mare și totodata mai complex).

## Clasele

Acest proiect dispune de 3 clase:

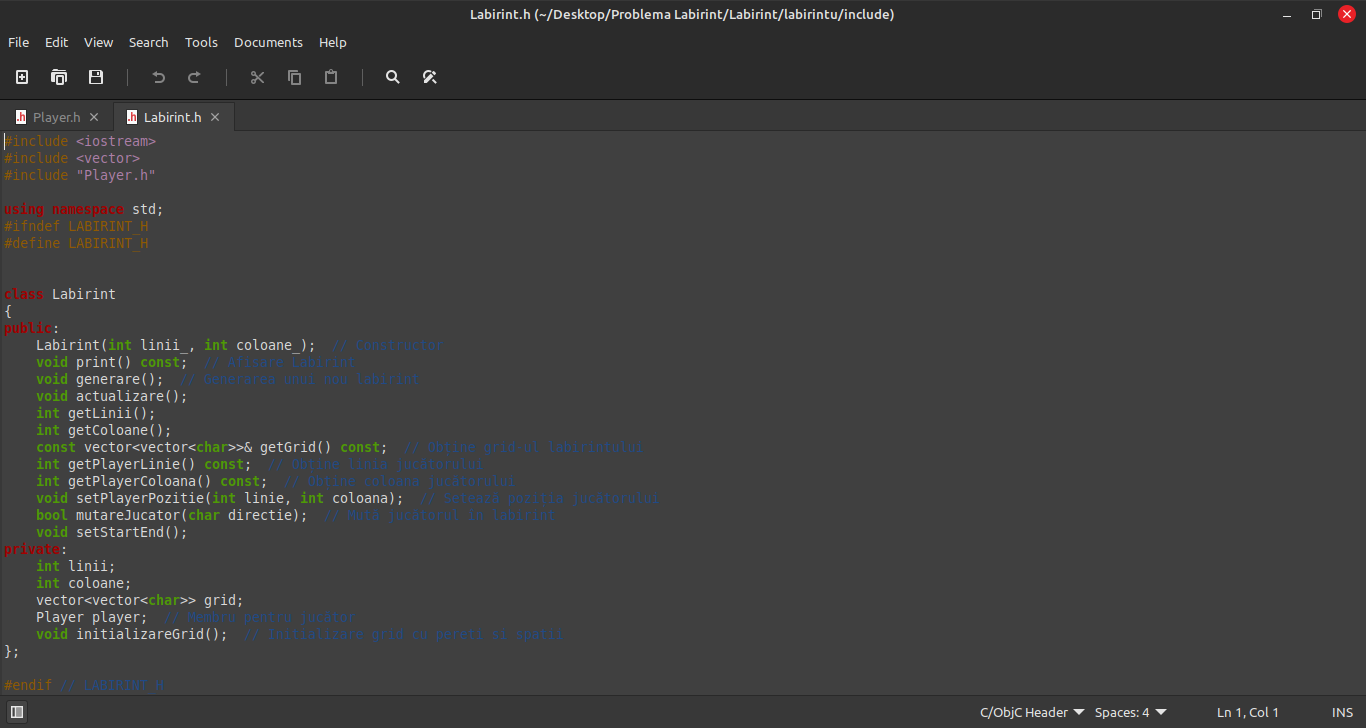
1. Clasa Player reprezintă jucătorul și are următoarele funcționalități:

* Constructorul Player(int linie\_, int coloana\_) inițializează jucătorul la poziția specificată de linie și coloană.
* Funcțiile getLinie() și getColoana() returnează poziția liniei și coloanei jucătorului.
* Funcțiile setLinie(int linie\_) și setColoana(int coloana\_) setează poziția liniei și coloanei jucătorului la valorile specificate.



