Proiect C++

Fundamentele programarii

Nicoleta Radu

# Hunt the Wumpus

## Descriere

Hunt the Wumpus este un [joc de aventură](https://ro.wikipedia.org/wiki/Joc_de_aventur%C4%83) bazat pe text dezvoltat de [Gregory Yob](https://ro.wikipedia.org/w/index.php?title=Gregory_Yob&action=edit&redlink=1) în [1973](https://ro.wikipedia.org/wiki/1973). În joc, jucătorul trece printr-o serie de peșteri conectate, aranjate într-un [dodecaedru](https://ro.wikipedia.org/wiki/Dodecaedru), în timp ce vânează un monstru numit [Wumpus](https://ro.wikipedia.org/wiki/Wumpus). În acest [joc bazat pe ture](https://ro.wikipedia.org/wiki/Joc_video_de_strategie_pe_ture) jucătorul trebuie să evite gropile fatale și super-liliecii care se mișcă în jurul sistemului de peșteri. Scopul este de a trage cu una dintre "săgețile lor strâmbe" prin peșteri pentru a-l ucide pe Wumpus.

Peșterile sunt complet întunecate, astfel încât jucătorul nu poate vedea în peșterile adiacente; în schimb, când se deplasează într-o nouă peșteră goală, jocul descrie dacă poate mirosi un Wumpus, auzi un liliac sau simți un curent de aer dintr-o groapă într-una dintre peșterile conectate. Intrarea într-o peșteră cu o groapă pune capăt jocului din cauza căderii jucătorului în ea, în timp ce intrarea în peștera cu Wumpus îl sperie; Wumpus se va muta în altă peșteră sau va rămâne și va ucide jucătorul. Dacă jucătorul alege să tragă o săgeată si dacă îl lovește pe Wumpus, acesta câștigă. Dacă săgeata nu lovește nimic, atunci Wumpus se sperie și se poate muta într-o peșteră nouă; spre deosebire de jucător, Wumpus nu este afectat de super lilieci sau gropi. Dacă Wumpus se mută în locația jucătorului, acesta pierde.

## Text Flow-chart

1. Prezinta poziția actuală.

1.1 prezintă mesajele de mediu, dacă există.

2. Prezinta mai multe pozitii în care să se poata deplasa jucatorul.

3. 3. Solicită utilizatorului acțiunea preferată (tragere sau deplasare).

3.1 Dacă se trage

🡪 cere utilizatorului sa specifice pestera prin care să tragă.

🡪 Daca Wumpus este nimerit, jucatorul castiga.

🡪 Daca Wumpus nu este nimerit, acesta isi schimba pozitia aleatoriu in matrice.

🡪 afișează mesajul corespunzător după acțiune.

3.2 Dacă se mută

🡪 solicită utilizatorului următoarea peșteră/pozitie.

* Daca Wumpus se afla pe pozitia respectiva, jucatorul pierde.
* Daca liliecii se afla pe pozitia respectica, jucatorul este mutat la o pozitie noua aleatorie
* Daca gaura se afla pe pozitia respectiva, jucatorul pierde.

🡪 repetă pașii 1 – 3.

## Cod

// HuntTheWumpus.cpp : This file contains the 'main' function. Program execution begins and ends there.

//

#include <iostream>

#include <vector>

#include <random>

#include <cstdlib>

using namespace std;

void printMatrix(vector<vector <int>> yourVector)

{

for (int i = 0; i < yourVector.size(); i++)

{

for (int j = 0; j < yourVector.size(); j++)

{

cout << yourVector.at(i).at(j) << " ";

}

cout << endl;

}

}

int playerCurrentCave(vector<vector <int>> yourVector, int playerRow, int playerCol)

{

cout << "You are in cave " << yourVector.at(playerRow).at(playerCol);

return yourVector.at(playerRow).at(playerCol);

}

void findAvailablePositions(vector<vector <int>> yourVector, vector <int> &cavePos,int playerRow, int playerCol)

{

cout << "The tunnels lead to the caves ";

int counter{ 0 };

