

**Universitatea Tehnică de Construcții din București**

**Facultatea de Hidrotehnică**

**Specializarea: Automatică și Informatică Aplicată**

**Proiect la LP2**

**Profesor coordonator: Student:**

**Olteanu Gabriela**  **Chiosa Rares**

**Rosca Remus**

**București, 2024**



**Universitatea Tehnică de Construcții din București**

**Facultatea de Hidrotehnică**

**Specializarea: Automatică și Informatică Aplicată**

**Pac-Man**

**Profesor coordonator: Student:**

**Olteanu Gabriela**  **Chiosa Rares**

**Rosca Remus**

**București, 2024**

**CUPRINS**

[Introducere 3](#_Toc361179909)

[Motivarea alegerii temei 3](#_Toc124504799)

[Obiectivele propuse in cadrul lucrarii 3](#_Toc1157141346)

[Structura 5](#_Toc1642432737)

[1. Aspecte teoretice referitoare la jocului de tip pac-man. 6](#_Toc869945221)

[1. Introducere în conceptul de jocuri arcade 6](#_Toc1200457038)

[2. Elemente cheie în jocurile de tip Pac-Man 7](#_Toc1336207820)

[3. Evoluția jocurilor Pac-Man-like 8](#_Toc924205518)

[4. Elemente comune în jocurile moderne cu influențe Pac-Man 8](#_Toc1870482724)

[5. Caz special: The Binding of Isaac – o reinterpretare modernă a conceptului 8](#_Toc1109080955)

[Concluzie 10](#_Toc14516593)

[2.1 code blocks 12](#_Toc487962349)

[2.2 C++ 12](#_Toc1930681293)

[Concluzii si contributii personale 15](#_Toc1675125380)

[Concluzii 15](#_Toc1517492541)

[Contributii personale 16](#_Toc2067796964)

[Perspective de viitor 17](#_Toc384931013)

[Bibliografie 17](#_Toc224626345)

# Introducere

## Motivarea alegerii temei

Am ales tema *"Implementarea jocului Pac-Man în C++"* deoarece îmbină într-un mod interesant conceptele fundamentale de programare cu dezvoltarea de aplicații interactive. Pac-Man este un joc clasic, bine cunoscut, ceea ce oferă un bun punct de plecare .

Prin această temă, ne-am propus să aprofundam cunoștințele de programare orientată pe obiect în C++, lucrul cu structuri de date, gestionarea coliziunilor, crearea unei interfețe grafice și înțelegerea mecanismelor de bază ale inteligenței artificiale pentru comportamentul inamicilor (fantomele).Totodata am incercat sa ne depasim limitele incercand sa realizam acest fara un enviroment prestabilit precum (UnrealEngine5) si am optat pentru o biblioteca care sa ajute compilatorul in gestionarea taskurilor specifice(SFML 3.0)

De asemenea, dezvoltarea unui joc presupune o abordare complexă și interdisciplinară, implicând atât gândire algoritmică, cât și creativitate în proiectarea nivelurilor, a regulilor jocului și a interfeței vizuale. Alegerea acestei teme reflectă interesul personal față de domeniul dezvoltării de jocuri și dorința de a ne aplica într-un mod practic noțiunile teoretice învățate.

## Obiectivele propuse in cadrul lucrarii

In cadrul dezvoltarii jocului cu titlul "Pac-Man", ne-am propus sa realizam mai multe module capabile sa indeplineasca obiective precum: Proiectarea și implementarea unei structuri logice clare a jocului, Reprezentarea grafică a elementelor jocului, Gestionarea inputului din partea utilizatorului, Implementarea mecanismelor de coliziune și reguli de joc, Dezvoltarea unei logici simple de inteligență artificială, Realizarea unei interfețe de utilizator prietenoase, Testarea și optimizarea jocului, Documentarea completă a codului sursă și a pașilor de implementare, etc.

**Proiectarea și implementarea unei structuri logice clare a jocului, bazată pe programarea orientată pe obiect (OOP)**  
 Se urmărește organizarea codului în clase bine delimitate, precum PacMan, Fantoma, Labirint, Joc, ș.a., fiecare având atribute și metode relevante. Această abordare facilitează gestionarea complexității, reutilizarea codului și extinderea aplicației. De exemplu, toate entitățile mobile (Pac-Man și fantomele) pot moșteni o clasă de bază Entitate, care definește comportamente comune.

**Reprezentarea grafică a elementelor jocului folosind o bibliotecă grafică**  
 Jocul va utiliza o bibliotecă precum **SFML (Simple and Fast Multimedia Library)** pentru a desena pe ecran labirintul, personajele, punctele și elementele de interfață. Se vor încărca și afișa sprite-uri, se va gestiona fereastra de joc, precum și actualizarea grafică în funcție de logica jocului. Grafica este esențială pentru o experiență de joc plăcută și intuitivă.