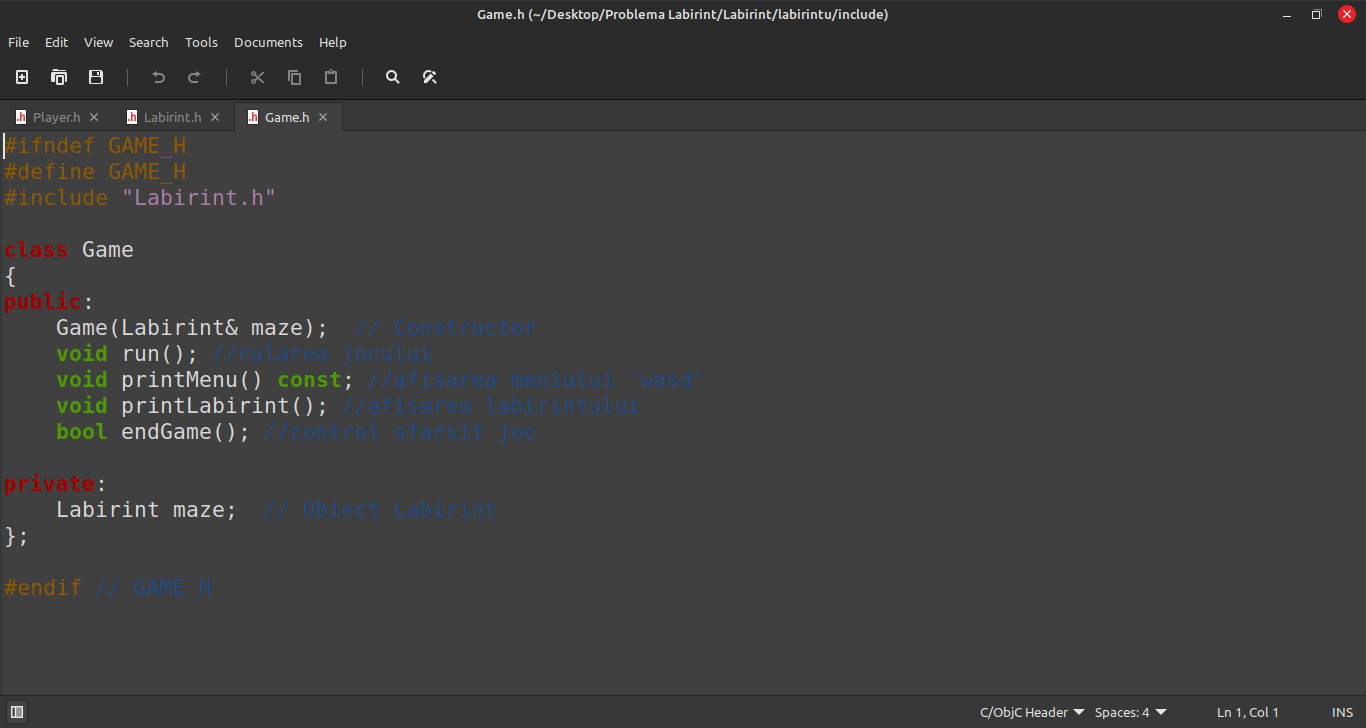
2. Clasa Labirint reprezintă labirintul și are următoarele funcționalități principale:

* Constructorul Labirint(int linii\_, int coloane\_) inițializează obiectul labirint cu numărul specificat de linii și coloane.
* Funcția print() afișează labirintul în consolă, inclusiv poziția jucătorului.
* Funcția generare() generează un nou labirint utilizând algoritmul Recursive Backtracking. Aceasta începe de la o celulă aleatoare și vizitează recursiv celulele vecine nevizitate până când toate celulele au fost vizitate. Astfel se creează o rețea de drumuri și se formează labirintul.
* Funcția actualizare() actualizează labirintul pentru a asigura că există o cale de ieșire în cazul în care numărul de linii și coloane este par.
* Funcția getGrid() returnează grid-ul labirintului, care este o matrice de caractere reprezentând peretele ('#') și spațiul gol (' ').
* Funcțiile getPlayerLinie() și getPlayerColoana() returnează poziția liniei și coloanei jucătorului în labirint.
* Funcția setPlayerPozitie(int linie, int coloana) setează poziția jucătorului la coordonatele specificate.
* Funcția mutareJucator(char directie) mută jucătorul în labirint în funcție de direcția specificată ('w', 'a', 's', 'd') și returnează true dacă mutarea a fost validă sau false în caz contrar.
* Functia setStartEnd() seteaza punctele de start și sfârșit.



3. Clasa Game reprezintă jocul și are următoarele funcționalități:

* Constructorul Game(Labirint& maze) inițializează jocul cu un obiect labirint specificat.
* Functia run() ruleaza jocul labirintului în consola.
* Funcția printMenu() afișează opțiunile disponibile în joc.
* Funcția printLabirint() afișează labirintul și poziția jucătorului în consolă.
* Funcția endGame() verifică dacă Playerul a ajuns la destinație.



Toate aceste 3 clase au o ierarhie specifica:

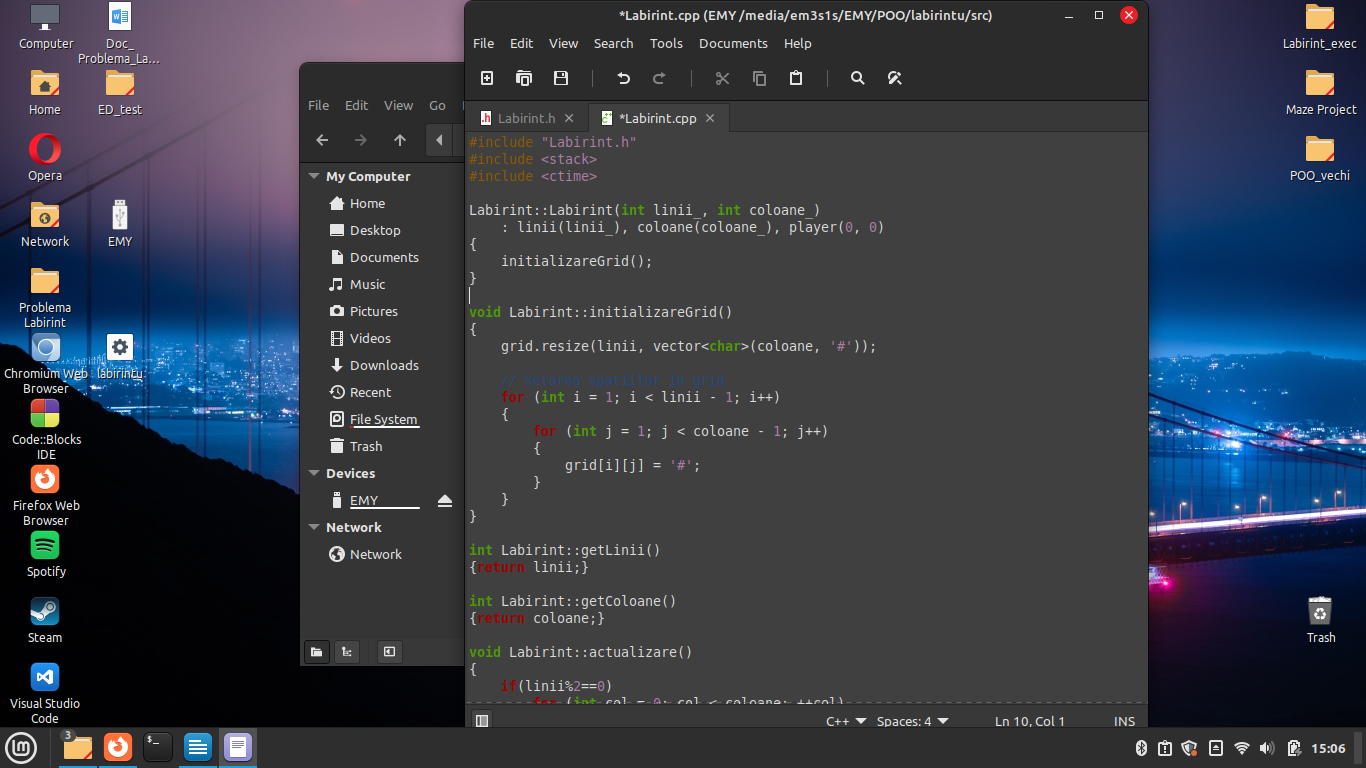
Clasa Player este inclusa în clasa Labirint deoarece Player-ul este o entitate ce se afla în labirint și se mișca în cadrul acestuia.

Clasa Labirint este inclusa în clasa Game deoarece jocul este ceea ce controleaza labirintul respectiv;

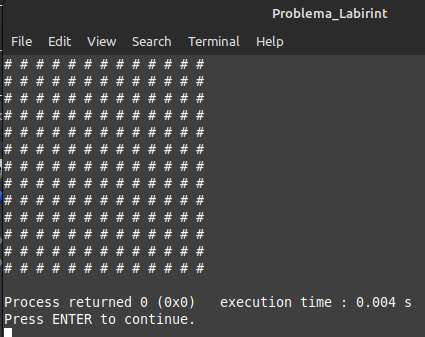
În acest proiect labirintul este reprezentat sub forma de matrice (grid) în care fiecare celula a acesteia poate avea 2 stari:

* Starea liber (o celula de tip alee, ‘ ‘)– Jucătorul va putea parcurge aleele .
* Starea ocupat (o celula de tip perete, ‘#’) – Jucătorul nu va putea sa treacă prin pereți .

Pentru inceput am creat matrice plina de celule care nu pot fi parcurse („#”), sau pe scurt un labirint fara alee sau cai de acces.



In acest cod am initializat grid-ul folosindu-ma de variabila grid din clasa labirint declarata ca matrice de tip vector<vector<char>>



Exemplu de afisare in terminal a unui grid pentru labirint:

Aceasta matrice/grid reprezinta forma initiala a unui labirint dupa ce acesta este initializat ca obiect;

## Librarii folosite

Pentru acest proiect am mai folosit cateva biblioteci in plus pentru a ma ajuta la aceasta metoda:

* #include <stack> - pentru memorarea tuturor casutelor parcurse intr-o stiva.
* #include <vector> - pentru o manipulare mai eficienta a grid-ului labirint.
* #include <cstdlib> - pentru a ma folosi de functia aleatorie rand() la generarea labirintului.
* #include <ctime>- Am folosit aceasta librărie pentru a initializa seed-ului functiei rand() un numar aleatoriu în funcție de ora, minutul și secunda respectiva a sistemului utilizat pentru a genera de fiecare data un labirint diferit.
* Secvența de cod pentru initializarea seed-ului functiei rand(): „srand(time(nullptr));”.

## Generarea Labirintului

Pentru generarea labirintului am folosit o metoda recursiva;

Inițial, se alege o celulă de start aleatorie cu ajutorul functiei rand() în labirint. Aceasta este reprezentată de coordonatele start\_linie și start\_coloana. Celula de start este adăugată în stiva cellStack și este marcată ca fiind o celulă liberă în matricea grid.

În continuare, se intră într-o structura repetitiva while care se execută până când stiva cellStack este goala.

În fiecare repetare a ciclului, se extrage ultima celulă din stiva folosind cellStack.top(), și se salvează coordonatele acesteia în variabilele lin și col.

Se verifică vecinii nevizitați ai celulei curente. Aceasta se face prin verificarea celulelor de sus, jos, stânga și dreapta față de celula curentă în matricea grid. Dacă un vecin este nevizitat și are un zid între el și celula curentă (reprezentat de caracterul '#' în grid), acesta este adăugat în vectorul neighbors(vecini) ca o pereche de coordonate (linie, coloană).