// CavePos is holding the possible caves the player can go to

if (yourVector.at(playerRow - 1).at(playerCol) != 0)

{

cout << yourVector.at(playerRow - 1).at(playerCol) << " ";

cavePos.at(counter) = yourVector.at(playerRow - 1).at(playerCol);

counter = counter + 1;

}

if (yourVector.at(playerRow + 1).at(playerCol) != 0)

{

cout << yourVector.at(playerRow + 1).at(playerCol) << " ";

cavePos.at(counter) = yourVector.at(playerRow + 1).at(playerCol);

counter = counter + 1;

}

if (yourVector.at(playerRow).at(playerCol - 1) != 0)

{

cout << yourVector.at(playerRow).at(playerCol - 1) << " ";

cavePos.at(counter) = yourVector.at(playerRow).at(playerCol - 1);

counter = counter + 1;

}

if (yourVector.at(playerRow).at(playerCol + 1) != 0)

{

cout << yourVector.at(playerRow).at(playerCol + 1) << " ";

cavePos.at(counter) = yourVector.at(playerRow).at(playerCol + 1);

}

}

void playerInputNewCave(int \*newPlayerPosition)

{

cout << "Where would you like to move to? ";

cin >> \*newPlayerPosition;

}

int randomNumberGenerator(int limit)

{

// uniformly-distributed integer random number

// generator that produces non-deterministic

// random numbers.

random\_device rd;

// A Mersenne Twister pseudo-random generator

// of 32-bit numbers with a state size of

// 19937 bits.

mt19937 gen(rd());

// Uniform distribution

uniform\_int\_distribution<> dis(2, limit);

return dis(gen);

}

void updatePlayerPosition(vector<vector <int>> yourVector, int &playerRow, int &playerCol, int value)

{

for (int i = 0; i < yourVector.size(); i++)

{

for (int j = 0; j < yourVector.size(); j++)

{

if (yourVector.at(i).at(j) == value)

{

playerRow = i;

playerCol = j;

}

}

}

}

bool checkMatch(vector <int> yourVector, int value)

{

bool match = false;

for (int i = 0; i < yourVector.size(); i++)

{

if (yourVector.at(i) == value)

{

match = true;

return match;

}

}

return match;

}

bool checkVectorsMatch(vector <int> yourVector1, vector <int> yourVector2)

{

bool isMatch = false;

for (int i = 0; i < yourVector1.size(); i++)

{

for (int j = 0; j < yourVector2.size(); j++)

{

if (yourVector1.at(i) == yourVector2.at(j))

{

isMatch = true;

return isMatch;

}

}

}

return isMatch;

}

int main()

{

// cout << "SYSTEM CHECK: bats values: "<<endl; printArray(bats,bats.size());

// cout << "SYSTEM CHECK: holes values: "<<endl; printArray(holes,holes.size());

// defining player map

int mapRows{ 6 }, mapColumns{ 6 }, startCount{ 1 };

vector <vector <int>> map(mapRows, vector <int>(mapColumns, 0));

for (int i = 1; i < map.size() - 1; i++)

{

for (int j = 1; j < map.size() - 1; j++)

{

map.at(i).at(j) = startCount;

startCount = startCount + 1;

}

}

// defining vector for cave numbers

vector <int> cavePos(4);

// defining the game's: player,wumpus,holes and bats

struct player

{

string name{};

int position{};

int health{};

char action{};

int arrows{};

int row{};

int col{};

} player;

struct wumpus

{

int position{};

int health{};

} wumpus;

struct bats

{

int position{};

};

struct holes

{

int position{};

};

// player starting coordinates and stats

player.position = 1;

player.row = 1;

player.col = 1;

player.arrows = 5;

// Wumpus starting coordinates and stats

wumpus.position = randomNumberGenerator(16);

if (wumpus.position == 1 || map.at(1).at(1) == wumpus.position || map.at(1).at(2) == wumpus.position || map.at(2).at(1) == wumpus.position)

{

bool wrongNum = true;

while (wrongNum)