**Gestionarea inputului din partea utilizatorului**  
 Sistemul de control va permite utilizatorului să miște personajul principal folosind tastele direcționale (săgeți sau W/A/S/D). Este necesară o implementare receptivă și fără întârzieri, astfel încât reacțiile lui Pac-Man la comenzi să fie rapide și naturale. Inputul trebuie corelat cu starea jocului și blocat în anumite situații (e.g., la pauză sau după terminarea jocului).

**Implementarea mecanismelor de coliziune și reguli de joc**  
 Se vor implementa algoritmi care detectează coliziunile dintre Pac-Man și diverse obiecte: pereți, puncte, fantome sau power-up-uri. De asemenea, vor fi definite regulile de bază ale jocului: pierderea unei vieți în contact cu o fantomă, activarea modului "super" atunci când este colectat un power-up, câștigarea punctelor și condițiile de câștig/pierzare a jocului.

**Dezvoltarea unei logici simple de inteligență artificială pentru comportamentul fantomelor**  
 Fiecare fantomă va avea un comportament specific: urmărirea directă a jucătorului, deplasarea aleatorie sau evitarea jucătorului atunci când este activ un power-up. Se vor implementa algoritmi de deplasare care să simuleze aceste comportamente, folosind direcții valide în labirint și evitând pereții. Astfel, se asigură o dificultate echilibrată și un gameplay dinamic.

**Realizarea unei interfețe de utilizator prietenoase**  
 Interfața grafică va afișa în mod constant scorul, numărul de vieți rămase și nivelul curent. De asemenea, jocul va include un ecran de start (cu instrucțiuni sau buton de „Start”), un ecran de pauză și un ecran final care indică victoria sau eșecul. Toate aceste componente vor fi integrate într-un mod intuitiv și estetic.

**Testarea și optimizarea jocului**  
 Aplicația va fi testată în diferite scenarii pentru a identifica și corecta eventualele erori logice sau grafice. De asemenea, se va acorda atenție optimizării performanței, pentru a asigura o rulare fluidă chiar și pe sisteme cu resurse limitate. Vor fi testate coliziunile, mișcările, scorul, tranzițiile între stări și comportamentele AI.

**Documentarea completă a codului sursă și a pașilor de implementare**  
 Pe parcursul dezvoltării, fiecare componentă a codului va fi documentată cu comentarii explicative, iar în lucrarea scrisă vor fi descrise etapele de implementare, deciziile tehnice, dificultățile întâmpinate și soluțiile aplicate. Această documentare va permite o înțelegere clară a proiectului atât pentru evaluare, cât și pentru posibile îmbunătățiri viitoare.

## Structura

Lucrarea este compusă din 5 capitole şi un capitol ce concretizează concluziile, contribuțiile şi perspectivele de dezvoltare viitoare. Proiectul prezintă contribuțiile cu privire la implementarea si dezvoltarea unui joc cu ajutorul bibliotecilor si programelor ajutatoare.

In cadrul primului capitol am prezentat conceptul de joc de tip Pac-Man, unde este intalnit, de ce este benefic, unde si cine il poate folosi si ce beneficii aduce folosirea acestuia in cadrul proiectelor mici precum si in dezvoltatrea ideii in proiecte mai mari precum (Biding of Isac).

In cel de al doilea capitol am descris succint tehnologiile pe care le-am folosit pentru dezvoltarea aplicatiei si limbajele de programare folosite in scrierea acesteia. Am folosit programe precum CodeBlocks 16.01, dar si limbaje precum C++.

In cadrul capitolului trei am prezentat modul de functionare al fiecarui modul cuprins in aplicatie, astfel explicand care este rolul modulelor si cum isi aduc acestea aportul pentru buna functionare a aplicatiei.

Capitolul patru este o scurta prezentare al modului in care am folosit liobraria SFML in detrimentul unui enviroment pentru creerea unui joc precum (Unity,UnrealEngine,CryEngine3 sau Cocos 2d).

In ultimul capitol am descris aplicatia in sine, modul in care aceasta se configureaza, cum se foloseste si am prezentat succint un scenariu de desfasurare al jocului, precum si dezvoltatrea ideii de joc pac in proiecte mai mari precum (Biding of Isac).