Dacă există vecini nevizitați în neighbors, se alege un vecin aleatoriu din acest vector (neighbors[index]) și se salvează coordonatele acestuia în variabilele next\_lin și next\_col.

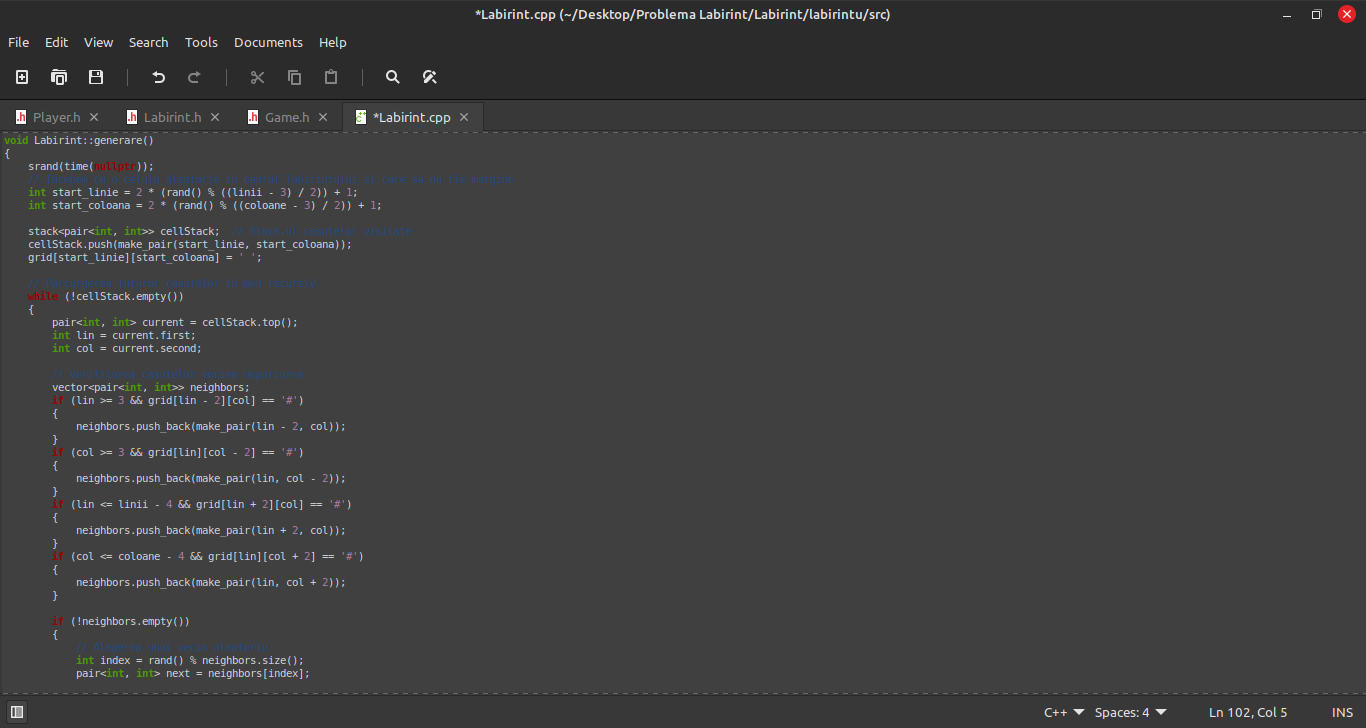
Se eliberează zidul dintre celula curentă și vecinul ales prin marcarea celulei intermediare ca fiind liberă (grid[(lin + next\_lin) / 2][(col + next\_col) / 2] = ' ') și marcarea vecinului ca fiind vizitat (grid[next\_lin][next\_col] = ' ').

Vecinul ales este adăugat în stiva cellStack, pentru a continua parcurgerea recursivă a acestuia.

Dacă nu există vecini nevizitați în neighbors, înseamnă că s-a ajuns la o celulă fără vecini nevizitați și se face o parcurgere înapoi prin extragerea celulei curente din stiva folosind cellStack.pop().