{

wumpus.position = randomNumberGenerator(16);

if (wumpus.position == 1 || map.at(1).at(1) == wumpus.position || map.at(1).at(2) == wumpus.position || map.at(2).at(1) == wumpus.position)

{

bool wrongNum = false;

}

}

}

// Bats coordinates

struct bats bat\_1;

struct bats bat\_2;

bat\_1.position = randomNumberGenerator(16);

bat\_2.position = randomNumberGenerator(16);

vector <int> bats = { bat\_1.position, bat\_2.position };

for (size\_t i = 0; i < bats.size(); i++)

{

if (bats.at(i) == 1 || map.at(1).at(1) == bats.at(i) || map.at(1).at(2) == bats.at(i) || map.at(2).at(1) == bats.at(i))

{

bool wrongNum = true;

while (wrongNum)

{

bats.at(i) = randomNumberGenerator(16);

if (bats.at(i) == 1 || map.at(1).at(1) == bats.at(i) || map.at(1).at(2) == bats.at(i) || map.at(2).at(1) == bats.at(i))

{

bool wrongNum = false;

}

}

}

}

// holes coordinates

struct holes hole\_1;

struct holes hole\_2;

hole\_1.position = randomNumberGenerator(16);

hole\_2.position = randomNumberGenerator(16);

vector <int> holes = { hole\_1.position, hole\_2.position };

for (size\_t i = 0; i < holes.size(); i++)

{

if (holes.at(i) == 1 || map.at(1).at(1) == holes.at(i) || map.at(1).at(2) == holes.at(i) || map.at(2).at(1) == holes.at(i))

{

bool wrongNum = true;

while (wrongNum)

{

holes.at(i) = randomNumberGenerator(16);

if (holes.at(i) == 1 || map.at(1).at(1) == holes.at(i) || map.at(1).at(2) == holes.at(i) || map.at(2).at(1) == holes.at(i))

{

bool wrongNum = false;

}

}

}

}

// game status

bool isRunning = true;

player.action;

int playerShot{};

// Some Lore

cout << "Hello adventurer, what is your name? ";

cin >> player.name;

cout << "Hi " << player.name << "!" << endl;

cout << "Your task is to find and shoot the Wumpus. A creature that has been lurking in these caves for hundreds of years..." << endl;

cout << "All he does is sleep, but if he hears you, he will most likely jump on you .. and well .. I think you know how that ends."<<endl;

cout << "P.S. - Watch out for holes and crazy bats!" << endl;

cout << endl;

cout << endl;

do

{

playerCurrentCave(map, player.row, player.col);

cout << endl;

findAvailablePositions(map,cavePos,player.row, player.col);

cout << endl;

// Environment messages

if (checkMatch(cavePos, wumpus.position) == true)

{

cout << player.name << ": I smell a Wumpus" << endl;

}

if (checkVectorsMatch(cavePos,bats) == true)

{

cout << player.name << ": I think I heard wings batting" << endl;

}

if (checkVectorsMatch(cavePos, holes) == true)

{

cout << player.name << ": I feel a slight draft" << endl;

}

cout << "Shoot an arrow or move(S / M) ? ";

cin >> player.action;

// Player chooses move

if (player.action == 'M' || player.action == 'm')

{

playerInputNewCave(&player.position);

cout << endl;

// check cave choice validity

bool wrongChoice = true;

do

{

if (checkMatch(cavePos,player.position) == false)

{

cout << "You cannot go to that cave!" << endl;

playerInputNewCave(&player.position);

cout << endl;

}

else

{

wrongChoice = false;

}

} while (wrongChoice);

if (player.position == wumpus.position)

{

cout << "OM NOM OM NOM OM NOM ... said the Wumpus" << endl;

cout << "\*\*\* " << player.name << " was eaten by the Wumpus. \*\*\*" << endl;

isRunning = false;

break;

}

if (checkMatch(bats, player.position) == true)

{

cout << player.name << ": Wait ... where are you taking me?!?" << endl;

cout << "\*\*\* " << player.name << " was picked up by angry bats. \*\*\*" << endl;

player.position = randomNumberGenerator(16);