# 1. Aspecte teoretice referitoare la jocului de tip pac-man.

### **1. Introducere în conceptul de jocuri arcade**

Jocurile video au evoluat considerabil de la primele apariții în anii ’70 și ’80, însă multe dintre principiile fundamentale au fost trasate de jocuri simple, dar extrem de influente. Unul dintre cele mai reprezentative titluri ale epocii arcade este *Pac-Man*, lansat în 1980 de compania japoneză Namco. Acest joc a devenit rapid un fenomen cultural, fiind considerat nu doar un simbol al epocii arcade, ci și un pionier în domeniul inteligenței artificiale și al designului de nivel.

Jocurile de tip *Pac-Man* sunt caracterizate, în esență, prin explorarea unui spațiu închis (labirint), evitarea sau confruntarea cu inamici, colectarea de obiecte, precum și utilizarea de power-up-uri temporare. Simplitatea lor este doar aparentă: în spatele unui gameplay accesibil se ascund mecanici de joc bine calibrate, design de nivel atent planificat și o logică de IA surprinzător de complexă pentru acea perioadă.

### **2. Elemente cheie în jocurile de tip Pac-Man**

#### **Structura labirintului**

Majoritatea jocurilor inspirate de *Pac-Man* se desfășoară într-un spațiu bidimensional sub formă de labirint. Acest spațiu este compus din coridoare înguste, intersecții și zone de risc, care obligă jucătorul să ia decizii rapide. Labirintul este în același timp un spațiu de explorare și de capcană, contribuind la dinamica tensionată a jocului.

#### **Obiectivul principal – colectarea**

În *Pac-Man*, obiectivul este simplu: colectarea tuturor punctelor din labirint, fără a fi prins de fantome. Acest tip de obiectiv se regăsește și în multe alte jocuri, fiind ușor de înțeles, dar dificil de atins pe măsură ce dificultatea crește. Colectarea este deseori asociată cu recompense, scoruri sau deblocarea următorului nivel.

#### **Inamicii controlați de AI**

Fantomele din *Pac-Man* sunt considerate un exemplu timpuriu de implementare a inteligenței artificiale în jocuri. Fiecare dintre cele patru fantome are un comportament distinct – de la urmărirea directă a jucătorului, la blocarea căilor de acces. Această variație face jocul imprevizibil și captivant.

#### **Power-up-uri și mecanici temporare**

Un element definitoriu în jocurile de tip *Pac-Man* îl reprezintă power-up-urile. Acestea modifică temporar regulile jocului, oferindu-i jucătorului un avantaj (ex: abilitatea de a mânca fantomele, viteze crescute, invincibilitate). Ele creează un ritm dinamic, în care perioadele de vulnerabilitate alternează cu cele de superioritate.

### **3. Evoluția jocurilor Pac-Man-like**

De-a lungul anilor, conceptul de *Pac-Man* a fost reinterpretat în numeroase moduri. Jocuri precum *Ms. Pac-Man* au adus îmbunătățiri semnificative în ceea ce privește AI-ul și varietatea nivelurilor. Ulterior, variante moderne precum *Pac-Man Championship Edition* sau chiar titluri 3D inspirate de mecanica de bază au transformat experiența inițială în forme mult mai complexe.

De asemenea, unele jocuri au preluat doar idei parțiale – precum structura labirintului sau ideea de urmărire – integrându-le într-un gameplay total diferit. Aceste reinterpretări demonstrează flexibilitatea conceptului și cât de mult pot evolua mecanici simple în funcție de context și tehnologie.

### **4. Elemente comune în jocurile moderne cu influențe Pac-Man**

În jocurile moderne inspirate din *Pac-Man*, putem identifica mai multe elemente preluate și adaptate:

**Niveluri labirintice**, fie în 2D, fie în 3D, care necesită explorare strategică.

**Navigare în timp real**, unde deciziile rapide și reflexele sunt cruciale.

**Colectarea de obiecte** ca mecanică principală sau secundară.

**Inamici inteligenți**, care reacționează diferit în funcție de comportamentul jucătorului.

**Power-up-uri**, care modifică echilibrul de forțe pentru perioade scurte.

### **5. Caz special: *The Binding of Isaac* – o reinterpretare modernă a conceptului**

*The Binding of Isaac* este un joc indie lansat în 2011, dezvoltat de Edmund McMillen. La prima vedere, acest titlu pare foarte diferit de *Pac-Man*: are o temă întunecată, o poveste complexă și elemente de roguelike (niveluri generate procedural, moarte permanentă, itemuri variabile). Totuși, în esență, există mai multe puncte de legătură:

#### **Labirintul**

Fiecare nivel din *The Binding of Isaac* este format din camere interconectate care trebuie explorate. La fel ca în *Pac-Man*, jucătorul este în permanentă mișcare într-un spațiu închis, în căutare de obiecte și evitarea inamicilor.