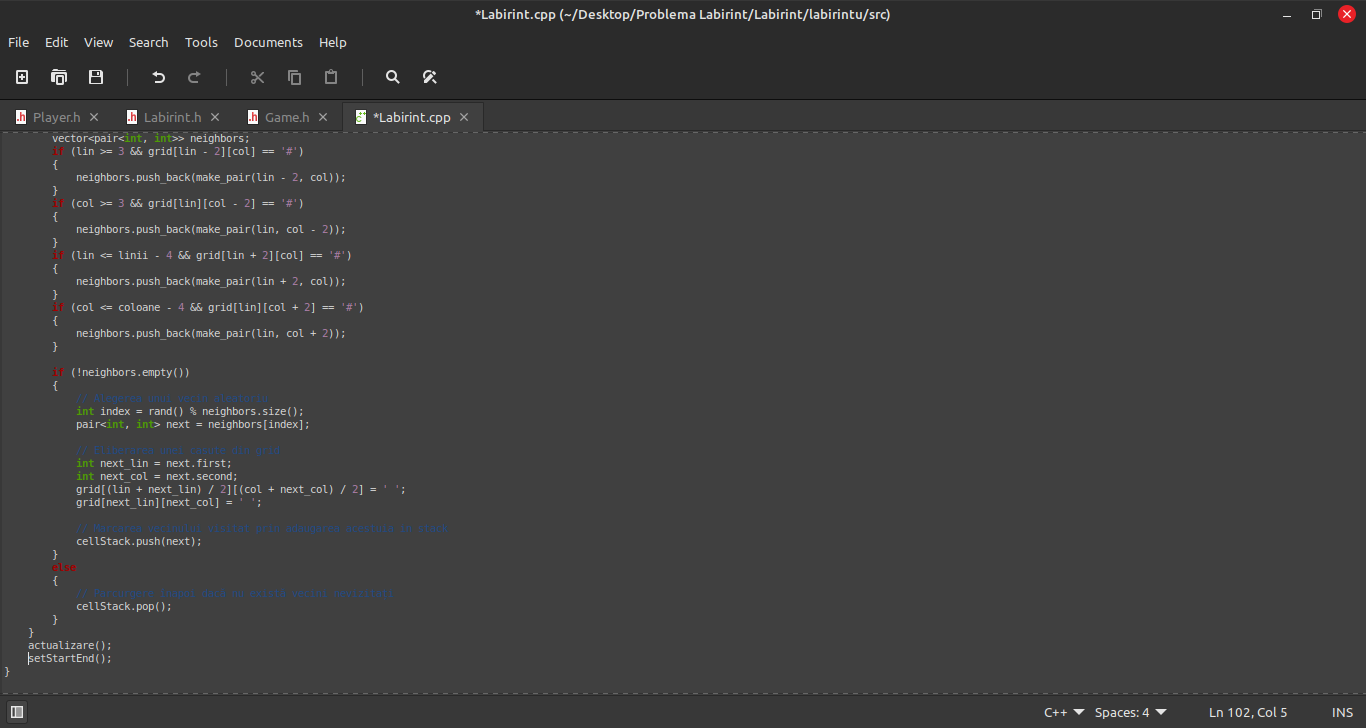
Acest proces se repetă până când stiva cellStack devine goala, moment în care parcurgerea recursivă a labirintului se încheie.

Codul pentru Generarea labirintului:



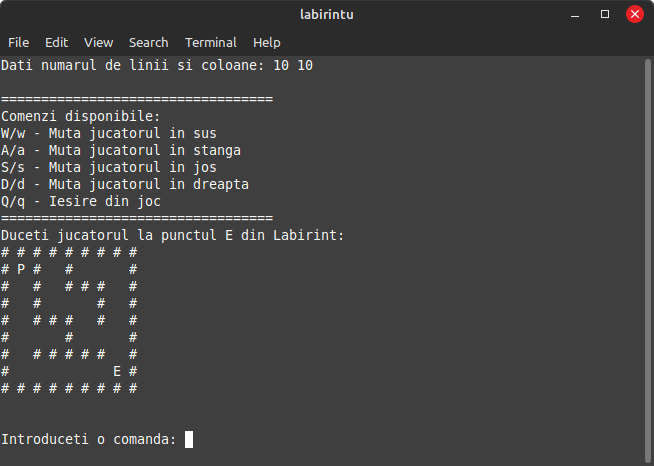
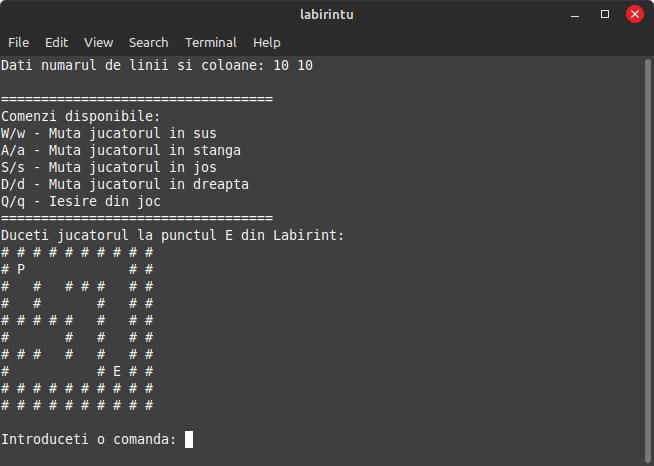
După cum se poate observa la începutul functiei de generare am initializat seed-ul pentru functia random;

Totodata se poate observa ca la aceasta funcție de generare sunt prezente la sfarsit și apelurile de actualizare și setare a punctelor de start/end ale labirintului;