// if bats land player on wumpus location he gets killed

if (player.position == wumpus.position)

{

cout << "OM NOM OM NOM OM NOM ... said the Wumpus" << endl;

cout << "\*\*\* " << player.name << " was eaten by the Wumpus. \*\*\*" << endl;

isRunning = false;

break;

}

}

if (checkMatch(holes, player.position) == true)

{

cout << player.name << ": AAAAAaaaaaaaaaaaaaaaa \*loud crash\*... there was a hole here?" << endl;

cout << "\*\*\* " << player.name << " fell in a botomless pit. \*\*\*" << endl;

isRunning = false;

}

}

// Player chooses to shoot after Wumpus

if (player.action == 'S' || player.action == 's')

{

cout << "In which cave are going to shoot your crooked arrow? ";

cin >> playerShot;

if (playerShot == wumpus.position)

{

cout << "HE HE HE - You got me this time!!! ... said the Wumpus" << endl; // player shoots wumpus

cout << "\*\*\* " << player.name << " has defeated the Wumpus. \*\*\*" << endl;

isRunning = false;

break;

}

else

{

player.arrows = player.arrows - 1;

cout << player.name << ", you have " << player.arrows << " arrows left in your quiver" << endl;

if (player.arrows < 1)

{

cout << "Ooops! It seems like you are out of arrows! You cannot hunt the Wumpus now." << endl;

isRunning = false;

break;

}// check for amount of arrows left

wumpus.position = randomNumberGenerator(16); // reset Wumpus position if player misses shot

}

}

updatePlayerPosition(map, player.row, player.col, player.position);

cout << endl;

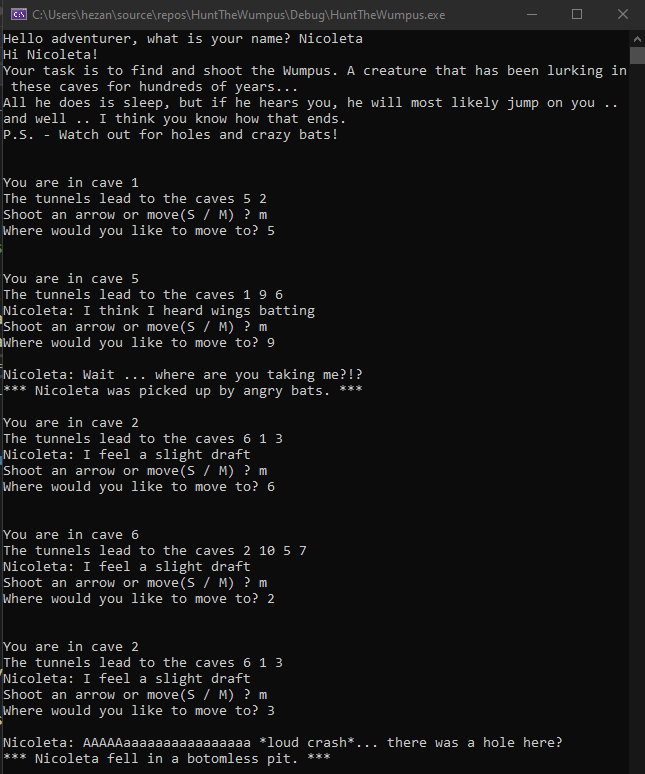
} while (isRunning);

system("pause");

return 0;

}

## Capturi de ecran



# HangMan

## Descriere

Spânzurătoarea este un [joc](https://ro.wikipedia.org/wiki/Joc) pe [hârtie](https://ro.wikipedia.org/wiki/H%C3%A2rtie) pentru doi jucători. Unul din jucători se gândește la un [cuvânt](https://ro.wikipedia.org/wiki/Cuv%C3%A2nt), iar celălalt încearcă să-l ghicească sugerând [litere](https://ro.wikipedia.org/wiki/Liter%C4%83).

Cuvântul ce trebuie ghicit este reprezentat de un șir de linii, fiecare linie reprezentând o literă a cuvântului. Dacă jucătorul care ghicește sugerează o literă ce se află în cuvânt, celălalt jucător o completează în toate pozițiile unde aceasta apare. Dacă litera nu se află în cuvânt, celălalt jucător desenează un element din diagrama „spânzurătoarea”.