#### 

#### **Colectarea de obiecte**

Deși obiectele nu sunt la vedere ca în *Pac-Man*, fiecare cameră poate ascunde upgrade-uri, monede sau power-up-uri, care sunt esențiale pentru progres.



#### **c. Inamicii și IA**

Fiecare cameră este plină de inamici cu modele de atac specifice, care necesită adaptare și strategie. Deși nu este vorba de IA avansată, diversitatea inamicilor adaugă complexitate asemănătoare fantomelor din *Pac-Man*.



#### **d. Ritm și tensiune**

Ambele jocuri oferă o combinație de strategie, viteză de reacție și decizii rapide. În *Pac-Man* trebuie să alegi între a colecta un punct sau a evita o fantomă; în *Isaac*, alegerea poate însemna supraviețuire sau moarte.

### **Concluzie**

Chiar dacă *The Binding of Isaac* este mai complex, mai variat și mai întunecat decât *Pac-Man*, el păstrează multe dintre elementele fundamentale ale acestuia: explorarea unui spațiu limitat, colectarea de resurse, evitarea pericolului și lupta pentru supraviețuire. Această continuitate demonstrează cât de mult pot influența jocurile clasice generațiile moderne de jocuri și cât de actuale pot rămâne mecanicile simple dacă sunt adaptate creativ.

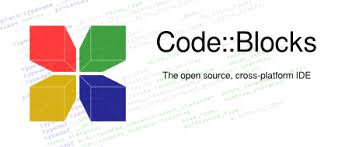
**2.Tenologii folosite in dezvoltarea proiectului**

# 2.1 code blocks

**Code::Blocks** este un mediu de dezvoltare integrat (IDE) gratuit pentru C/C++ și Fortran, creat pentru a satisface cele mai exigente nevoi ale utilizatorilor săi. Este conceput pentru a fi foarte extensibil și complet configurabil.

Construit în jurul unei arhitecturi pe bază de pluginuri, **Code::Blocks** poate fi extins prin instalarea de pluginuri. Orice tip de funcționalitate poate fi adăugată prin instalarea sau crearea unui plugin. De exemplu, chiar și funcționalitățile de compilare și depanare sunt oferite prin pluginuri!

Dacă ești nou aici, poți citi manualul utilizatorului sau poți vizita Wiki-ul pentru documentație. Și nu uita să vizitezi și să te alături forumurilor noastre pentru a găsi ajutor sau pentru discuții generale despre Code::Blocks.[[1]](#footnote-14929)

1

# 2.2 C++

**Limbajul C++** a fost dezvoltat de **Bjarne Stroustrup** în anul **1979** prin extinderea limbajului C pentru a include caracteristici precum **moștenirea, polimorfismul, abstractizarea** etc. Este unul dintre cele mai populare limbaje de programare, având o gamă largă de aplicații în diverse industrii.

Limbajul de programare C++ este unul dintre cele mai răspândite limbaje existente, fiind utilizat de peste **patru decenii**. Dar știi care este **istoria limbajului C++** și cum a evoluat de-a lungul timpului? Dacă nu, atunci cu siguranță urmează să afli lucruri interesante!

În acest articol, vom analiza mai îndeaproape **istoria limbajului de programare C++**, originea sa, cine l-a inventat, **importanța** și **domeniile sale de aplicare**. De asemenea, vom oferi o **comparație** între C++ și alte limbaje populare de programare. Așadar, să începem!

**C++** este un limbaj de programare **generalist, de nivel mediu**, care a fost inițial dezvoltat de **Bjarne Stroustrup**, un informatician danez, în **1979**, la **Bell Laboratories, SUA**. După cum mulți știu, un limbaj de programare de **nivel mediu** combină caracteristici ale limbajelor de **nivel jos** (aproape de hardware) și **nivel înalt** (aproape de limbajul uman).

Limbajul C++ a fost conceput inițial ca o **extensie a limbajului C**, care era deja foarte utilizat în programarea de sistem și în sistemele de operare bazate pe Unix. De asemenea, limbajul C a fost dezvoltat în anii ’70 de către **Dennis Ritchie**, tot la Bell Labs (cunoscut anterior ca AT&T Bell Labs).