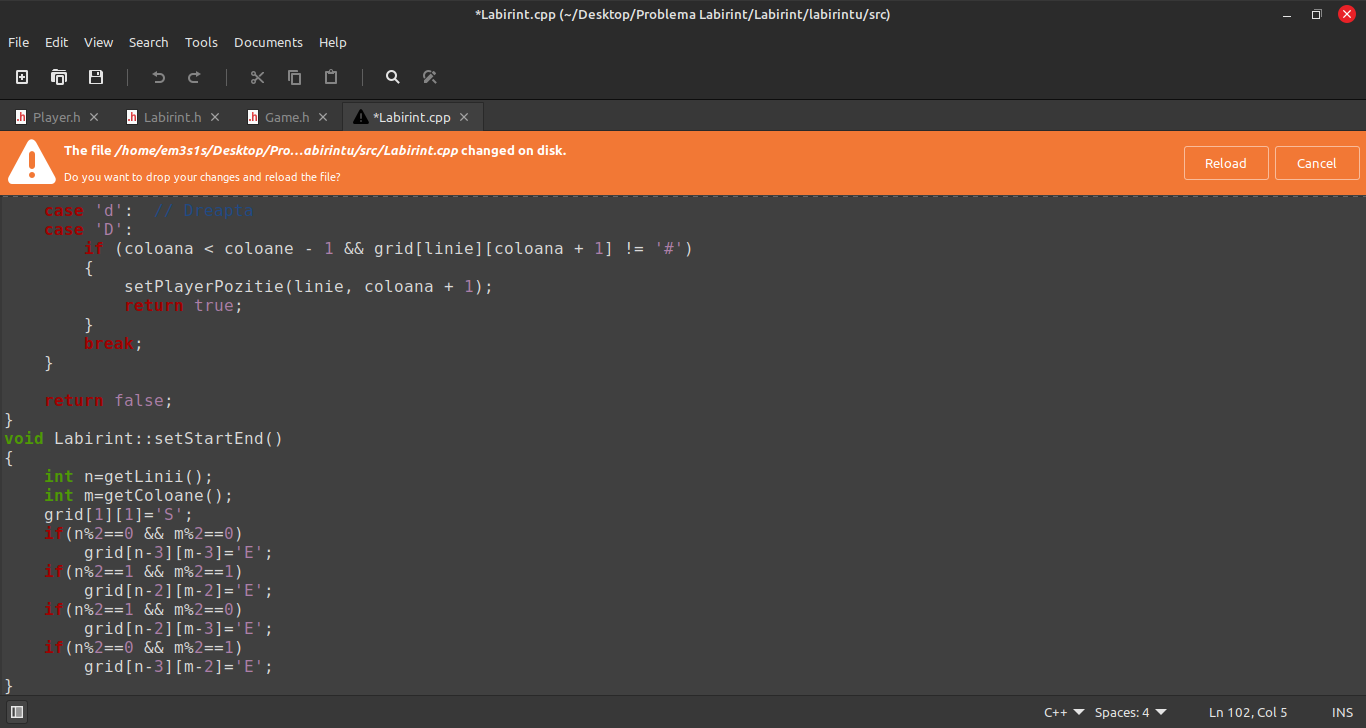
 Principalul motiv pentru care am scris functia de actualizare se datoreaza generarii ambigue al labirintului când sunt introduse dimensiuni pare.

Deși labirintul era generat fără probleme acesta prezenta 2 rânduri de pereți la marginile de jos si/sau dreapta.

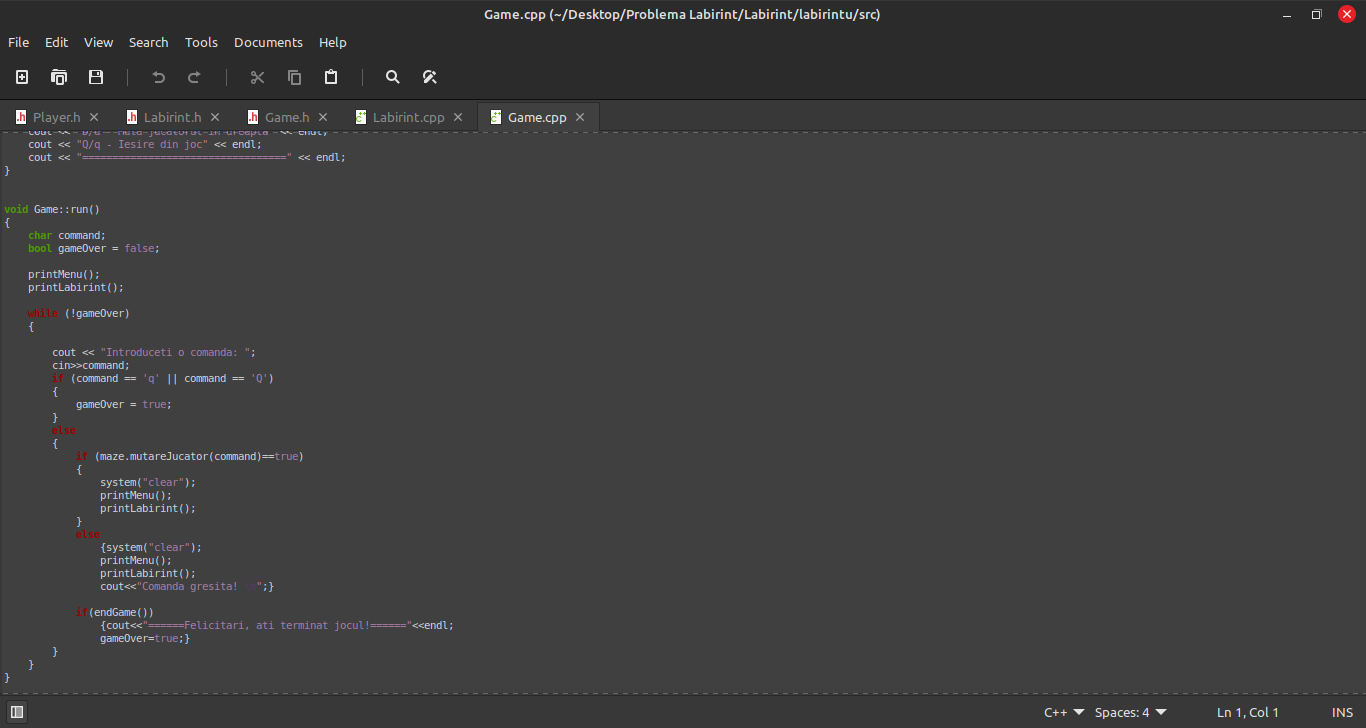
Afisarea labirintului 10x10 fără actualizare Afisarea labirintului 10x10 cu actualizare