Jocul se încheie când:

* Jucătorul care ghicește completează tot cuvântul, sau îl ghicește exact.
* Celălalt jucător completează diagrama.

## Text Flow-chart

1. Retrage linie cu linie cuvintele care trebuiesc ghicite in joc din fisierul text Hang.txt.
2. Salveaza cuvantul prezent intr-un vector (vector origine) de caractere.
3. Prezinta cuvantul care trebuie ghicit intr-un vector (vector-copie), populat cu simbolul underscore (“\_”).
4. Caracterele de pe pozitia 0 si (size) – 1 vor fi literele corespunzatoare cuvantului pentru a oferi un indiciu jucatorului.
5. Solicita jucatorului sa introduca o litera
6. Daca este introdusa una din literele corespunzatoare, updateaza vectorul copie cu noua valoare.
   * 1. Daca vectorul-origine si vectorul-copie sunt identici, jucatorul castiga.
        + 1. Intreaba jucatorul daca doreste sa continue jocul

Daca raspunsul este da, retrage din Hang.txt urmatorul cuvant.

Daca raspunsul este nu, stop joc.

1. Daca nu este introdusa litera corespunzatoare, o incercare (din cele 5) va fi folosita.
   * 1. Daca jucatorul are incercari ramase, repeat from step 5.
     2. Daca incercarile au ajuns la 0, stop joc.

## Cod

// HangMan.cpp : This file contains the 'main' function. Program execution begins and ends there.

//

#include <iostream>

#include <cstring>

#include <fstream>

#include <string>

#include <ctype.h>

using namespace std;

bool valueMatch(char yourChar[], int size ,char value)

{

bool match = false;

for (size\_t i = 0; i < size; i++)

{

if (yourChar[i] == value)

{

match = true;

return true;

}

}

return match;

}

bool vectorsMatch(char yourChar1[], char yourChar2[], int size)

{

bool match{ false };

for (size\_t i = 0; i < size; i++)

{

if (yourChar1[i] == yourChar2[i])

{

match = true;

}

else

{

match = false;

break;

}

}

return match;

}

void replace(char yourChar1[], char yourChar2[] ,int size, char value)

{

for (size\_t i = 1; i < size-1; i++)

{

if (yourChar1[i] == value)

{

yourChar2[i] = yourChar1[i];

}

}

}

void populateTestArray(char yourChar1[], char yourChar2[], int size)

{

for (size\_t i = 0; i < size; i++)

{

yourChar2[i] = '\_';

}

yourChar2[0] = yourChar1[0];

yourChar2[size - 1] = yourChar1[size - 1];

}

void printArray(char yourChar[], int size)

{

for (size\_t i = 0; i < size; i++)

{

cout << yourChar[i] << " ";

}

cout << endl;

}

void emptyArray(char yourChar[], int size)

{

for (size\_t i = 0; i < size; i++)

{

yourChar[i] = '\0';

}

}

int main()

{

ifstream hang("hangman.txt");

if (!hang)

{

cout << "Operatie de deschidere esuata" << endl;

}

char word[15];

hang.getline(word, 15);

int wordSize{0};

wordSize = strlen(word);

cout << "Hi! welcome to HangMan." << endl;

cout << "Guess the word! :D" << endl;

char \*verifyWord = new char[wordSize];

populateTestArray(word,verifyWord, wordSize);

char playerChoice{};

int tries{5};

char continueGame{};

bool isRunning{ true };

do

{

printArray(verifyWord, wordSize);

cout << "enter a letter: ";

cin >> playerChoice;

if (valueMatch(word,wordSize,playerChoice))

{

replace(word, verifyWord, wordSize, playerChoice);

}

else

{

tries = tries - 1;

cout << "you have " << tries << " tries left!" << endl;

if (tries == 0)

{

isRunning = false;

break;

}

}

if (vectorsMatch(word, verifyWord, wordSize))