De-a lungul anilor, **C++ a devenit unul dintre cele mai utilizate limbaje de programare** și are aplicații variate în domenii precum:

**jocuri video**,

**robotică**,

**finanțe**,

**calcul științific**.

Dezvoltatorii au lansat, de asemenea, **mai multe versiuni noi ale limbajului**, cu modificări considerabile, pentru a ține pasul cu cerințele moderne și pentru a concura cu alte limbaje de programare actuale. Așadar, haideți să aruncăm o privire asupra **cronologiei și versiunilor limbajului C++**.[[2]](#footnote-1697)

[[3]](#footnote-11091)

**4.SFML vs UnrealEngine5\External Engines**

**\**

**3.1 SFML**

**SFML** (Simple and Fast Multimedia Library) este o bibliotecă [multimedia](https://ro.wikipedia.org/wiki/Multimedia) portabilă și ușor de folosit. Este scrisă în [C++](https://ro.wikipedia.org/wiki/C%2B%2B), dar poate fi și portată in [C](https://ro.wikipedia.org/wiki/C), [D](https://ro.wikipedia.org/wiki/D), [Python](https://ro.wikipedia.org/wiki/Python), [Ruby](https://ro.wikipedia.org/wiki/Ruby), [OCaml](https://ro.wikipedia.org/w/index.php?title=OCaml&action=edit&redlink=1), [.NET](https://ro.wikipedia.org/w/index.php?title=.NET&action=edit&redlink=1). Este o alternativă [orientată obiect](https://ro.wikipedia.org/w/index.php?title=Orientat%C4%83_obiect&action=edit&redlink=1) pentru [SDL](https://ro.wikipedia.org/w/index.php?title=SDL&action=edit&redlink=1). SFML este folosit în mod principal pentru jocuri.

SFML pune la dispoziție grafică [2D](https://ro.wikipedia.org/wiki/2D) accelerată prin [hardware](https://ro.wikipedia.org/wiki/Hardware), folosind [OpenGL](https://ro.wikipedia.org/wiki/OpenGL), și câteva module pentru a ușura programarea jocurilor și a aplicațiilor multimedia. SFML poate fi folosit și pentru furnizarea unei ferestre pentru OpenGL. Website-ul SFML oferă întregul [SDK](https://ro.wikipedia.org/wiki/SDK) pentru download și tutoriale pentru a ajuta dezvoltatorii.[[4]](#footnote-20926)

### **De ce să alegi SFML în locul Unreal Engine 5:**

### **1. Simplitate și control direct asupra codului**

SFML este o **bibliotecă C++ simplă**, care îți oferă acces direct la grafica 2D, sunet, evenimente, timp, și ferestre.

Unreal Engine 5 este un motor complex, cu un nivel ridicat de abstractizare, ceea ce poate fi prea mult pentru un proiect de bază sau un joc simplu.

Cu SFML, **tu scrii codul de la zero**, ceea ce te ajută să înțelegi mai bine conceptele fundamentale (cum se desenează un sprite, cum se gestionează inputul, cum funcționează o buclă de joc etc.).

### **2. Ideal pentru învățare și proiecte academice**

Pentru lucrări de **tip atestat sau licență**, SFML este perfect deoarece demonstrează:

stăpânirea limbajului C++,

înțelegerea arhitecturii unui joc,

capacitatea de a lucra cu o bibliotecă grafică fără a te baza pe un motor cu totul gata făcut.

Unreal este util pentru proiecte comerciale mari, dar nu scoate în evidență abilitățile de programare de bază într-un mod atât de clar.

### **3. Performanță și rulare rapidă**

Aplicațiile făcute în SFML sunt **foarte rapide** și **consumă puține resurse**, fiind potrivite pentru calculatoare modeste sau prezentări rapide.

Unreal Engine 5 are cerințe de sistem mari (placă video performantă, RAM, CPU) și poate fi prea „greu” pentru un simplu joc tip *Pac-Man*.

### **4. Dimensiune redusă și portabilitate**

Proiectele SFML sunt **ușor de împachetat și distribuit** – câteva fișiere executabile și DLL-uri.

Proiectele Unreal sunt foarte voluminoase și implică multe fișiere și structuri complexe de directoare.

### **5. Control complet asupra logicii jocului**

Cu SFML, tu decizi cum arată motorul de joc – bucla principală, actualizarea obiectelor, desenarea pe ecran, coliziunile etc.

Unreal are sisteme predefinite (Blueprints, comportamente de actor etc.), dar nu înveți la fel de bine **„ce se întâmplă în spate”**.

### **6. Integrare mai ușoară în proiecte C++ existente**

Dacă lucrezi deja cu C++ pentru logica jocului sau pentru alte module (ex: AI, algoritmi de căutare), SFML se integrează nativ.

Unreal necesită adaptare la sistemul său de tip *Gameplay Framework*, ceea ce poate fi un obstacol.