Functia setStartEnd()



Pentru aceasta funcție am setat Startul și Endul în funcție de fiecare caz al generarii labirintului (paritatea sau imparitatea liniilor/coloanelor).



## Funcționarea jocului

Functionalitatea jocului labirint se datorează functiei run() din cadrul clasei Game.

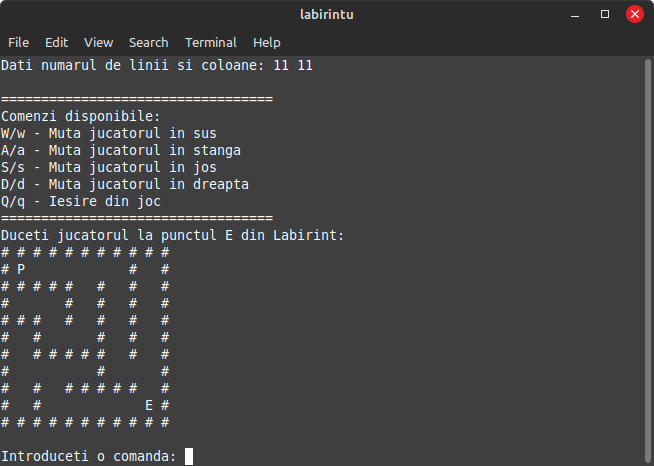
Aceasta funcție are la rândul ei o serie de apeluri și variabile :

* printMenu() - pentru afisarea comenzilor disponibile în joc.
* PrintLabirint().
* mutareJucator(command) – de tip bool pentru validarea comenzii și mutarea jucatorului.
* EndGame() - pentru a verifica dacă jocul s-a terminat.
* Command de tip char (w,a,s,d,q) și gameOver de tip bool pentru a sfârși jocul dacă sunt indeplinite conditiile.

### Rularea jocului/programului

În momentul în care rulam programul, consola ne va afisa un mesaj corespunzător pentru setarea dimensiunilor labirintului (cu restrictia ca dimensiunile sa nu fie mai mici decât 5 și nu mai mari decât 50 pentru linii și respectiv 85 pentru coloane).

Odată introduse dimensiunile labirintului jocul va începe.



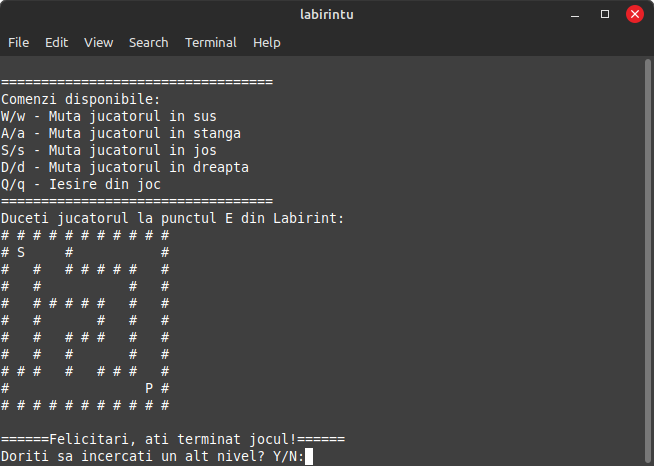
În timpul jocului:

* va fi afisat permanent un meniu în care sunt afisate comenzile disponibile;
* un mesaj în care ne va spune obiectivul jocului;
* labirintul respectiv;
* linia de introducere a comenzilor;

Pentru fiecare mutare se va apasa tasta Enter, se pot tasta și comenzi multiple în linia de introducere a comenzilor , jucătorul se va muta în funcție de comenzile introduse (ex:”ddddsssaaww” sau „w”).