{

cout << "Congrats! You guessed the word!" << endl;

printArray(verifyWord, wordSize);

cout << endl;

tries = 5;

cout << "Want to try again? Y/N: ";

cin >> continueGame;

//get new word or quit the game

if (continueGame == 'Y' || continueGame == 'y')

{

emptyArray(word, 15);

hang.getline(word, 15);

wordSize = strlen(word);

emptyArray(verifyWord, 15);

populateTestArray(word, verifyWord, wordSize);

cout << endl;

}

else

{

isRunning = false;

break;

}

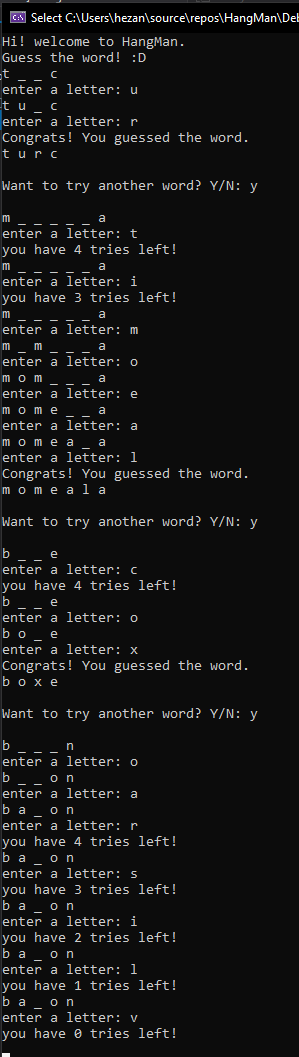
}

} while (isRunning == true || tries != 0);

return 0;