# Explicatie cod

### **Biblioteci incluse și definirea constantelor**

#include <SFML/Graphics.hpp>  
#include <vector>  
#include <ctime>  
#include <cstdlib>  
  
#define CELL\_SIZE 32  
#define MAP\_WIDTH 20  
#define MAP\_HEIGHT 15

* #include <SFML/Graphics.hpp>: Include toate funcționalitățile grafice din biblioteca SFML.
* #include <vector>: Permite utilizarea tipului de date std::vector (un fel de listă dinamică).
* #include <ctime> și #include <cstdlib>: Sunt folosite pentru generarea numerelor aleatorii.
* #define: Definește constante:
  + CELL\_SIZE — dimensiunea unei celule din hartă în pixeli.
  + MAP\_WIDTH și MAP\_HEIGHT — dimensiunile hărții (20x15 celule).

### **Structuri pentru jucător și inamic**

struct Entity {  
 int x, y;  
 sf::RectangleShape shape;  
  
 Entity(int startX, int startY, sf::Color color) {  
 x = startX;  
 y = startY;  
 shape.setSize(sf::Vector2f(CELL\_SIZE, CELL\_SIZE));  
 shape.setFillColor(color);  
 shape.setPosition(x \* CELL\_SIZE, y \* CELL\_SIZE);  
 }  
  
 void move(int dx, int dy) {  
 x += dx;  
 y += dy;  
 shape.setPosition(x \* CELL\_SIZE, y \* CELL\_SIZE);  
 }  
};

* Entity este o structură care reprezintă un obiect de pe hartă (poate fi jucătorul sau un inamic).
* Are coordonatele (x, y), o formă grafică (shape) și:
  + Un constructor care setează poziția și culoarea.
  + O metodă move(dx, dy) care mută obiectul și actualizează poziția vizuală pe ecran.

### **Funcția main() – Inițializare fereastră și entități**

int main() {  
 sf::RenderWindow window(sf::VideoMode(MAP\_WIDTH \* CELL\_SIZE, MAP\_HEIGHT \* CELL\_SIZE), "Mini Pac-Man");  
 window.setFramerateLimit(10); // Jocul rulează la 10 cadre pe secundă

* Se creează o fereastră de joc cu dimensiunea calculată pe baza numărului de celule și dimensiunii celulei.
* Se limitează rata cadrelor la 10 FPS pentru a face mișcarea mai lentă și controlabilă.

**Inițializarea jucătorului, inamicilor și punctelor**

* // Inițializare jucător  
   Entity player(1, 1, sf::Color::Yellow);  
    
   // Inițializare inamici  
   std::vector<Entity> enemies = {  
   Entity(5, 5, sf::Color::Red),  
   Entity(10, 10, sf::Color::Magenta)  
   };
* Se creează un obiect player de tip Entity la poziția (1, 1) cu culoarea galbenă.
* Se creează doi inamici, într-un vector:
  + Unul la (5, 5) de culoare roșie.
  + Altul la (10, 10) de culoare magenta.
* // Inițializare puncte  
   std::vector<sf::CircleShape> points;  
    
   for (int i = 0; i < MAP\_WIDTH; ++i) {  
   for (int j = 0; j < MAP\_HEIGHT; ++j) {  
   if ((i != player.x || j != player.y) && rand() % 5 == 0) {  
   sf::CircleShape point(CELL\_SIZE / 6);  
   point.setFillColor(sf::Color::White);  
   point.setPosition(i \* CELL\_SIZE + CELL\_SIZE / 3, j \* CELL\_SIZE + CELL\_SIZE / 3);  
   points.push\_back(point);  
   }  
   }  
   }
* Se creează un vector points pentru a stoca punctele pe care jucătorul le va colecta.
* Două for-uri parcurg întreaga hartă (20x15 celule).
* Dacă poziția nu este aceeași cu cea a jucătorului **și** dacă rand() % 5 == 0 (adică o șansă din 5), se creează un punct:
  + sf::CircleShape(CELL\_SIZE / 6) — un cerc mic (dimensiunea e o treime din celulă).
  + Se setează poziția în mijlocul celulei.
  + Se adaugă în lista de puncte.

### **Bucla principală a joculu**

while (window.isOpen()) {

* Aceasta este bucla principală a jocului.
* Se execută cât timp fereastra SFML este deschisă.

### **6. Gestionarea evenimentelor (închidere fereastră)**

sf::Event event;  
 while (window.pollEvent(event)) {  
 if (event.type == sf::Event::Closed)  
 window.close();  
 }

* Se verifică evenimentele (ca apăsarea pe "X").
* Dacă jucătorul închide fereastra, jocul se oprește.