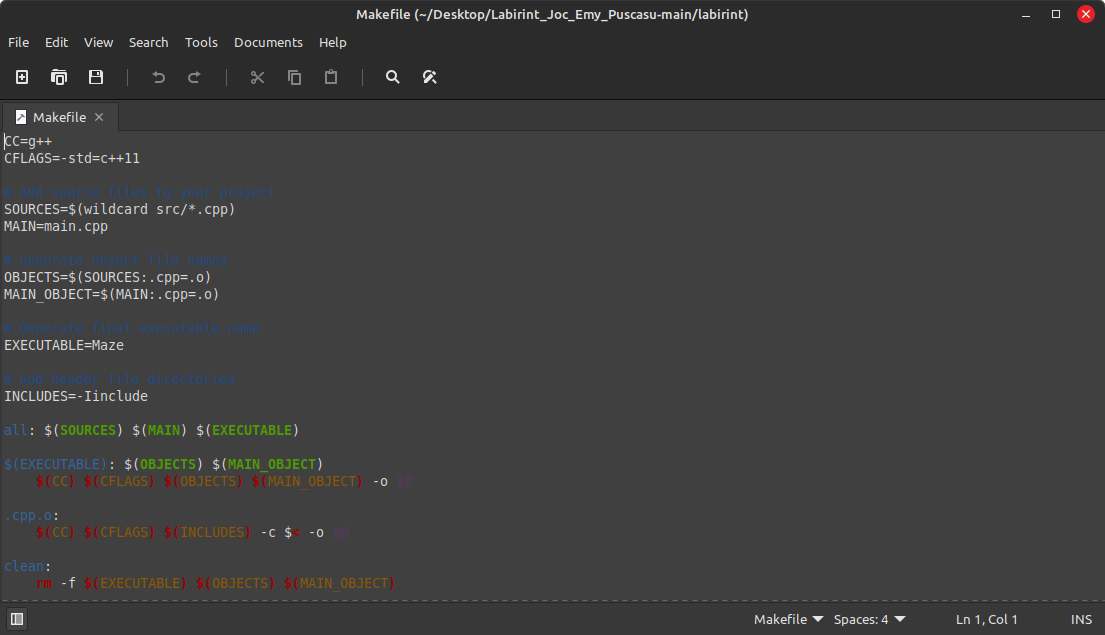
Dacă se va introduce o comanda greșită sau imposibila de executat pentru jucător consola va afisa mesajul „Comanda greșită sau invalida!” iar dacă Player-ul ajunge la destinația marcata „E” jocul va lua sfârșit.

La fiecare sfârșit de joc, se va afisa o optiune în care jucătorul poate alege dacă vrea sa încerce din nou un alt nivel.



## Makefile

Pentru a executa programul direct din terminal am implementat în folderul proiect un document de tip Makefile care are rolul de a compila programul și de a genera un fisier executabil.



Pentru a crea executabilul Maze pentru rularea programului va trebui sa navigati cu terminalul la locația fisierului proiect și sa tastati comanda make și după comanda ./Maze pentru executare.

\*pentru alte informații accesati ReadMe.txt

# Webografie

[https://rpubs.com/mstefan-rpubs/maze1](https://l.facebook.com/l.php?u=https%3A%2F%2Frpubs.com%2Fmstefan-rpubs%2Fmaze1%3Ffbclid%3DIwAR3GS_1OszFEZ3EXddI1FAIAcjjFn9nmbvpfoF2SPZO59dwOetl1ueMD0TE&h=AT2WYzMDG5C_AoCVeTaJZgXjxFKyHVeJkahvGoN6EWMRYkXq_yVYN73gqSNMyu30EjIJ-MNDFh7fnlhCLXjJzOkYsv3vbx55zXnUGF-37MlghLsTOZyBjqX6Jh3-Sa9kxE5pSC5zNSdNuw)

[https://www.geeksforgeeks.org/rand-and-srand-in-ccpp/](https://l.facebook.com/l.php?u=https%3A%2F%2Fwww.geeksforgeeks.org%2Frand-and-srand-in-ccpp%2F%3Ffbclid%3DIwAR2ExNWot3-eoDvssTL9P7A_TaBMn08HjhXA6l09Z1z8BnKqGrwsfsT17RQ&h=AT2WYzMDG5C_AoCVeTaJZgXjxFKyHVeJkahvGoN6EWMRYkXq_yVYN73gqSNMyu30EjIJ-MNDFh7fnlhCLXjJzOkYsv3vbx55zXnUGF-37MlghLsTOZyBjqX6Jh3-Sa9kxE5pSC5zNSdNuw)

[https://www.geeksforgeeks.org/stack-in-cpp-stl/](https://www.geeksforgeeks.org/stack-in-cpp-stl/?fbclid=IwAR3omGU4WdQiY3EKqxrzzhBow7fnt2w6WX8pEKG9WA5cW7v0AoRQBJbYZVc)

[https://linuxhint.com/use\_cpp\_vector/](https://linuxhint.com/use_cpp_vector/?fbclid=IwAR1J8F4rjwbOlHzcyN_62ZQ5dmxokR9weqZlc1pSbo0amkUEuvyoxOYVyGM)

[https://linuxhint.com/vector-of-vectors-cpp/](https://linuxhint.com/vector-of-vectors-cpp/?fbclid=IwAR1lGi186_ER8hT5C0HFXUamRhJMZBRXaeb87AE8qlXHaEkNjmEZQ_PJEK0)

[https://cplusplus.com/reference/ctime](https://cplusplus.com/reference/ctime?fbclid=IwAR2yPkaYmF-ptBhG1dIpm_JYumWCD2JdKnhxJfy8iGjqQxeJDx2kzqli_oQ)