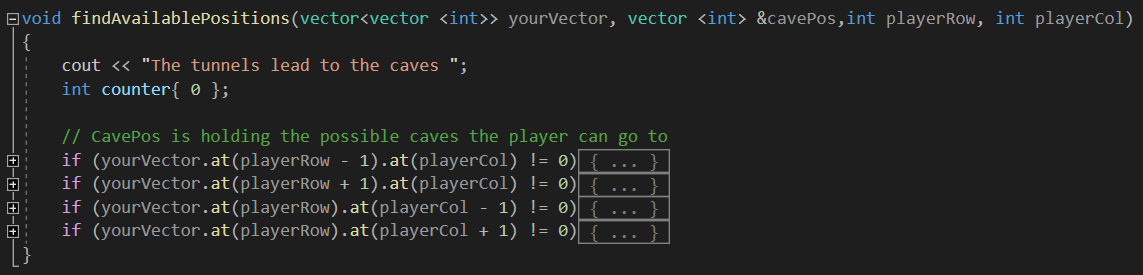
}

## Capturi de ecran

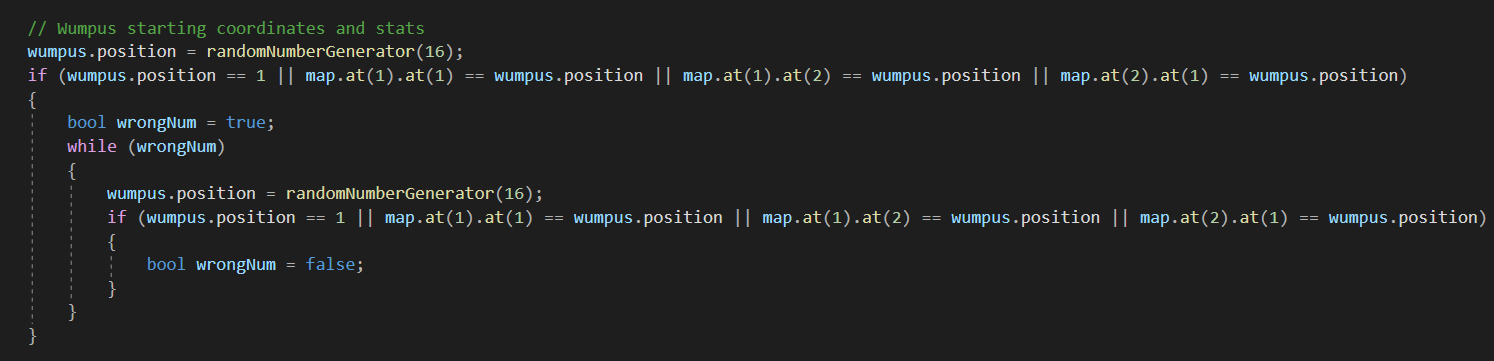


# Algoritmi folositi

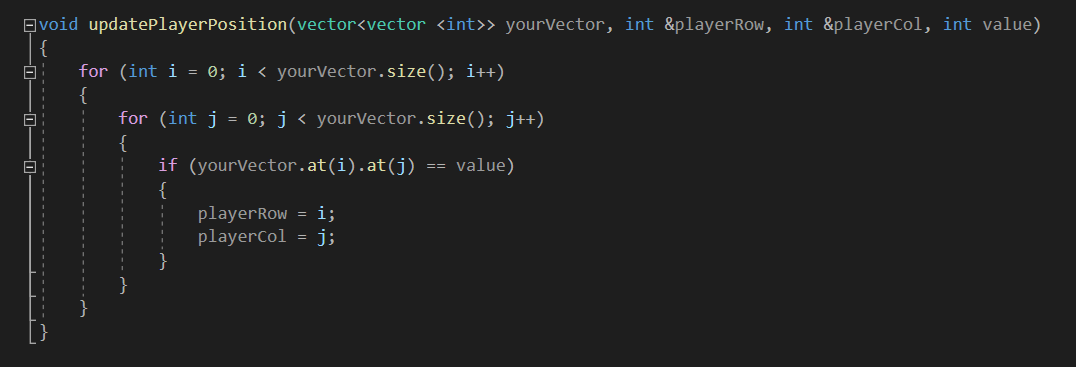
1. Elemente fundamentale ale limbajului c++ (operatori, expresii,variabile)

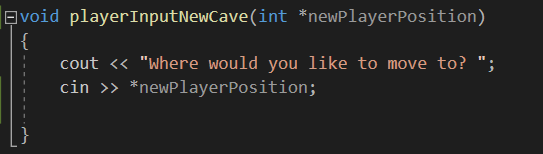


1. Instructuni ale limbajului (while,if)

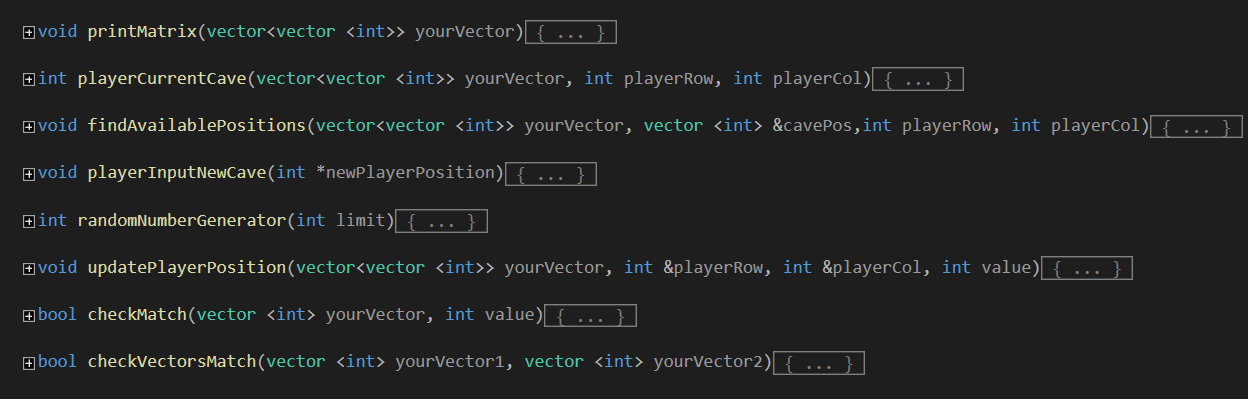
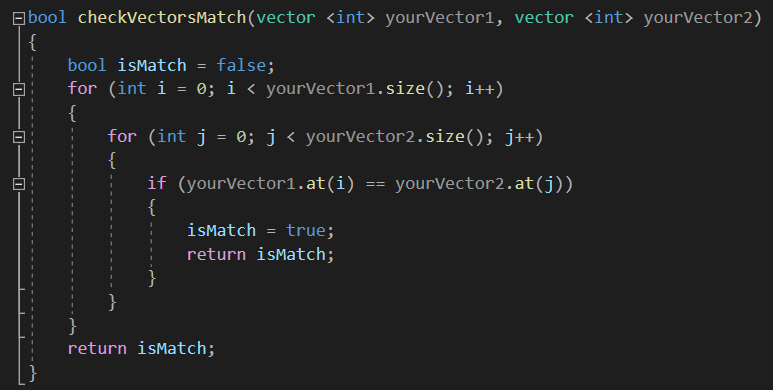