### **7. Mișcarea jucătorului (input de la tastatură)**

if (sf::Keyboard::isKeyPressed(sf::Keyboard::Left) && gameMap[player.x - 1][player.y] != 1)  
 player.move(-1, 0);  
 else if (sf::Keyboard::isKeyPressed(sf::Keyboard::Right) && gameMap[player.x + 1][player.y] != 1)  
 player.move(1, 0);  
 else if (sf::Keyboard::isKeyPressed(sf::Keyboard::Up) && gameMap[player.x][player.y - 1] != 1)  
 player.move(0, -1);  
 else if (sf::Keyboard::isKeyPressed(sf::Keyboard::Down) && gameMap[player.x][player.y + 1] != 1)  
 player.move(0, 1);

* Se verifică dacă se apasă tastele **săgeți** (Left, Right, Up, Down).
* Se verifică și dacă poziția în care vrea să meargă **nu este un perete** (gameMap[x][y] != 1).
* Dacă nu e perete, se mișcă jucătorul folosind metoda move().

### **8. Mișcarea inamicilor (random)**

for (auto& enemy : enemies) {  
 int dx = 0, dy = 0;  
 switch (rand() % 4) {  
 case 0: dx = -1; break;  
 case 1: dx = 1; break;  
 case 2: dy = -1; break;  
 case 3: dy = 1; break;  
 }  
 if (gameMap[enemy.x + dx][enemy.y + dy] != 1)  
 enemy.move(dx, dy);  
 }

* Fiecare inamic se mișcă într-o direcție aleatoare:
  + rand() % 4 alege o direcție (stânga, dreapta, sus, jos).
* Se verifică dacă noua poziție nu este un perete.
* Dacă este liber, inamicul se mișcă acolo.

### **9. Verificare coliziune jucător-inamic**

for (auto& enemy : enemies) {  
 if (player.x == enemy.x && player.y == enemy.y) {  
 window.close(); // Game Over  
 }  
 }

* Se verifică dacă vreun inamic are aceeași poziție cu jucătorul.
* Dacă da, jocul se termină (închide fereastra).

### **10. Colectare puncte**

for (auto it = points.begin(); it != points.end(); ) {  
 int pointX = static\_cast<int>(it->getPosition().x) / CELL\_SIZE;  
 int pointY = static\_cast<int>(it->getPosition().y) / CELL\_SIZE;  
 if (player.x == pointX && player.y == pointY) {  
 it = points.erase(it);  
 } else {  
 ++it;  
 }  
 }

* Se parcurge lista de puncte.
* Pentru fiecare punct, se află în ce celulă se află.
* Dacă jucătorul este pe acea celulă, punctul este **eliminat** (colectat).
* Dacă nu, se trece la următorul.

### **Ștergerea ferestrei și desenarea hărții**

window.clear();

* Se curăță fereastra (șterge tot ce a fost desenat anterior).
* Este necesar pentru a redesena la fiecare cadru (frame).

for (int i = 0; i < MAP\_WIDTH; ++i) {  
 for (int j = 0; j < MAP\_HEIGHT; ++j) {  
 sf::RectangleShape cell(sf::Vector2f(CELL\_SIZE, CELL\_SIZE));  
 cell.setPosition(i \* CELL\_SIZE, j \* CELL\_SIZE);  
 if (gameMap[i][j] == 1) {  
 cell.setFillColor(sf::Color::Blue); // perete  
 window.draw(cell);  
 }  
 }  
 }

* Se parcurge fiecare celulă a hărții.
* Dacă în acea celulă este un **perete** (gameMap[i][j] == 1), se creează un dreptunghi albastru care se desenează.
* Astfel se desenează labirintul.

### **12. Desenarea punctelor**

for (const auto& point : points)  
 window.draw(point);

* Se desenează fiecare punct (care nu a fost colectat).
* Fiecare punct este un sf::CircleShape.

### **13. Desenarea jucătorului**

player.shape.setPosition(player.x \* CELL\_SIZE, player.y \* CELL\_SIZE);  
 window.draw(player.shape);

* Se poziționează jucătorul pe hartă conform poziției sale x, y.
* Apoi se desenează pe ecran.

### **14. Desenarea inamicilor**

for (const auto& enemy : enemies) {  
 enemy.shape.setPosition(enemy.x \* CELL\_SIZE, enemy.y \* CELL\_SIZE);  
 window.draw(enemy.shape);  
 }

* Fiecare inamic este poziționat în funcție de coordonatele sale.
* Se desenează apoi în fereastră.

### **15. Afișarea pe ecran (finalizarea desenului)**

window.display();  
 }

#### **Explicație:**

* După ce toate elementele sunt desenate, window.display() afișează rezultatul în fereastră.
* Fără acest apel, nu s-ar vedea nimic.