1. Pointeri si referinte

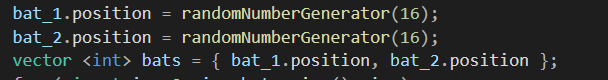


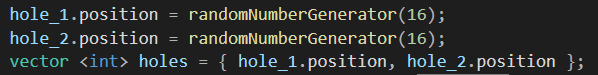


1. Functii definite de utlizator



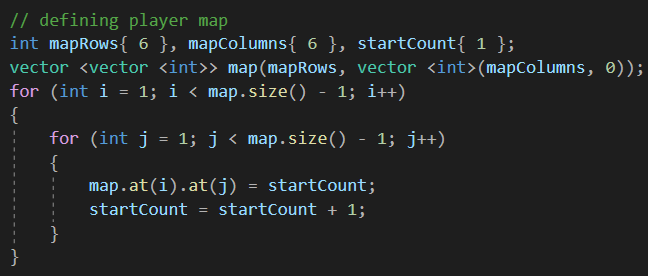
1. Array-uri



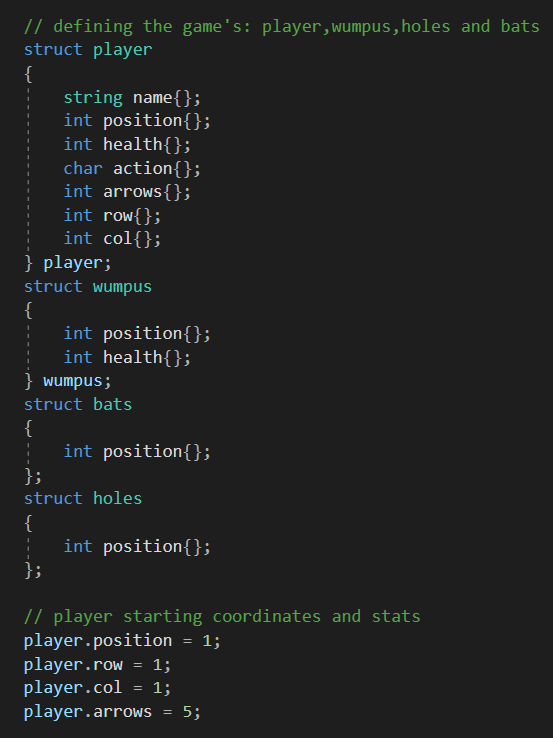




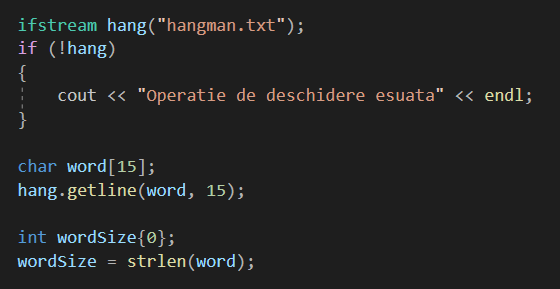
1. Matrice multi-dimensionale



1. Inregistrati (structuri si uniuni)



1. Fisiere



1. Siruri de caractere