# Concluzii si contributii personale

## Concluzii

Prin dezvoltarea acestei aplicatii am atins mai multe obiective propuse in etapa de planificare a acesteia. Primul si cel mai important a fost dezvoltarea modulului de tickete in cadrul caruia utilizatorii de tip client pot introduce sesizari, pot aduce anumite modificari in cadrul celor deja introduse si pot vizualiza si vechile tichete care au fost inchise. Utilizatorii de tip helpdesk vor putea prelua tichete noi, vor putea adauga comentarii, fotografii sau fisiere si vor putea comunica cu cel care a deschis ticketul, iar dupa ce problema va fi solutionata, acesta va inchide ticketul.

Un alt obiectiv important a fost portarea aplicatiei si catre dispozitivele mobile, utilizand tehnologii precum Bootstrap pentru a permite aplicatiei sa isi modifice singura rezolutia si sa afiseze doar informatiile utile in cazul unui ecran mai mic, extinzand astfel numarul de utilizatori pe care aplicatia ii poate avea, oferind si posibilitatea utilizarii acesteia de pe un dispozitiv mobil.

De asemenea am considerat ca este important pentru un viitor utilizator al acestei aplicatii sa aiba posibilitatea de a putea vizualiza cu usurinta rapoarte ale orelor lucrate de catre anagajatii sai in solutionarea ticketelor, dar si o privire de ansamblu asupra tuturor ticketelor in functie de client si starea lor actuala.

Consider ca prin dezvoltarea acestei aplicatii, am adus un aport la tehnologia ce defineste o aplicatie de ticketing, astfel oferind posibilitatea de a adauga poze tichetelor deschise, doar prin copierea si lipirea pozei in cadrul aplicatiei. Un alt aspect important al aplicatiei este folosirea apelurilor AJAX in cadrul aplicatiei, in mod special in cadrul modulului de aplicatii unde atribuirea aplicatilor se face doar prin apasarea unei casute fara ca pagina sa se posteze, astfel creeand o interfata mult mai prietenoasa si usor de folosit de utilizator.

## Contributii personale

In dezvoltarea acestei aplicatii am folosit tehnologii rar intalnite in aplicatiile de ticketing cum ar fi urcarea pozelor in pagina de modificare a ticketului prin copierea si lipirea pozelor direct in ecranul navigatorului, am dezvoltat un sistem de atribuire a aplicatiilor clientului cu ajutorul tehnologiei AJAX, astfel pagina nefiind postata la fiecare modificare asupra aplicatiilor. Implementarea celor doua tipuri de buget permite aplicatiei sa inregistreze tickete care urmaresc o rezolvare rapida a problemelor, dar in acelasi timp permite si inregistrarea anumitor dezvoltari mai ample ale altor aplicatii, de exemplu. In felul acesta, tichetele pot fi monetizate diferit in functie de bugetele in care acestea au fost incadrate. Totodata am creat un modul ce permite traducerea aplicatiei in mai multe limbi, traduceri ce pot fi modificate oricand de catre administrator.

In timpul pe care l-am petrecut dezvoltand aceasta aplicatie, mi-am aprofundat cunostintele in materie de limbaje de programare, invatand astfel limbaje precum C#, JavaScript, JQuery si am avut ocazia de a folosi tehnologii noi, care sunt foarte utlizate in acest domeniu precum AJAX, Bootstrap, MVC.

Pentru dezvoltarea aplicatiei si aducerea acesteia in acest stadiu am lucrat in total 6 luni de zile, cumuland peste 10000 de linii de cod. In acest moment, aplicatia este deja comercializata catre un client final, urmand ca eu sa o dezvolt pe masura ce voi primi si alte sugestii de la utilizatorii acesteia.

## Perspective de viitor

Ca si perspective de dezvoltare ulterioara a aplicatiei, imi doresc ca pe viitor sa dezvolt un modul aditional al aplicatiei care se va numi Licente, modul prin care administratorul principal al aplicatiei sa poata sa genereze fisiere de licentiere a companiilor. Aceste pot beneficia de un numar maxim de utilizatori pe care compania ii va putea inregistra in cadrul acestei aplicatii, care va avea o perioada limitata de utilizare de un an, dupa aceea clientul fiind nevoit sa isi prelungeasca aceasta licenta pentru a continua sa foloseasca sistemul de ticketing. Astfel prin acest sistem imi doresc sa continui dezvoltarea aplicatiei si sa urmaresc primirea unui feedback continuu din partea clientilor ce vor folosi aceasta aplicatie.

# Bibliografie

[1] https://www.codeblocks.org/

[2] https://unstop.com/blog/history-of-cpp

[3] <https://en.m.wikipedia.org/wiki/File:ISO_C%2B%2B_Logo.svg>

1. []

   [↑](#footnote-ref-14929)
2. [↑](#footnote-ref-1697)
3. [↑](#footnote-ref-11091)
4. [↑](#footnote-ref-20926)